Networked Life:
20 Questions and Answers

Mung Chiang
Princeton University
April 2012 Draft

Contents

Preface

You pick up your iPhone while waiting in line at a coffee shop. You Google a not-so-famous actor and get linked to a Wikipedia entry listing his recent movies and popular YouTube clips. You check out user reviews on IMDB and pick one, download that movie on BitTorrent or stream that in Netflix. But suddenly the WiFi logo on your phone is gone and you're on 4G. Video quality starts to degrade a little, but you don't know if it's the video server getting crowded in the cloud or the Internet is congested somewhere. In any case, it costs you $10 per gigabyte, and you decide to stop watching the movie, and instead multitask between sending tweets and calling your friend on Skype, while songs stream from iCloud to your phone. You're happy with the call quality, but get a little irritated when you see there're no new followers on Twitter.

You've got a typical networked life, an online networked life.

And you might wonder how all of these technologies "kind of" work, and why sometimes they don't. Just flip through the table of contents of this book. It's a mixture: some of these questions have well defined formulations and clear answer while others still face a significant gap between the theoretical models and actual practice; a few don't even have widely-accepted problem statements. This book is about formulating and answering these 20 questions.

This book is about the networking technologies we use each day as well as the fundamental ideas in the study of networks. Each question is selected not just for its relevance to our daily lives, but also for the core concepts and key methodologies in the field of networking that are illustrated by its answer. These concepts include aggregation and influence, distributed coordination, feedback control, and strategic equilibrium. And the analytic machineries are based on mathematical languages that people refer to as graph, optimization, game, and learning theories.

This is an undergraduate textbook for a new course at Princeton University: **Networks: Friends, Money, and Bytes**. The course targets primarily juniors in electrical engineering and computer science, but also seniors and beginning graduate students as well as students from mathematics, sciences, economics, and engineering in general. It can be viewed as the second course after the "signals and systems" course that anchors the undergraduate electrical and computer engineering curriculum today.

This book weaves a diverse set of topics you would not normally see under

the same cover into a coherent stream: from Arrow's impossibility and Rawls' fairness to Skype signaling and Clos networks, from collaborative filtering and firefly synchronization to MPEG/RTSP/TCP/IP and WiFi CSMA DCF. This begs a question: "So, what *is* the discipline of this book?", a question that most of the undergraduates simply do *not* care about. Neither does this book: it only wants to address these practical questions, using whatever modeling languages that have been observed to be the most relevant ones so far. Turns out there is a small and coherent set of mathematics we will need, but that's mostly because people have only invented a limited suite of modeling languages.

This is not a typical textbook for another reason. It does not start with general theories as do many books on these subjects, *e.g.*, graph theory, game theory, optimization theory, or abstract concepts like feedback, coordination, and equilibrium. Instead it starts with concrete applications and practical answers, and sticks to them (almost) every step of the way. Theories and generalizations only emerge, as if "accidental by-products", during the process of formulating and answering these questions.

This book, when used as an undergraduate textbook, can be complemented with its website features: `http://www.network20q.com`, including lecture slides, problem solutions, additional questions, further pointers to references, collection of news media coverage of the topics, currency-earning activities, course projects, blogs, tweets, surveys, and student-generated course materials in wiki. We created web features that turn this class into an online social network and a networked economy.

This book can also be used by engineers, technology managers, and pretty much anyone with a keen interest in understanding how social and technological networks work. On many spots, we sacrifice generality for accessibility, and supplement symbolic representation by numerical illustration.

- The first section of each chapter is a "short answer", and it is accessible by most people.
- Then there's a "long answer" section. If you remember differentiation and linear algebra (and occasionally a little bit of integration and basic probability), you can follow all the material there. We take great care to include only those symbols and equations that're really necessary to unambigiously express the ideas.
- The "examples" section contains detailed, numerical examples to reinforce the learning from the "long answer" section.
- Each chapter concludes with a section on "advanced material," which requires the reader to be quite comfortable with symbolic operations and abstract reasoning, but can be skipped without losing the coherence and gist of the book. In the undergraduate course taught at Princeton, almost none of the advanced material is covered. Covering all the advanced material sections would constitute an introductory graduate level course.
- At the end of each chapter, there're 5 homework questions, including easy

drills, essential supplements, and some "out-of-syllabus" explorations. The level of difficulty is indicated on a scale of 1 (easy) to 3 (hard) stars.

- There are also 5 key references per chapter (yes, only 5, in the hope that undergraduates may actually read some of these 5, and my apologies to the authors of thousands of papers and books that could have been cited). These references open the door to many worthwhile further readings, including textbooks, research monographs, and survey articles.

This is a (relatively) thin book. It's a collage of snapshots, not an encyclopedia. It's an appetizer, not an entree. We realize that the majority of readers will not pursue a career specializing in the technical material in this book, so we take every opportunity to delete material that's very interesting to specialists but not essential to this undergraduate course. Each one of these 20 chapters deserves many books for a detailed treatment. We only highlight a few key ideas in the span of about 20 pages per chapter and 80 minutes per lecture. There are many other mathematical languages in the study of networks, many other questions about a networked life, and many other types of networks that we do not have time to cover in one semester. But as the saying goes for a course: "It's more important to uncover than to cover a lot."

This is a book illustrating some pretty big ideas in networking, through 20 questions we can all relate to in our daily lives. Questions that tickle our imagination with surprises and incomplete answers. Questions that I wished I had known how to answer several years ago. Questions that are quickly becoming an essential part of modern education in electrical and computer engineering.

But above all, we hope this book is fun to read.

Acknowledgements

In so many ways I've been enjoying the process of writing this book and creating the new undergraduate course at Princeton University. The best part is that I got to, ironically in light of the content of this book, stay *offline* and focus on learning a few hours a day for several hundred days. I got to digest wonderful books and papers that I didn't get a chance to read before, to think about what're the essential points and simple structures behind the drowning sea of knowledge in my research fields, and to edit and re-edit each sentence I put down on paper. It reminded me of my own sophomore year, one and half decade ago, at Stanford University. I often biked to the surreally beautiful Oval in the morning and dived into books of many kinds, most of which not even remotely related to my majors. As the saying goes, that was a pretty good approximation of paradise.

That paradise usually ends together with the college years. So I have many to thank for granting me a precious opportunity to indulge myself again at this much later stage in life.

- The new course "Networks: Friends, Money, and Bytes" could not have been created without the dedication from its three remarkable TAs: Jiasi Chen, Felix Wong, and Pei-yuan Wu. They did so much more for the course than a "normal" TA experience.
- Many students and postdocs in Princeton's EDGE Lab and EE Department worked with me in creating worked examples: Chris Brinton, Amitabha Ghosh, Sangtae Ha, Joe Jiang, Carlee Joe-Wong, Yiannis Kamitsos, Haris Kremo, Chris Leberknight, Soumya Sen, Arvid Wang, and Michael Wang.
- Princeton students in ELE/COS 381's first offering were brave enough to take a completely new course and contributed in many ways, not the least the class website blogs and course projects. Students in the graduate course ELE539A also helped proofread the book draft and created multiple choice questions.
- Before I even get a chance to advertise the course, some colleagues started planning to offer similar courses at their institutions: Jianwei Huang (CUHK, Hong Kong), Hongseok Kim (Sogang U., Korea), Tian Lan (GWU), Walid Saad (U. Miami), Chee Wei Tan (City U., Hong Kong), Kevin Tang (Cornell), more...
- Over 50 colleagues provided valuable suggestions to the course and the book.

In particular, I've received many detailed comments from Kaiser Fung (Sirius), Victor Glass (NECA), Jason Li (IAI), Jennifer Rexford (Princeton), Keith Ross (NYU Poly), Krishan Sabnani (Bell Labs), Walid Saad (U. Miami), Matthew Salganik (Princeton), Jacob Shapiro (Princeton), and Walter Willinger (AT&T Labs), more...

- Phil Meyler from Cambridge University Press encouraged me to turn the lecture notes into a textbook, and further connected me with a group of enthusiastic staff at CUP.
- This course was in part supported by a grant from the U.S. National Science Foundation, in a program run by Darleen Fisher, for a team consisting of two engineers and two social scientists at Princeton. I'm glad to report that we achieved the proposed educational goals, and did that before the project's official end date.

And my appreciation traces back to many of my teachers. For example, I've had the fortune to be co-advised in my Ph.D. study by Stephen Boyd and Tom Cover, two superb scholars who are also superb teachers. Their textbooks *Convex Optimization* and *Elements of Information Theory* are two towering achievements in engineering education. Read these two books, and you'll experience the definition of "clarity", "accessibility", and "insight". When I was writing research papers with them, Tom would spend many iterations just to get one notation right, and Stephen would even pick out each and every LaTex inconsistency. It was a priviledge to see first-hand how the masters established the benchmarks of technical writing.

Stephen and Tom were also the most effective lecturers in classroom, as was Paul Cohen, from whom I took a math course in my sophomore year. Pulling off the sweatshirt and writing with passion on the blackboard from the first moment he entered the classroom, Paul could put your breadth on hold for 80 minutes. Even better, he forgot to give us a midterm and then gave a week-long, take-home final that the whole class couldn't solve. He made himself available for office hours on-demand to talk about pretty much anything related to math. The course was supposed to be on PDE. He spent just four lectures on that, and then introduced us to 18 different topics that quarter. I've forgotten most of what I learned in that course, but I'll always remember that learning should be fun.

In the same quarter that I took Stephen's and Tom's courses, I also took from Richard Rorty a unique course at Stanford called "From Religion through Philosophy to Literature", which pulled me out of Platonism that I had been increasingly attached to as a teenager. Talking to Rorty drastically sharpened my appreciation of the pitfalls of mistaking representations for reality. A side-benefit of that awakening was a repositioning of my philosophy of science, which propagated to the undercurrents of this book.

Three more inspirations, from those I never met:

- Out of all the biographies I've read, the shortest one, by far, is by Paul Johnson on Churchill. And it's by far the most impactful one. Brevity is power.
- But even a short book feels like infinitely long to the author until it goes to the press. What prevented me from getting paralyzed by procrastination is Terman's approach of writing textbooks while serving as a Dean and then the Provost at Stanford (and creating the whole Silicon Valley model): write one page each day.
- Almost exactly one century ago, my great grandfather, together with his brother, wrote some of the first modern textbooks in China on algebra and on astronomy. (And three decades ago, my grandfather wrote a textbook on econometrics at the age of seventy.) As I was writing this book, sometimes I couldn't help but picture the days and nights that they spent writing theirs.

For some reason, the many time commitments of a professor are often hard to compress. And I couldn't afford to cut back on sleep, for otherwise the proportion of garbage in this book would have been even higher. So it's probably fair to say that each hour I spent writing this book has been an hour of family time lost. Has that been a good tradeoff? Definitely not. So I'm glad that the book is done, and I'm grateful to my family for making that happen: my parents who helped take care of my toddler daughter when I was off to dwell in this book, my wife who supported me sitting there staring at my study's ceiling despite her more important job of curing the ill, and Novia who could have played with her Daddy a lot more in the past year. This book was written with my pen and their time.

Roadmap

This roadmap is written for course instructors, or as an *epilogue* for students who have already finished reading the book. It starts with a taxonomy of the book and introduces its organization and notation. Then it discusses the similarities and differences between this book and some excellent, related books published over the last decade. Then it highlights three pedagogical principles guiding the book: Just In Time, Bridge Theory and Practice, and Book As a Network, and two contexts: the importance of domain-specific functionalities in network science and the need for undergraduate curriculum evolution in electrical and computer engineering. It concludes with anecdotes of arranging this course as a social and economic network itself.

Taxonomy and Organization

The target audience of this book are both students and engineering professional. For students, the primary audience are those from engineering, science, economics, operations research and applied mathematics, but also those on the quantitative side of sociology and psychology.

There are three ways to use this book as a textbook:

- *Undergraduate general course at sophomore or junior level*: Go through all 20 chapters without Advanced Material sections. This course serves as an introduction to networks before going further into senior level courses in four possible directions: computer networking, wireless communication, social networks, or network economics.
- *Undergraduate specialized course at senior level*: Pick either the social and economic network chapters or the technology and economic network chapters, and go through Advanced Material sections in those chapters.
- *First year graduate level*: Go through all 20 chapters including Advanced Material sections.

While this book consists of 20 chapters, there are just 4 key recurring concepts underlying this array of topics. Table 0.1 summarizes the mapping from chapter number to the concept it illustrates.

Table 0.1 Key Concepts: The chapters where each of the four key concepts show up for different types of networks.

Network Type	Aggregation & Influence	Distributed Coordination	Feedback Control	Strategic Equilibrium
Wireless		1	19	
Internet		10, 13, 16	14	
Content Distribution		15, 17	18	
Web	3, 4, 5			2
Online Social	6,8	9	7	
Internet Economics		20		11, 12

The modeling language and analysis machinery come from quite a few fields in applied mathematics, especially the four foundations summarized in Table 0.2.

Table 0.2 Main Methodologies: The chapters where each of the four families of mathematical languages are used in different types of networks.

Network Type	Graph Theory	Optimization Theory	Game Theory	Learning Theory
Wireless		18, 19	1	
Internet	10	13,14,16		
Content Distribution		15, 17		
Web	3		2	4,5
Online Social	7,8,9		6	
Internet Economics		11	20	12

The order of appearance of these 20 questions is arranged so that clusters of highly related topics show up next to each other. Therefore, we recommend going through the chapters in this sequence, unless you're OK with flipping back every now and then when key concepts from prior chapters are referenced. Figure 0.1 summarizes the "prerequisite" relationship among the chapters.

This book cuts across both networks among devices and networks among people. We examine networks among people that overlay on top of networks among devices, but also spend half of the book on wireless networks, content distribution networks, and the Internet itself. We'll illustrate important ideas and useful methodologies across both types of networks. We'll see striking parallels in the underlying analytic models, but also crucial differences due to domain-specific details.

We can also classify the 20 questions into three groups based on the stages of development in formulating and answering them:

- Question well formulated, and theory-inspired answers adopted in practice: 1, 2, 3 4, 9, 10, 11, 13 14, 15, 16, 17, 18, 19.

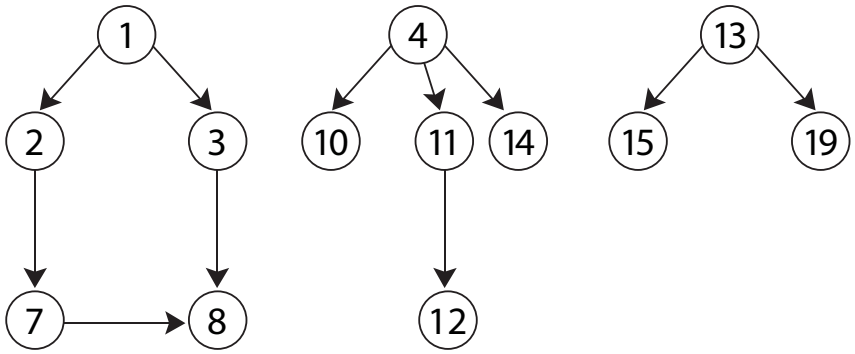


Figure 0.1 Dependency of mathematical background across some of the chapters. Each node is a chapter. Each directional link is a dependence relationship, *e.g.*, Chapter 8's material requires those in Chapters 3 (which in turn requires those in Chapter 1) and those in Chapter 7 (which in turn requires those in Chapter 2, which in turn requires those in Chapter 1). Chapters 1, 4, and 13, the root nodes of these three trees, offer foundational materials for ten other chapters. Some chapters aren't shown here because they don't form part of a dependence tree.

- Question well formulated, but there's a gap between theory and practice (and we discuss some possible bridges over the gaps): 12, 20.

- Question less well formulated (but certainly important to raise and explore): 5, 6, 7, 8.

It's comforting to see that the majority of our 20 chapters belong to the first group. Not surprisingly, questions about technological networks tend to belong to the first group, with those about social and economic networks more towards the second and third groups. It's often easier to model networked devices than networked human beings with predictive power.

Not all chapters explicitly study the impact of network topology, *e.g.*, Chapter 7 studies influence models with decision externalities based on population sizes, while Chapter 8 looks at influence models with topology taken into account.

A quick word about the homework problems. There are 5 problems at the end of each chapter. These are a mixture of easy drills, simple extensions, challenging mini-research projects, and open-ended questions. And they're ordered as such. Some important topics that we cannot readily fit into the main flow of the text are postponed to the homework problem section.

Notation

We use **boldface** text to denote key index terms when each of them is defined. We use *italics* to highlight important, subtle, or potentially confusing points.

We use boldface math symbols to denote vectors or matrices, *e.g.*, $\mathbf{x}$, $\mathbf{A}$. Vectors are column vectors by default. We do not use special fonts to represent sets. We use (t) to index iterations over continuous time, and $[t]$ or $[k]$ to index iterations over discrete time. We use * to denote optimal or equilibrium quantities.

Some symbols have different meanings in different chapters, because they are the standard notation in different communities.

Related Books

There's no shortage of books on networks of many kinds. The popular ones that appeared in the past decade or so fall into two main groups:

- Popular science books, many of them filled with historical stories, empirical evidence, and sometimes a non-mathematical sketch of technical content. Some of the widely read ones are: *Bursts, Connected, Linked, Money Lab, Planet Google, Six Degrees, Sync, The Perfect Storm, The Tipping Point, The Wisdom of Crowds*. Two other books, while not exactly on networks, provide important insights to many topics in networking: *Thinking, Fast and Slow* and *The Black Swan*. On the technology networks side, there are plenty of "for dummies" books, industry certification prep books, and entrepreneurship books. There're also several history of technology books, *e.g.*, *Where the Geeks Stay Up Late* and *The Qualcomm Equation*.
- Popular undergraduate or graduate level textbooks. On the graph-theoretic and economic side of networking, three excellent textbooks appeared in 2010: *Networks, Crowds, and Markets* by Easley and Kleinberg, *Networks* by Newman, and *Social and Economic Networks* by Jackson. The latter two are more on the graduate level. An earlier popular textbook is *Social Network Analysis: Methods and Applications* by Wasserman and Faust. On the computer networking side, there's a plethora of excellent textbooks written over the past decade. Two particularly popular ones are: *Computer Networking: A Top-Down Approach* by Kurose and Ross, and *Computer Networks: A Systems Approach* by Peterson and Davie. On wireless communications, several textbooks in the last few years have become popular: *Wireless Communications* by Molisch, *Wireless Communications* by Goldsmith, *Fundamentals of Wireless Communication* by Tse and Viswanath.

As illustrated in Figure 0.2, this book fills in the gap between existing groups of books. Each chapter is driven by a practical question or observation, but the answers (or approximate answers) are explained using the rigorous language of mathematics. It also maintains a balance bewteen social/economic networks and

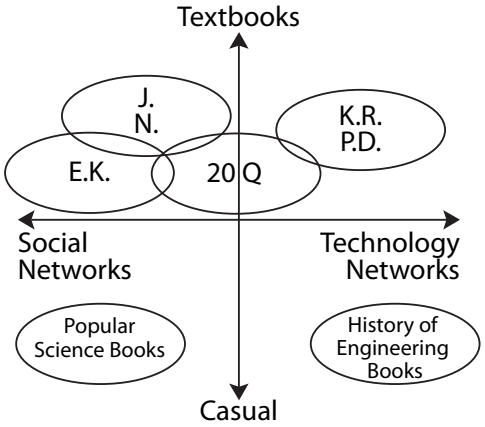


Figure 0.2 A cartoon illustrating roughly where some of the related books sit on two axes: one on the level of difficulty ranging from leisurely reading to graduate level textbooks, and another on the mix of topics ranging from social and economic networks to technological networks. E.K. stands for Easley and Kleinberg. J. stands for Jackson. N. stands for Newman. K. R. stands for Kurose and Ross. P. D. stands for Peterson and Davie. 20Q stands for this book.

Internet/wireless networks, and between graph/economic theory and optimization/learning theory. For example, understanding why WiFi works slower in hot spots is given as much attention as how Google auctions its ad spaces. A main goal of this book is to put social economic networks and technological networks side by side, and highlight their surprising similarities in spirit and subtle differences in detail.

In similar ways, the Princeton undergraduate course also differs from the seminal courses by Easley and Kleinberg at Cornell, and by Kearns at Penn, which have inspired a few similar courses, such as the one by Parke at Harvard, by Prabhakar at Stanford, by Wierman at Caltech, by Chaintreau at Columbia, by Spielman at Yale, by Kempe at USC... These excellent courses have started structuring social and economic networking topics to undergraduates, and inspired our course at Princeton. Of course, both computer networking and wireless communications courses are standard, and sometimes required courses at many universities CS and EE departments. We hope there'll be more courses in electrical engineering departments around the world that use rigorous languages to teach the concepts and methods common to social, economic, and technological networks.

Pedagogical Principles

This book and the associated course are also an experiment in three principles of teaching networks: JIT, BTP, and BAN.

Principle 1: Just In Time (JIT)

Models are often crippled by their own assumptions to start with, and end up being largely irrelevant to what they set out to enable. Once in a while this is not true, but that's a low probability event. So, before presenting any model, we first try to justify why the models are really necessary. The material is arranged so that extensive mathematical machinery is introduced bit by bit, each bit presented just in time for the question raised. We enforce this "just-in-time" policy pretty strictly: no mathematical machinery is introduced unless it's used within the same section.

This might seem to be a rather unconventional way to write a textbook on the mathematical side of engineering. Usually a textbook asks the students to be patient with 50, 100, sometimes 200 pages of mathematics to lay the foundation first, and promises that motivating applications are coming after these pages. It's like asking a 3-year-old to be patient for a long drive and promising ice-cream cones after many miles on the highway. In contrast, this book hands out an ice-cream cone every minute along the way. Hopefully the 3-year-old becomes very motivated to keep the journey going. It's more fun when gratification isn't delayed. "Fun right now" and "instant gratification" are what this book tries to achieve.

This book is an experiment motivated by this hypothesis: what professors call "fundamental knowledge" can be taught as "by-products" in the answers to practical questions that students are interested in. A devoted sequence of lectures focusing exclusively (or predominantly) on the fundamental knowledge is not the *only* way to teach the material. Maybe we could also chop up the material and sprinkle it around. This does not "water-down" the material, it simply reorganizes it so that it shows up right next to the applications in each and every lecture. The downside is that the standard trains of thought running through the mathematical foundation of research communities are interrupted many times. This often leaves me feeling weird because I could not finish my normal teaching sequence, but that's probably a good sign. The upside is that undergraduates, who may not even be interested in a long-term career in this field, view the course as completely driven by practical questions.

For example, the methodologies of optimization theory are introduced bit by bit: The basic definitions and Perron Frobenius theory in power control, convexity and least squares in Netflix recommendation, network utility maximization in Internet pricing, dynamic programming and multi-commodity flow in Internet routing, gradient and dual decomposition in congestion control, and combinatorial optimization in peer-to-peer networks.

The methodologies of game theory are introduced bit by bit in this book: The basic definitions in power control, auction theory in ad space auctioning, bargaining theory in Wikipedia consensus formation as well as in two-sided pricing of Internet access, and selfish maximization in tipping.

The methodologies of graph theory are introduced bit by bit: matching in ad space auctioning, consistency and pagerank in Google search, bipartite graph in Netflix recommendation, centrality, betweenness, and clustering measures in influence models, small world models in social search, scale free models in Internet topology, Bellman Ford algorithm and max flow min cut in Internet routing, and tree embedding in P2P.

The methodologies of learning theory are introduced bit by bit: collaborative filtering in Netflix recommendation, Bayesian analysis and adaptive boosting in ratings, and community detection in influence models.

Principle 2: BTP (Bridge Theory and Practice)

The size of the global industry touched upon by these 20 question is many trillons of dollars. Just the sum of market capitalizations of the 20 most relevant U.S. companies to this book: Google (including YouTube), Microsoft (including Skype), Amazon, eBay, Facebook, Twitter, Groupon, LinkedIn, Netflix, Disney, Apple, AT&T, Verizon, Comcast, Qualcomm, Ericsson, Cisco, EMC, HP, Intel, added up to over $1.8 trillion as of November 2011.

In theory, this book's theories are directly connected to practice in this multi-trillion-dollar industry. In practice, that's not always true, especially in fields like networking where stable models, like the additive Gaussian noise channel for copper wire in communication theory, often do not exist.

Nonetheless, we try to strike a balance between (a) presenting enough detail so that answers to these practical questions are grounded in actual practice rather than in "spherical cows" and "infinite planes," (although we couldn't help but keep "rational human beings" in several chapters), and (b) avoiding too much detail that reduces the "signal-noise-ratio" in illustrating the fundamental principles. This balance is demonstrated in the level of detail with which we treat network protocol descriptions, Wikipedia rules, Netflix recommendation algorithm parameters, etc. And this tradeoff explains the (near) absence of random graph theory and of Internet protocol header formats, two very popular sets of material in standard textbooks in math/CS-theory/sociology and in CS-systems/EE, respectively.

Some of these 20 questions are currently trapped in particularly deep theory-practice gaps, especially those hard-to-formulate questions in Chapters 5 and 6, and those hard-to-falsify answers in Chapters 7 and 8. The network economics material in Chapters 11 and 12 also fits many of the jokes about economists, too many to quote here. (A good source of them is Taleb's *The Bed of Procrustes*.) Reverse engineering, shown across many chapters, has its own share of accurate jokes: "Normal people look at something that works in theory, and wonder if it'll

also work in practice. Theoreticians look at something that works in practice, and wonder if it'll also work in (their) theory."

Time and time again, we skip the mechanics of mathematical acrobats, and instead highlight the never-ending struggles between representations and realities during modeling: the process of "mathematical crystallization" where (most) parts of reality are thrown out of the window so that what remains becomes tractable using today's analytic machineries. What is often unclear is whether the resulting answerable-questions are still relevant and the resulting tractable models still have predictive powers. However, when modeling is done "right", engineering artifacts can be explained rather than just described, and better design can be carried out top-down rather than by "tweak and debug." It's often been quoted (mostly by theoreticians like me) that "there's nothing more practical than a good theory," and that "a good theory is the first order exponent in the Taylor's expansion of reality." Perhaps these can be interpreted as *definitions* of what constitutes a "good" theory. By such a definition, this book has traces of good theory, thanks to many researchers and practitioners who have been working hard on bridging the theory-practice gap in networking.

Principle 3: BAN (Book As a Network)

Throughout the chapters, comparison and contrast are constantly drawn with other chapters. This book itself is a network, a network of ideas living in nodes called chapters, and we grasp every opportunity to highlight each possible link between these nodes. The most interesting part of this book is perhaps this networking effect among ideas: to see how curiously related, and yet crucially different they are.

Figure 0.3 shows the main connections among the chapters. This is what the book is about: weave a network of ideas (about networks), and the positive networking effect comes out of that.

We can extract the top 20 ideas across the chapters. The first 10 are features of networks, the next 5 design ideas, and the last 5 modeling approaches.

1. Resource sharing (such as statistical multiplexing and fairness): Ch. 1, 11, 13, 14, 15, 16, 17, 18, 20
2. Opinion aggregation and consensus formation: Ch. 3, 4, 5, 6, 18
3. Positive network effect (such as resource pooling and economy of scale): Ch. 9, 11, 13, 15, 16
4. Negative network effect (such as tragedy of the commons): Ch. 11, 20
5. The wisdom of crowds (diversity gain and efficiency gain): Ch. 7, 8, 18, 19
6. The fallacy of crowds (cascade and contagion): Ch. 7, 8
7. Functional hierarchy and layering: Ch. 13, 14, 15, 17, 19
8. Spatial hierarchy and overlaying: Ch. 10, 13, 15, 16, 17
9. From local actions to global property: Ch. 1, 6, 7, 8, 13, 14, 15, 18
10. Overprovision capacity vs. connectivity: Ch. 14, 15, 16

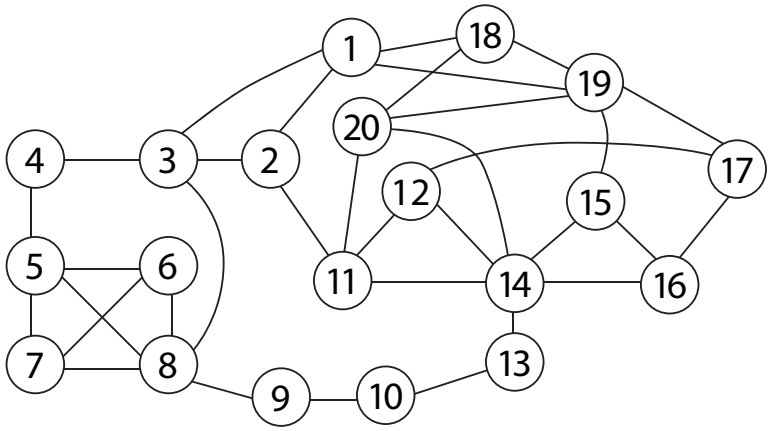


Figure 0.3 Intellectual connections across the chapters. A node is a chapter, and a bidirectional link is an intellectual connection, via either similar concepts or common methodologies. Cliques of nodes and multipath paths from one node to another are particularly interesting to observe in this graph.

11. Feedback control: Ch. 1, 7, 13, 14
12. Utility maximization: Ch. 1, 2, 11, 12, 14, 20
13. Protocol: Ch. 14, 15, 17, 19
14. Signaling: Ch. 6, 19
15. Randomization: Ch. 3, 15, 18
16. Graph consistency models: Ch. 3, 13
17. Strategic equilibrium models: Ch. 1, 2, 15
18. Generative model (and reverse engineering): Ch. 9, 10, 14
19. Latent factor models: Ch. 4
20. Axiomatization models: Ch. 6, 20

In the first offering of the course at Princeton, the undergrads voted (by Borda count) resource sharing, opinion aggregation, and positive network effect as the top three concepts they found most useful. And they also voted the key equations in pagerank, distributed power control, and Bellman Ford as the top three equations.

Almost every one of these 20 ideas cuts across social/economic networks and technological networks. For example,

- The emergence of global coordination through local actions based on local views is a recurring theme from influence models in social networks to routing and congestion control in the Internet, from consumer reaction to pricing signals to power control in wireless networks.
- Resource sharing models, in the form of additive sharing $x + y \leq 1$, or mulitplicative sharing $x/y \geq 1$, or binary sharing $x, y \in \{0, 1\}, x + y \leq 1$, are

introduced for network pricing as well as the classic problems of congestion control, power control, and contention control. Indeed, congestion control in TCP has been interpreted as a form of dynamic pricing in network access.

- The (positive) network effect is often highlighted in social and economic networks. It also finds a concrete realization in how content is shared over the Internet through peer-to-peer protocols and scaling-up of data center.
- "The wisdom of (independent and unbiased) crowds" is another common theme in social networks. There're two types of "wisdom" here: diversity gain in reducing the chance of some bad event (typically represented mathematically $1-(1-p)^N$ where N is the size of the crowd), and efficiency gain in smoothing out some average metric (typically represented mathematically as a factor $\sqrt{N}$ in the metric). Both types are observed in social networks and the latest generation of 4G and 802.11n wireless networks.
- Forming consensus is used in computing webpage importance score in pagerank as well as in discovering the right time to transmit in WiFi.
- Spatial hierarchy is used in both small world models and how the Internet is structured.
- The design methodology of feedback control is used in influence models in social networks and congestion control in the Internet.
- Utility maximization is used in auctioning advertising spots and setting Internet access pricing.
- The power method is used in both Google pagerank and distributed power control.
- Randomization is used in Google pagerank and 802.11 CSMA.
- Strategic equilibrium models are used in auctioning and BitTorrent.
- Reverse engineering is used in studying scale free networks and TCP.
- Axiomization is used in voting procedure and fairness evaluation.

Yet equally important are the subtle differences between technological and social-economic networks. Exhibit A for this alert is the (non-existence of) the Achilles' heel of the Internet and the debate between two generative models (preferential attachment vs. constrained optimization) of scale free networks.

Two Bigger Pictures

There are also two broader pictures in the backdrop of this book:

- *Instill domain specific functionalities to a generic network science.* Network science around these 20 questions must be based on domain-specific models and on the pursuit of falsification. For example, while a random graph is elegant, it's often neither a relevant approach to design nor the only generative model to explain what we see in this book. And as much as metrics of a static graph are important, engineering protocols governing

the *functionalities* of feedback, coordination, and robustness are just as crucial as the *topological* properties of the graph like degree distribution.

- *Revisit the Electrical and Computer Engineering (ECE) undergraduate curriculum.* In the standard curriculum in electrical engineering since around the 1960s, a "signals and systems" course is one of the first foundational courses. As networks of various kinds play an increasingly important role both in engineering design and in society, it's time to capture fundamental concepts in networking in a second systems course. Just like linear time-invariant systems, sampling, integral transforms, and filter design have laid the foundation of ECE curriculum since the 1960s, we think the following concepts have now become fundamental to teach to the future ECE students, whether they are taught in the JIT way or not: patterns of connections among nodes, modularization and hierarchy in networked systems, consistency and consensus in graphs, distributed coordination by pricing feedback, strategic equilibrium, pros and cons of scaling up, etc.

  So this book is an experiment in both *what* to teach and *how* to teach in an electrical and computer engineering undergrad curriculum: what constitutes core knowledge that needs to be taught and how to teach it in a context that enhances learning efficiency. As much as we appreciate FIR and IIR filter design, Laplace and Z transforms, etc., maybe it's about time to explore the possibility of reducing the coverage of these topics by just a tiny bit to make room for mathematical notions just as fundamental to engineering today. And we believe the best way to drive home these ideas is to tie in with applications that teenagers, and many of the older folks, use every day.

Class as a Social and Economic Network

The class "Networks: Friends, Money, and Bytes," created in parallel to this book in fall 2011 and cross listed in electrical engineering and computer science at Princeton University, was a social and economic network itself. We tweeted, we blogged, and we created wikis. On the first day of the class, we drew a class social graph, where each node is a student, and a link represents a "know by first name before coming to the first class" relationship. After the last lecture, we drew the graph again.

We also created our own currency called "nuggets." The TAs and I "printed" our money as we saw fit. There were several standard ways to earn nuggets, including catching typos in lecture notes and writing popular blogs. There were 10 class activities beyond homework problems that were rewarded by nuggets, including one activity in which the students were allowed to buy and sell their homework solutions using auctions. The matching of students and class project topics was also run through bidding with nuggets. Eventually the nugget bal-

ances translate into an upward adjustment of grades. To see some of the fun of ELE/COS 381 at Princeton University, visit `www.network20q.com`.

1 What makes CDMA work for my smartphone?

1.1 A Short Answer

Take a look at your iPhone, Android phone, or a smartphone running on some other operating system. It embodies a remarkable story of technology innovations. The rise of wireless networks, the Internet, and the web over the last five decades, coupled with advances in chip design, touchscreen materials, battery packaging, software systems, business model... led to this amazing device you are holding in your hand. It symbolizes the age of networked life.

These phones have become the mobile, lightweight, smart centers of focus in our lives. They are not just used for voice calls, but also for **data applications** : texting, emailing, browsing the web, streaming videos, downloading books, uploading photos, playing games, or video-conferencing friends. These data fly through a **cellular network** and the **Internet**. The cellular network in turn consists of the radio air-interface and the core network. We focus on the air-interface part in this chapter, and turn to the cellular core network in Chapter 19.

Terrestrial wireless communication started back in the 1940s. And cellular networks have gone through generations of evolution since the 1970s, moving into what we hear as 4G these days. Back in the 1980s, some estimated that there would be 1 million cellular users in the US by 2000. That turned out to be one of those under-estimations that did not even get close to the actual impact of networking technologies.

Over more than three decades of evolution, the fundamental concept of cellular architecture has remained essentially the same. The entire space of deployment is divided into smaller regions called **cells** , often represented by hexagons as in Figure 1.1, thus the name cellular networks and cell phones. There is one **base station** (BS) in each cell, connected on the one side to switches in the core network, and on the other side the **mobile stations** (MS) assigned to this cell. An MS could be a smart phone, a tablet, a laptop with a dongle, or any device with antennas that can transmit and receive in the right frequencies following a cellular network standard.

We see a clear hierarchy, a fixed infrastructure, and one-hop radio links in cellular networks. This is in contrast to other types of wireless networks. Moreover,

the deployment of base stations is based on careful radio engineering and tightly controlled by a wireless provider, in constrast to WiFi networks in Chapter 18.

Why do we divide the space into smaller regions? Because the wireless spectrum is scarce and radio signals weaken over space.

Transmitting signals over the air means emitting energy over parts of the **electromagnetic spectrum**. Certain regions of the spectrum are allocated by different countries to cellular communications, just like other parts of the spectrum are allocated to AM and FM radio. For example, the 900 MHz range is allocated for the most popular 2G standard called GSM, and in Europe the 1.95 GHz range and 2.15 GHz range are allocated for UMTS, a version of the 3G standard. Some part of the spectrum is unlicensed, like in WiFi as we will see in Chapter 18. Other parts are licensed, like those for cellular networks, and a wireless service provider needs to purchase these rare resources with hefty prices. The spectrum for cellular networks is further divided into chunks, since it is much easier for transmitters and receivers to work with frequency bands with widths on the order of 10MHz.

The signals sent in the air become weaker as they travel over longer distances. The amount of this attenuation is often proportional to the square, or even the fourth power, of the distance traversed. So, in a typical cellular network , the signals become too weak to be accurately detected after a couple of miles. At first glance, this may sound like bad news. But it also means that the same frequency band used by base station A can be reused by another base station B sufficiently far away from A. All we need to do is to tesselate the frequency bands, as illustrated in Figure 1, so that no two cells share the same frequency band if they are too close. In Figure 1.1, we say that there is a **frequency reuse factor** of 7, since we need that many frequency bands to avoid two close cells sharing the same frequency. **Cellular architecture** enables the network to scale up over space. We will visit several other ways to scale up a network later.

Now, how can the users in the same cell share the same frequency band? There are two main approaches: orthogonal and non-orthogonal allocation of resources.

Frequency is clearly one type of resource, and time is another. In **orthogonal allocation**, each user is given a small band of frequency in Frequency Division Multiple Access (**FDMA**), or a timeslot in Time Division Multiple Access (**TDMA**). Each user's allocation is distinct from the others, as shown in Figure 1.1(a)(b). This often leads to an inefficient use of resources. We will see in later chapters a recurring theme: a dedicated assignment of resources to users becomes inefficient when users come and go frequently.

The alternative, **non-orthogonal allocation**, allows all users to transmit at the same time over the same frequency band, as in Code Division Multiple Access. **CDMA** went through many ups and downs with technology adoption from 1989-1995, but is now found in all the 3G cellular standards as part of the design. In CDMA's first standard, IS-95 in the 2G family, the same frequency band is reused in all the cells, as illustrated in Figure 1.2(c). But how can we tell the users apart if their signals overlap with each other?

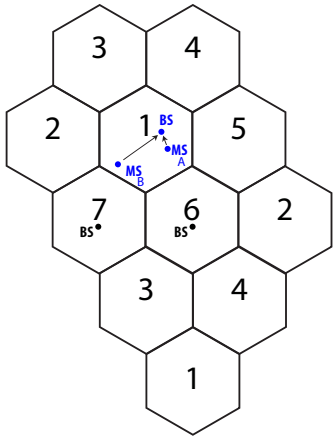


Figure 1.1 Part of a typical cellular network with a frequency reuse of 7. Each cell is a hexagon with a base station (BS) and multiple mobile stations (MSs). Only a few of them are drawn in the figure. Some mobile stations, like MS A, are close to the base station with strong channels to the BS. Others, like MS B, are on the cell edge with weak channels.

Think of a cocktail party with many pairs of people trying to carry out individual conversations. If each pair takes turns in communicating, and only one person gets to talk at each time slot, we have a TDMA system. If all pairs can communicate at the same time, and each uses a different language to avoid confusion, we have a CDMA system. But there are not enough languages whose pronounciations do not cause confusion, and human ears are not that good at decoding, so interference is still an issue. How about controlling each person's volume? Each transmitter adjusts the volume of its voice based on the relative distances among the speakers and listeners. In a real cocktail party, unless there is some politeness protocol or it hurts someone's vocal chord to raise voice, we end up in a situation where everyone is shouting at the top of their voices. Transmit power control can hopefully mitigate this problem.

The core idea behind the CDMA standards is as follows: the transmitter multiplies the digital signals with a sequence of 1s and -1s, a sequence we call the **spreading code**. The receiver multiplies the received bits with the same spreading code to recover the original signals. This is straight-forward to see: 1×1 is 1, an -1×-1 is also 1. What is non-trivial is that a family of spreading codes can be designed such that only *one* spreading code, the original one used by the transmitter, can recover the signals. If you use any other spreading codes in this family, you will get noise-like, meaningless bits. We call this a family of **orthogonal codes**. Users are still separated by orthogonalization, just along the "code dimension" as opposed to the more intuitive "time dimension" and "frequency dimension." This procedure is called **direct sequence spread spectrum**, one of the two ways to enable CDMA.

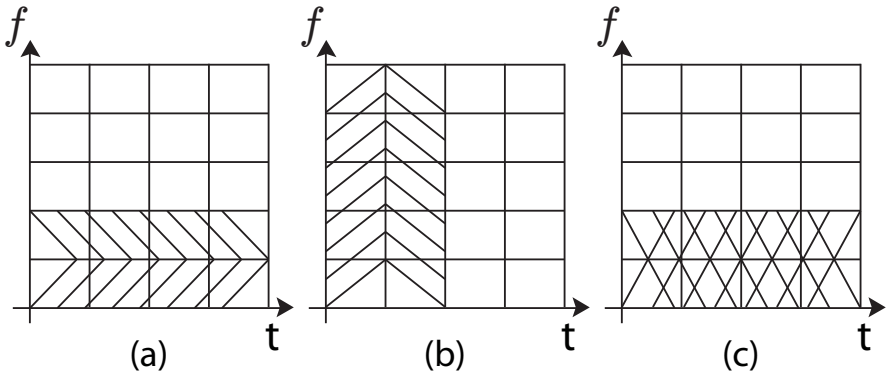


Figure 1.2 Part of a time-frequency grid is shown in each graph above. (a) FDMA and (b) TDMA are dedicated resource allocation: each frequency band or time slot is given to a user. In contrast, (c) CDMA is shared resource allocation: each time-frequency bin is shared by multiple users, all transmitting and receiving over the same frequency band and at the same time. These users are differentiated by signal processing. Power control also helps differentiate their signals. For visual simplicity, we only show two slices of resources being used.

However, there may not be enough orthogonal spreading codes for all the mobile stations. Families of orthogonal codes are limited in their sizes. Furthermore, a slight shift on the time axis can scramble the recovered bits at the receiver. We need the clocks on all the devices to be synchronized. But this is infeasible for the **uplink**, where mobiles talk to the base station: MSs cannot easily coordinate their clocks. It is difficult even in the **downlink**, where the base station talks to the mobiles: the BS has one single clock but the wireless channel distorts the bits. In the end, we do not have perfectly orthogonal spreading codes, even though these imperfect codes still provide significant "coding gain" in differentiating the signals.

We need an alternative mechanism to differentiate the users, and to tackle the **interference** problem; wireless signals are just energy propagating in the air, and one users's signal is every other user's interference. Interference, together with attenuation of signal over distance and fading of signal along multiple paths, are the top three issues we have to address in wireless channels.

As shown in Figure 1.1, a user standing right next to the BS can easily overwhelm another user far away at the edge of the cell. This is the classic **near-far problem** in CDMA networks. It was solved in the IS-95 standard by Qualcomm about 20 years ago. This solution was one of the cornerstones in realizing the potential of CDMA since then.

Qualcomm's solution in 1989 to the near-far problem is simple and effective. It leverages the inference and then the **feedback** of the channel quality estimate.

Consider an uplink transmission: multiple MSs trying to send signals to the BS in a particular cell. The BS can estimate the channel quality from each MS to itself, *e.g.*, by looking at the ratio of the received signal power to the transmitted power, the latter being pre-configured to some fixed value during the channel estimation timeslot. Then, the BS inverts the channel quality and sends that value, on some feedback control channel, back to the MSs, telling them that these are the gain parameters they should use in setting their transmit powers. This way, all the received signal strengths will be equal. This is the basic **transmit power control** algorithm in CDMA.

But what if equalization of the received signal powers is *not* the right goal? For voice calls, the typical application on cell phones in 2G networks, there is often a target value of the received signal quality that each call needs to achieve. This signal quality factor is called the Signal to Interference Ratio (**SIR**). It is the ratio between the received signal strength and the sum of all the interference's strength (plus the receiver noise strength). Of course it is easy to raise SIR for one user: just increase its transmitter's power. But that translates into higher interference for everyone else, which further leads to higher transmit powers by them if they also want to maintain or improve their SIRs. This positive feedback escalates into a transmit power "arms race"; until each user is shouting at the top of her voice. That would not be a desirable state to operate in.

If each user fixes a reasonable target SIR, can we do better than this "arms race" through a more intelligent power control? Here, "being reasonable" means that the set of SIRs targeted by all the users in a cell are mutually compatible; they can be *simultaneously* achieved by some configuration of transmit powers at the MSs.

The answer is yes. In 1992-1993, a sequence of research results developed the basic version of **Distributed Power Control** (DPC), a fully **distributed algorithm**. We will come back to discuss what we mean by "distributed" and "fully distributed." For now, it suffices to say that, in DPC, each pair of transmitter (*e.g.*, an MS) and receiver (*e.g.*, the BS) does not need to know the transmit power or channel quality of any other pairs. At each time slot, all it needs to know is the actual SIR it currently achieves at the receiver. Then take the ratio between the *fixed*, target SIR and the actual SIR value measured at this time slot, multiply the current transmit power with that ratio, and you get the transmit power for the next timeslot. This update happens simultaneously at each pair of transmitter and receiver.

This simple method is an **iterative algorithm**; the updates continue from one timeslot to the next, unlike the one-shot, received-power-equalization algorithm. But it is still simple, and when the target SIRs can be simultaneously achieved, it has been proven to **converge**: the iterative procedure will stop over time. When it stops, it stops at the right solution: a power-minimal configuration of transmit powers that achieve the target SIRs for all. DPC converges quite fast, approaching the right power levels with an error that decays as a geometric

series. DPC can even be carried out asynchronously: each radio has a different clock and therefore different definition of what timeslot it is now.

Of course, in real systems the time slot is indeed asynchronous and power levels are *discrete*. Asynchronous and quantized versions of DPC have been implemented in all the CDMA standards in 3G networks. Some standards run power control 1500 times every second, while others run 800 times a second. Some discretize power levels to 0.1dB, while others 0.2-0.5dB. Without CDMA, our cellular networks today could not work as efficiently. Without power control algorithms (and the associated handoff method to support user mobility), CDMA could not function properly. In a 4G standard called LTE, a technology called OFDM is used instead of CDMA, but power control is still employed for interference reduction and for energy management.

Later in Chapter 18, we will discuss some of the latest ideas that help further push the data rates in new wireless network standards, ranging from splitting, shrinking, and adjusting the cells to overlaying small cells on top of large ones for offloading, from leveraging multiple antennas and tilting their positions to chopping up the frequency bands for more efficient signal processing.

1.2 A Long Answer

1.2.1 Distributed Power Control (DPC)

Before we proceed to a general discussion of the DPC algorithm, we must first define some symbols.

Consider N pairs of transmitters and receivers. Each pair is a logical channel, indexed by i. The transmit power of the transmitter of link i is p_i, some positive number, usually capped at a maximum value: $p_i \leq p_{max}$ (although we will not consider the effect of this cap in the analysis of the algorithm). The transmitted power impacts both the received power at the intended receiver and the received interference at the receivers of all the other pairs.

Now, consider the channel from the transmitter of link (*i.e.*, transmitter-receiver pair) j to the receiver of link i, and denote the **channel gain** by G_{ij}. So G_{ii} is the direct channel gain, the bigger the better, since it is the channel for the intended transmission for tranmitter-receiver of link i. All the other G_{ij}, for j not equal to i, are gains for interference channels, so the smaller the better. We call these channel "gains" whereas actually they are less than 1, so maybe a better term is channel "loss."

This notation is visualized in Figure 1.3 for a simple case of two MSs talking to a BS, which can be thought of as two different "logical" receivers physically located together.

These $\{G_{ij}\}$ are determined by two main factors: location of the transmitters and receivers, and the quality of the channel in between. G_{ii} is also enhanced

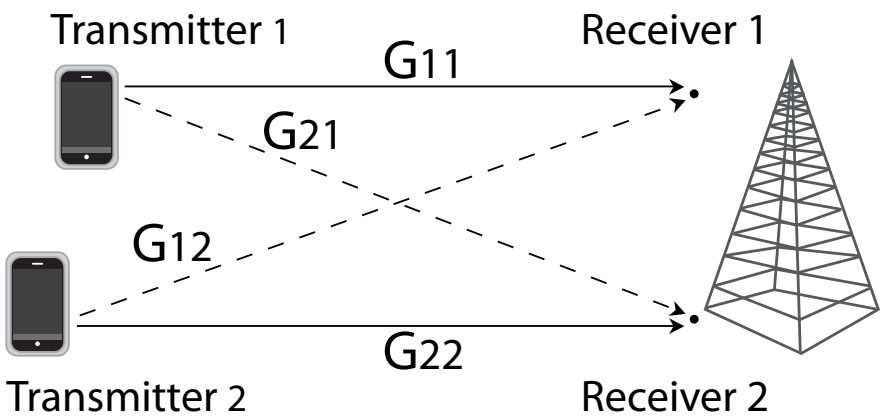


Figure 1.3 Uplink interference between two mobile stations at the base station. We can think of the base station as two receivers collocated. G_{11} and G_{22} are direct channel gains, the bigger the better. G_{12} and G_{21} are interference channel gains, the smaller the better. Transmit powers are denoted by p_i and received signal-interference-ratio by SIR_i.

by the CDMA spreading codes that help the intended receivers decode more accurately.

The received power of the intended transmission at the receiver is therefore $G_{ii}p_i$. What about the interference? It is the sum of $G_{ij}p_j$ over all transmitters j (other than the correct one i): $\sum_{j\neq i} G_{ij}p_j$. There is also noise n_i in the receiver electronics for each receiver i. So we can write the SIR, a unit-less ratio, at the receiver of logical link i as

$$\text{SIR}_i = \frac{G_{ii}p_i}{\sum_{j\neq i} G_{ij}p_j + n_i}. \tag{1.1}$$

For proper decoding of the packets, the receiver needs to maintain a target level of SIR. We will denote that as γ_i for link i, and we want $\text{SIR}_i \geq \gamma_i$ for all i. Clearly, increasing p_1 raises the SIR for receiver 1 but lowers the SIR for all other receivers.

As in a typical algorithm we will encounter throughout book, we assume that time is divided into discrete slots, each indexed by $[t]$. At each time t, the receiver on link i can measure the received SIR readily, and feed back that number $\text{SIR}_i[t]$ to the transmitter.

The DPC algorithm can be described through a simple equation: each transmitter simply multiplies the current power level $p_i[t]$ by the ratio between the target SIR, γ_i, and the current measured SIR_i, to obtain the power level to use in the next timeslot:

$$p_i[t+1] = \frac{\gamma_i}{\text{SIR}_i[t]} p_i[t], \quad \text{for each } i. \tag{1.2}$$

We see that each user i only needs to measure its own SIR at each iteration, and remember its own target SIR. There is no need for passing any control message around, like telling other users what power level you are using. Simple in *communication*, it is a *very* distributed algorithm, and we will later encounter many types of distributed algorithms in various kinds of networks.

This algorithm is also simple in its *computation*: just one division and one multiplication. And it is simple in its parameter *configuration*: there is actually no parameters in the algorithm that need to be tuned, unlike quite a few other algorithms that we will encounter in the book. Simplicity is a key reason why certain algorithms are widely adopted in practice.

Intuitively, this algorithm makes sense. First, when the iterations stop because no one's power is changing any more, *i.e.*, we have converge to an **equilibrium**, we can see that $\text{SIR}_i = \gamma_i$ for all i.

Second, there is hope that the algorithm will actually converge, based on the direction in which the power levels are moving. The transmit power moves up when the received SIR is below the target, and moves down when it is above the target. Proving that convergence will happen is not as easy. As one transmitter changes its power, the other transmitters are doing the same, and it is unclear what the next timeslot's SIR values will be. In fact, this algorithm does *not* converge if too many γ_i are too large, *i.e.*, when too many users request large SIRs as their targets.

Third, if satisfying the target SIRs is the only criterion, there are many transmit power configurations that can do that. If $p_1 = p_2 = 1$ mW achieves these two users' target SIR, $p_1 = p_2 = 10$ mW will do so too. We would like to pick the configuration that uses the least amount of power; we want a power-minimal solution. And the algorithm above seems to be adjusting power lower when high power is unnecessary.

1.2.2 DPC as an optimization solution

In general, the questions of "will it converge" and "will it converge to the right solution" are the top two questions that we would like to address in the design of all iterative algorithms. Of course, what "the right solution" means will depend on the definition of optimality. In this case, power-minimal transmit powers that achieve the target SIR for all users are the "right solution." Power minimization is the **objective** and achieving target SIR for all users is the **constraint**.

In this case, there are many ways to address these questions, for example, using machineries from optimization theory or game theory. Either way, we can show that, under the condition that the target SIR values are indeed **achievable** at all, *i.e.*, there are some values of the transmit powers that can achieve the target SIRs for all users, DPC will converge, and converge to the right solution.

We can illustrate a typical set of feasible SIRs in the SIR feasibility region shown in Figure **??**. One user's higher SIR can be achieved at the expense of a lower SIR for another user. This highlights another recurrent theme in the book:

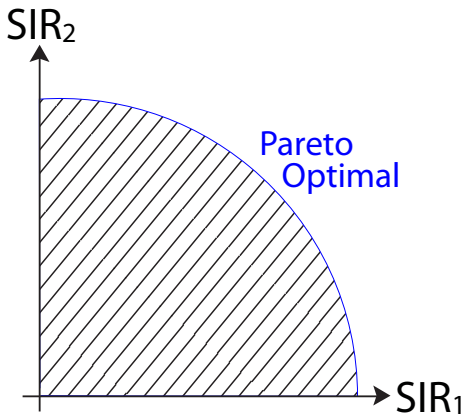


Figure 1.4 An illustration of the SIR feasibility region. It is a constraint set for power control optimization, and visualizes the competition among users. Every point strictly inside the shaded region is a feasible vector of target SIRs. Every point outside is infeasible. And every point on the boundary of the curve is Pareto optimal: you cannot increase one user's SIR without reducing another user's SIR.

the need to tackle tradeoffs among users and among design objectives, and the importance of providing incentives for people to react to in achieving a desirable tradeoff.

It turns out that DPC also solves a global optimization problem for the network. Here, "global" means that the interests of *all* the users are incorporated. In this case, it is the sum of transmit powers that is minimized, and every user's target SIR must be met.

Once we have an objective function and a set of constraints, and we have defined which quantities are variables and which are constants, an **optimization** problem is formulated. In this case, the transmit powers are **variables**. The achieved SIRs are also variables, but are derived from the powers. All the other quantities are **constants**: they are not degrees of freedom under your control.

If the variable vector $\mathbf{x}_0$ satisfies all the constraints, it is called a **feasible solution**. If an optimization problem's constraints are not mutually compatible, it is called **infeasible**. If an $\mathbf{x}^*$ is both feasible and better than any other feasible solution, *i.e.*, gives the smallest objective function value for a minimization problem (or the largest objective function value for a maximization problem), it is called an **optimal** solution. An optimal solution may not exist, *e.g.*, minimize $1/x$ for $x \in \mathcal{R}$. And optimal solutions may not be unique.

Here is the optimization problem of varying transmit power to satisfy fixed

target SIR constraints and then minimize the total power:

$$\begin{array}{ll} \text{minimize} & \sum_i p_i \\ \text{subject to} & \mathsf{SIR}_i(\mathbf{p}) \geq \gamma_i, \ \ \forall i \\ \text{variables} & \mathbf{p}. \end{array} \tag{1.3}$$

Problem (1.3) would look complicated if you substitute the definition (1.1) of SIR as a function of the whole vector $\mathbf{p}$:

$$\begin{array}{ll} \text{minimize} & \sum_i p_i \\ \text{subject to} & \frac{G_{ii}p_i}{\sum_{j\neq i} G_{ij}p_j + n_i} \geq \gamma_i, \ \ \forall i \\ \text{variables} & \mathbf{p}. \end{array}$$

But it can be simplified through a different representation. We can rewrite it as a **linear programming problem**: minimizing a linear function of the variables subject to linear constraints of the variables:

$$\begin{array}{ll} \text{minimize} & \sum_i p_i \\ \text{subject to} & G_{ii}p_i - \gamma_i(\sum_{j\neq i} G_{ij}p_j + n_i) \geq 1, \ \ \forall i \\ \text{variables} & \mathbf{p}. \end{array}$$

Linear programming problems are easy optimization problems, and more generally, convex optimization (to be introduced in Chapter 4) is easy; easy in theory with its complexity count and easy in practice with fast solution software. Also shown in Advanced Material is the derivation of DPC as the solution to this optimization problem (1.3).

We will be formulating and solving optimization problems many times in future chapters. Generally, solving a global optimization via local actions by each user is difficult. But in this case, it turns out that the selfish behavior of users in their own power minimization also solves the global, constrained optimization problem; their interests are correctly aligned already. This is more of an exception than the norm.

1.2.3 DPC as a game

Power control is a competition. One user's received power is another's interference. Each player searches for the right "move," (or in this case, the right transmit power) so that its "payoff" is optimized (in this case, the transmit power is the smallest possible while providing the user with its target SIR γ_i). We also hope that the whole network reaches some desirable equilibrium as each player strategizes. The concepts of "players," "move," and "payoff" can be defined in a precise and useful way.

We can model *competition* as a **game**. The word "game" here carries a technical meaning. The study of games is a branch of mathematics called game theory. If the competition is among human beings, a game might actually correspond to people's strategies. If it is among devices, as in this case among radios, a game is more like an angle of interpretation and a tool for analysis. It turns out that

cooperation can also be modeled in the language of game theory, as we will show in Chapter 6.

In the formal definition, a game is specified by three elements:

1. A set of **players** $\{1, 2, \ldots, N\}$
2. A **strategy space** A_i for each player
3. A **payoff function**, or utility function, U_i for each player to maximize (or a **cost function** to minimize). Function U_i maps each combination of all players' strategies to a real number, the payoff (or cost), to player i.

Table 1.1 Prisoner's Dilemma. This is a famous game in which there is a unique and undesirable Nash equilibrium. Player A's two strategies are the two rows. Player B's two strategies are the two columns. The values in the table represent the payoffs to the two players in each scenario.

	`Not Confess`	`Confess`
`Not Confess`	(-1,-1)	(-5,0)
`Confess`	(0,-5)	(-3,-3)

Now consider a two-player game in Table 1.1. Player A's strategies are shown in rows and player B's in columns. Each entry in the 2×2 table has two numbers, (x, y), where x is the payoff to A and y to B if the two players pick the corresponding strategies. As you would expect from the coupling between the players, each payoff value is determined jointly by the strategies of both players. For example, the payoff function maps `(Not Confess, Not Confess)` to -1 for both players A and B. These payoffs are negative because it is about the number of years the two prisoners are going to serve in prison. If one confesses but the other does not, the one who confesses gets a deal to walk away free and the other one is heavily penalized. If both confess, both serve three years. If both do not confess, only a lesser conviction can be pursued and both serve one year. Both players know this table but they cannot communicate with each other.

This is the famous **prisoner's dilemma** game, which we will also encounter later in voting paradoxes, tragedy of the commons, and P2P file sharing.

If player A chooses strategy `Not Confess`, player B should choose strategy `Confess`, since $0 > -1$. This is called the **best response strategy** by player B, in response to player A choosing strategy `Not Confess`.

If player A chooses strategy `Confess`, player B's best response strategy is still `Confess`, since $-3 > -5$. When the best response strategy of a player is the *same* no matter what strategy the other player chooses, we call that a **dominant strategy**. It may not exist. But when it exists, a player will obviously pick a dominant strategy.

In this case, `Confess` is the dominant strategy for player B. By symmetry, it is also the dominant strategy for player A. So both players will pick `Confess`, and `(Confess, Confess)` is an **equilibrium** for the game. This is a slightly

different definition of equilibrium from what we saw before, where equilibrium means an update equation reaches a fixed point.

Clearly, this equilibrium is undesirable: `(Not Confess, Not Confess)` gives a higher payoff value to both players: -1 instead of -3. But the two prisoners could not have coordinated to achieve `(Not Confess, Not Confess)`. An equilibrium might not be **socially optimal**, *i.e.*, a set of strategies maximizing the sum of payoffs $\sum_i U_i$ of all the players. It might not even be **Pareto optimal**, *i.e.*, a set of strategies such that no player's payoff can be increased without hurting another player's payoff.

Table 1.2 Coordination Game. In this game, there are two Nash equilibria. Neither player has an incentive to unilaterally change strategy in either equilibrium.

	`Action Movie`	`Romance Movie`
`Action Movie`	(2,1)	(0,0)
`Romance Movie`	(0,0)	(1,2)

Consider a different game in Table 1.2. This is a typical game model for coordination, a task we will see many times in future chapters. You and your friend are trying to coordinate which movie to watch together. If you disagree, you will not go to any movie and the payoff is zero for both of you. If you agree, each will get some positive payoff but the values are different, as you prefer the romance movie and your friend prefers the action movie. (By the way, we will try to understand how to predict a person's preference for different types of movies in Chapter 4.)

In this game, there is no dominant strategy, but it so happens that there are pairs of best response strategies that match each other. If my best response to my friend picking strategy a is strategy b, and my friend's best response to my picking strategy b is strategy a, then (a, b) pair "matches."

In the coordination game above, `(Action, Action)` is such a pair, and `(Romance, Romance)` another pair. For both pairs, neither player has an incentive to *unilaterally* move away from its choice in this pair of strategies. If both move at the same time, they could both benefit, but neither wants to do that *alone*. This creates an equilibrium in strategic thinking: I will not move unless you move, and you think the same way too. This is called a **Nash equilibrium**. In prisoner's dilemma, `(Confess, Confess)` is a Nash equilibrium.

Symbolically, for a two-user game, suppose the two payoff functions are (U_1, U_2) and the two strategy spaces are $(\mathcal{A}, \mathcal{B})$ for the two players, respectively. We say $(a^* \in \mathcal{A}, b^* \in \mathcal{B})$ is a Nash equilibrium if:

$$U_1(a^*, b^*) \geq U_1(a, b^*), \quad \text{for any } a \in \mathcal{A},$$

and

$$U_2(a^*, b^*) \geq U_2(a^*, b), \quad \text{for any } b \in \mathcal{B}.$$

A Nash equilibrium may not exist in a game. And when it exists, it may not be unique (like in the coordination game above), or socially optimal, or Pareto optimal. But if the players are allowed to throw a coin and decide probabilistically which strategy to play, *i.e.*, a **mixed strategy**, it is guaranteed, by Nash's famous result, that a Nash equilibrium always exists.

We will expand and use our game theory language in several future chapters. In our current case of power control as a game, the set of players is the set of transmitters. The strategy for each player is its transmit power level, and the strategy space is the set of transmit powers so that the target SIR is achieved. The cost function to minimize is the power level itself.

In this power control game, while the cost function is independent across the players, each player's strategy space A_i actually depends on the transmit powers of all the other players. This coupling across the players is introduced by the very nature of interference, and leads to strategic moves by the players.

Here is a simple fact, which is important to realize and easy to verify: iterating by the best response strategy of each player in the power control game is given precisely by (1.2). Given that all the other transmitters transmit at a certain power level, my best response strategy is to pick the power level that is my current power level times the ratio between the target SIR and the current SIR.

Now, consider player 1. If the transmit powers of all the other players become smaller, player 1's strategy space A_1 will be larger. This is the case since there are more transmit powers to pick and still be able to maintain the target SIR, as other players' transmit powers are smaller and the denominator in SIR is smaller. Games with this property (and under some technicalities) are called **submodular**. It is a fact, which we will not have space to prove here, that best response strategies of a submodular game converge.

This game-theoretic approach is one of the ways to prove the convergence of DPC. Another way is through the angle of global optimization and with the help of linear algebra, as we will present in Advanced Material. But first, a small and detailed example.

1.3 Examples

A word about all the examples. They are completely numerical and presented in great detail, so as to alleviate any "symbol-phobia" a reader may have. This means that we have to strike a tradeoff between a realistic size for illustration and a small size so that it does not take up too many pages. In some cases, these examples do not represent the actual scale of the problems while scaling-up is a core difficulty.

Suppose we have four (transmitter, receiver) pairs. Let the channel gains $\{G_{ij}\}$ be given in Table 1.3. As the table suggests, we can also represent these gains in a matrix. You can see that in general $G_{ij} \neq G_{ji}$ because the interference channels do not have to be symmetric.

Receiver of Link	Transmitter of Link 1	2	3	4
1	1	0.1	0.2	0.3
2	0.2	1	0.1	0.1
3	0.2	0.1	1	0.1
4	0.1	0.1	0.1	1

Table 1.3 Channel gains in an example of DPC. The entries are for illustrating the algorithm. They do not represent actual numerical values typically observed in real cellular networks.

Suppose that the initial power level is 1.0 mW on each link, and that the noise on each link is 0.1 mW. Then the initial signal-to-interference ratios are given by

$$\begin{aligned}
\mathrm{SIR}_1[0] &= \tfrac{1\times 1.0}{0.1\times 1.0+0.2\times 1.0+0.3\times 1.0+0.1} = 1.43\\
\mathrm{SIR}_2[0] &= \tfrac{1\times 1.0}{0.2\times 1.0+0.1\times 1.0+0.1\times 1.0+0.1} = 2.00\\
\mathrm{SIR}_3[0] &= \tfrac{1\times 1.0}{0.2\times 1.0+0.1\times 1.0+0.1\times 1.0+0.1} = 2.00\\
\mathrm{SIR}_4[0] &= \tfrac{1\times 1.0}{0.1\times 1.0+0.1\times 1.0+0.1\times 1.0+0.1} = 2.50,
\end{aligned}$$

where we use the formula

$$\mathrm{SIR}_i = \frac{G_{ii}p_i}{\sum_{j\neq i} G_{ij}p_j + n_i},$$

with p_i representing the power level of link i and n_i the noise on link i.

We will use DPC to adjust the power levels. Suppose that the target SIR's are

$$\begin{aligned}
\gamma_1 &= 2.0\\
\gamma_2 &= 2.5\\
\gamma_3 &= 1.5\\
\gamma_4 &= 2.0.
\end{aligned}$$

Then the new power levels are, in mW,

$$\begin{aligned}
p_1[1] &= \tfrac{\gamma_1}{\mathrm{SIR}_1[0]}p_1[0] = \tfrac{2.0}{1.43}\times 1.0 = 1.40\\
p_2[1] &= \tfrac{\gamma_2}{\mathrm{SIR}_2[0]}p_2[0] = \tfrac{2.5}{2.00}\times 1.0 = 1.25\\
p_3[1] &= \tfrac{\gamma_3}{\mathrm{SIR}_3[0]}p_3[0] = \tfrac{1.5}{2.00}\times 1.0 = 0.75\\
p_4[1] &= \tfrac{\gamma_4}{\mathrm{SIR}_4[0]}p_4[0] = \tfrac{2.0}{2.5}\times 1.0 = 0.80.
\end{aligned}$$

Now each receiver calculates the new SIR and feeds it back to its transmitter:

$$\begin{aligned}
\mathrm{SIR}_1[1] &= \frac{1\times 1.40}{0.1\times 1.25+0.2\times 0.75+0.3\times 0.8+0.1} = 2.28\\
\mathrm{SIR}_2[1] &= \frac{1\times 1.25}{0.2\times 1.40+0.1\times 0.75+0.1\times 0.8+0.1} = 2.34\\
\mathrm{SIR}_3[1] &= \frac{1\times 0.75}{0.2\times 1.40+0.1\times 1.25+0.1\times 0.8+0.1} = 1.28\\
\mathrm{SIR}_4[1] &= \frac{1\times 0.80}{0.1\times 1.40+0.1\times 1.25+0.1\times 0.75+0.1} = 1.82.
\end{aligned}$$

The new power levels in the next timeslot become, in mW,

$$\begin{aligned}
p_1[2] &= \frac{\gamma_1}{\mathrm{SIR}_1[1]}p_1[1] = \frac{2.0}{2.28}\times 1.40 = 1.23\\
p_2[2] &= \frac{\gamma_2}{\mathrm{SIR}_2[1]}p_2[1] = \frac{2.5}{2.33}\times 1.25 = 1.34\\
p_3[2] &= \frac{\gamma_3}{\mathrm{SIR}_3[1]}p_3[1] = \frac{1.5}{1.28}\times 0.75 = 0.88\\
p_4[2] &= \frac{\gamma_4}{\mathrm{SIR}_4[1]}p_4[1] = \frac{2.0}{1.82}\times 0.80 = 0.88,
\end{aligned}$$

with the corresponding SIRs as follows:

$$\begin{aligned}
\mathrm{SIR}_1[2] &= \frac{1\times 1.23}{0.1\times 1.34+0.2\times 0.88+0.3\times 0.88+0.1} = 1.83\\
\mathrm{SIR}_2[2] &= \frac{1\times 1.34}{0.2\times 1.23+0.1\times 0.88+0.1\times 0.88+0.1} = 2.56\\
\mathrm{SIR}_3[2] &= \frac{1\times 0.88}{0.2\times 1.23+0.1\times 1.34+0.1\times 0.88+0.1} = 1.55\\
\mathrm{SIR}_4[2] &= \frac{1\times 0.88}{0.1\times 1.23+0.1\times 1.34+0.1\times 0.88+0.1} = 1.98.
\end{aligned}$$

Calculating the new power levels again, we have, in mW,

$$\begin{aligned}
p_1[3] &= \frac{\gamma_1}{\mathrm{SIR}_1[2]}p_1[2] = \frac{2.0}{1.83}\times 1.23 = 1.35\\
p_2[3] &= \frac{\gamma_2}{\mathrm{SIR}_2[2]}p_2[2] = \frac{2.5}{2.56}\times 1.34 = 1.30\\
p_3[3] &= \frac{\gamma_3}{\mathrm{SIR}_3[2]}p_3[2] = \frac{1.5}{1.55}\times 0.88 = 0.85\\
p_4[3] &= \frac{\gamma_4}{\mathrm{SIR}_4[2]}p_4[2] = \frac{2.0}{1.98}\times 0.88 = 0.89.
\end{aligned}$$

Then the new SIRs are

$$\begin{aligned}
\mathrm{SIR}_1[3] &= \frac{1\times 1.35}{0.1\times 1.30+0.2\times 0.85+0.3\times 0.89+0.1} = 2.02\\
\mathrm{SIR}_2[3] &= \frac{1\times 1.30}{0.2\times 1.35+0.1\times 0.85+0.1\times 0.89+0.1} = 2.40\\
\mathrm{SIR}_3[3] &= \frac{1\times 0.85}{0.2\times 1.35+0.1\times 1.30+0.1\times 0.89+0.1} = 1.45\\
\mathrm{SIR}_4[3] &= \frac{1\times 0.89}{0.1\times 1.35+0.1\times 1.30+0.1\times 0.85+0.1} = 1.97,
\end{aligned}$$

and the new power levels, in mW, are

$$\begin{aligned}
p_1[4] &= \frac{\gamma_1}{\mathrm{SIR}_1[3]}p_1[3] = \frac{2.0}{2.02}\times 1.35 = 1.33\\
p_2[4] &= \frac{\gamma_2}{\mathrm{SIR}_2[3]}p_2[3] = \frac{2.5}{2.40}\times 1.30 = 1.36\\
p_3[4] &= \frac{\gamma_3}{\mathrm{SIR}_3[3}p_3[3] = \frac{1.5}{1.45}\times 0.85 = 0.88\\
p_4[4] &= \frac{\gamma_4}{\mathrm{SIR}_4[3]}p_4[3] = \frac{2.0}{1.97}\times 0.89 = 0.90.
\end{aligned}$$

We see that the power levels are beginning to converge: p_1, p_2, p_3, p_4 all change

by less than 0.1 mW. The new SIRs are

$$\begin{aligned}
\mathrm{SIR}_1[4] &= \tfrac{1\times 1.33}{0.1\times 1.36+0.2\times 0.88+0.3\times 0.90+0.1} = 1.96\\
\mathrm{SIR}_2[4] &= \tfrac{1\times 1.36}{0.2\times 1.33+0.1\times 0.88+0.1\times 0.90+0.1} = 2.49\\
\mathrm{SIR}_3[4] &= \tfrac{1\times 0.88}{0.2\times 1.33+0.1\times 1.36+0.1\times 0.90+0.1} = 1.49\\
\mathrm{SIR}_4[4] &= \tfrac{1\times 0.90}{0.1\times 1.33+0.1\times 1.36+0.1\times 0.88+0.1} = 1.97.
\end{aligned}$$

Iterating one more time, the new power levels, in mW, are

$$\begin{aligned}
p_1[5] &= \tfrac{\gamma_1}{\mathrm{SIR}_1[4]}p_1[4] = \tfrac{2.0}{1.96}\times 1.33 = 1.37\\
p_2[5] &= \tfrac{\gamma_2}{\mathrm{SIR}_2[4]}p_2[4] = \tfrac{2.5}{2.49}\times 1.36 = 1.36\\
p_3[5] &= \tfrac{\gamma_3}{\mathrm{SIR}_3[4]}p_3[4] = \tfrac{1.5}{1.49}\times 0.88 = 0.89\\
p_4[5] &= \tfrac{\gamma_4}{\mathrm{SIR}_4[4]}p_4[4] = \tfrac{2.0}{1.97}\times 0.90 = 0.92,
\end{aligned}$$

with corresponding SIRs:

$$\begin{aligned}
\mathrm{SIR}_1[5] &= \tfrac{1\times 1.37}{0.1\times 1.36+0.2\times 0.89+0.3\times 0.92+0.1} = 1.99\\
\mathrm{SIR}_2[5] &= \tfrac{1\times 1.36}{0.2\times 1.37+0.1\times 0.89+0.1\times 0.92+0.1} = 2.45\\
\mathrm{SIR}_3[5] &= \tfrac{1\times 0.89}{0.2\times 1.37+0.1\times 1.36+0.1\times 0.92+0.1} = 1.48\\
\mathrm{SIR}_4[5] &= \tfrac{1\times 0.92}{0.1\times 1.37+0.1\times 1.36+0.1\times 0.89+0.1} = 1.99.
\end{aligned}$$

All the SIRs are now within 0.05 of the target. The power levels keep iterating, taking the SIRs closer to the target. Figure 1.5 shows the graph of power level versus the iterations. After about 20 iterations, the change is too small to be seen on the graph; the power levels at that time are

$$\begin{aligned}
p_1 &= 1.46 \text{ mW}\\
p_2 &= 1.46 \text{ mW}\\
p_3 &= 0.95 \text{ mW}\\
p_4 &= 0.97 \text{ mW}.
\end{aligned}$$

The resulting SIR's are shown in Figure 1.6. We get very close to the target SIRs, by visual inspection, after about 50 iterations.

While we are at this example, let us also walk through a compact, matrix representation of the target SIR constraints. This will be useful in the next section.

If the target SIR γ_i is achieved or exceeded by $\{p_i\}$, we have

$$\frac{G_{ii}p_i}{\sum_{j\neq i} G_{ij}p_j + n_i} \geq \gamma_i,$$

for $i = 1, 2, 3, 4$. Multiplying both sides by $\sum_{j\neq i} G_{ij}p_j + n_i$ and dividing by G_{ii},

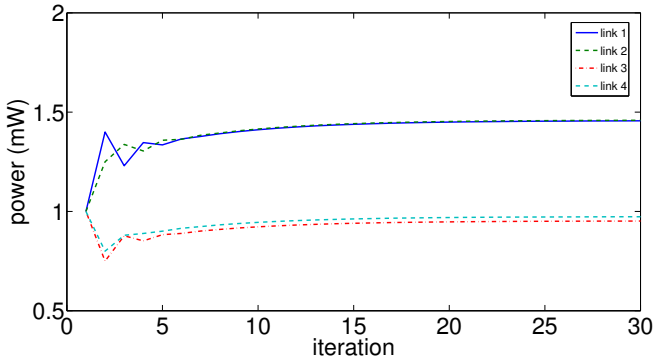


Figure 1.5 Convergence of power levels in an example of DPC.

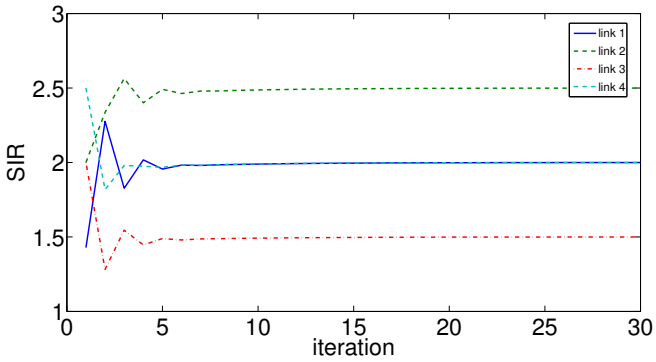


Figure 1.6 Convergence of SIRs in an example of DPC.

we have

$$p_i \geq \frac{\gamma_i}{G_{ii}} \left(\sum_{j \neq i} G_{ij} p_j + n_i \right),$$

which can be written as

$$p_i \geq \gamma_i \sum_{j \neq i} \frac{G_{ij}}{G_{ii}} p_j + \frac{\gamma_i}{G_{ii}} n_i. \tag{1.4}$$

Now we define the variable vector

$$\mathbf{p} = \begin{bmatrix} p_1 \\ p_2 \\ p_3 \\ p_4 \end{bmatrix},$$

and a constant vector

$$\mathbf{v} = \begin{bmatrix} \frac{\gamma_1 n_1}{G_{11}} \\ \frac{\gamma_2 n_2}{G_{22}} \\ \frac{\gamma_3 n_3}{G_{33}} \\ \frac{\gamma_4 n_4}{G_{44}} \end{bmatrix} = \begin{bmatrix} \frac{2.0 \times 0.1}{1.0} \\ \frac{2.5 \times 0.1}{1.0} \\ \frac{1.5 \times 0.1}{1.0} \\ \frac{2.0 \times 0.1}{1.0} \end{bmatrix} = \begin{bmatrix} 0.20 \\ 0.25 \\ 0.15 \\ 0.20 \end{bmatrix}.$$

Define also a 4×4 diagonal matrix $\mathbf{D}$ with γ_i on the diagonal, and another 4×4 matrix $\mathbf{F}$ where $F_{ij} = \frac{G_{ij}}{G_{ii}}$ for $i \neq j$, and the diagonal entries of F are zero. Plugging in the numbers, we have

$$\mathbf{D} = \begin{bmatrix} 2.0 & 0 & 0 & 0 \\ 0 & 2.5 & 0 & 0 \\ 0 & 0 & 1.5 & 0 \\ 0 & 0 & 0 & 2.0 \end{bmatrix},$$

$$\mathbf{F} = \begin{bmatrix} 0 & 0.1 & 0.2 & 0.3 \\ 0.2 & 0 & 0.1 & 0.1 \\ 0.2 & 0.1 & 0 & 0.1 \\ 0.1 & 0.1 & 0.1 & 0 \end{bmatrix}.$$

We can now rewrite (1.4) as

$$\mathbf{p} \geq \mathbf{DFp} + \mathbf{v} = \begin{bmatrix} 0 & 0.20 & 0.40 & 0.60 \\ 0.50 & 0 & 0.25 & 0.25 \\ 0.30 & 0.15 & 0 & 0.15 \\ 0.20 & 0.20 & 0.20 & 0 \end{bmatrix} \mathbf{p} + \begin{bmatrix} 0.20 \\ 0.25 \\ 0.15 \\ 0.20 \end{bmatrix},$$

where $\geq$ between two vectors (of equal length) simply represents component-wise inequality between the corresponding entries of the two vectors.

We can check that the power levels in the last iteration shown above satisfy this inequality tightly:

$$\begin{bmatrix} 1.46 \\ 1.46 \\ 0.95 \\ 0.97 \end{bmatrix} \geq \begin{bmatrix} 0 & 0.20 & 0.40 & 0.60 \\ 0.50 & 0 & 0.25 & 0.25 \\ 0.30 & 0.15 & 0 & 0.15 \\ 0.20 & 0.20 & 0.20 & 0 \end{bmatrix} \begin{bmatrix} 1.46 \\ 1.46 \\ 0.95 \\ 0.97 \end{bmatrix} + \begin{bmatrix} 0.20 \\ 0.25 \\ 0.15 \\ 0.20 \end{bmatrix} = \begin{bmatrix} 1.46 \\ 1.46 \\ 0.95 \\ 0.97 \end{bmatrix}.$$

We will use this matrix representation as we go from problem representation (1.3) to problem representation (1.5) in the next section.

1.4 Advanced Material

1.4.1 Iterative power method

We can generalize the vector notation in the last example. Let $\mathbf{1}$ represent a vector of 1s, so the objective function is simply $\sum_i p_i$. Let $\mathbf{I}$ be the identity matrix, and $\mathbf{D}(\gamma)$ a diagonal matrix with the diagonal entries being the target SIR values $\{\gamma_i\}$, $\mathbf{F}$ is a matrix capturing the given channel conditions: $F_{ij} = \frac{G_{ij}}{G_{ii}}$ if $i \neq j$, and 0 along the diagonal: $F_{ii} = 0$, and the constant vector $\mathbf{v}$ captures normalized noise:

$$\mathbf{v} = \left(\frac{\gamma_1 n_1}{G_{11}}, \frac{\gamma_2 n_2}{G_{22}}, \ldots, \frac{\gamma_N n_N}{G_{NN}} \right)^T.$$

We will soon see why this shorthand notation is useful.

Equipped with the notation above, we can represent the target SIR constraints in problem (1.3) as:

$$\mathbf{p} \geq \mathbf{D}(\gamma)\mathbf{F}\mathbf{p} + \mathbf{v},$$

and further group all the terms involving the variables $\mathbf{p}$. The linear programming problem we have becomes

$$\begin{array}{ll} \text{minimize} & \mathbf{1}^T\mathbf{p} \\ \text{subject to} & (\mathbf{I} - \mathbf{D}(\gamma)\mathbf{F})\mathbf{p} \geq \mathbf{v} \\ \text{variables} & \mathbf{p}. \end{array} \tag{1.5}$$

You should verify that indeed problem (1.3) and problem (1.5) are equivalent, by using the definitions of SIR, of matrices $(\mathbf{D}, \mathbf{F})$, and of vector $\mathbf{v}$.

Linear programming problems are conceptually and computationally easy to solve in general. Our special case here has even more structure. This $\mathbf{DF}$ matrix is a **non-negative matrix**, since all entries of the matrix are non-negative numbers. These matrices are a powerful modeling tool in linear algebra, with applications from economics to ecology. They are well-studied in matrix analysis through the **Perron-Frobenius theory**.

If the largest eigenvalue of the $\mathbf{DF}$ matrix, denoted as $\rho(\mathbf{DF})$, is less than 1, then the following three statements are true:

(a) We can guarantee that the set of target SIRs can indeed be achieved simultaneously, which makes sense since $\rho(\mathbf{DF}) < 1$ means that the $\{\gamma_i\}$ in $\mathbf{D}$ are not "too big," relative to the given channel conditions captured in $\mathbf{F}$.

(b) We can invert the matrix defining the linear constraints in our optimization problem (1.5): solve the problem by computing $(\mathbf{I}-\mathbf{DF})^{-1}\mathbf{v}$. But of course there is no easy way to directly run this matrix inversion distributively across the MSs.

(c) The inversion of the matrix can be expressed as a sum of terms, each term a multiplication of matrix $\mathbf{DF}$ by itself. More precisely: $(\mathbf{I}-\mathbf{DF})^{-1} = \sum_{k=0}^{\infty}(\mathbf{DF})^k$. This is an infinite sum, so we say that the partial sum of K terms, $\sum_{k=0}^{K}(\mathbf{DF})^k$, will converge as K becomes very large. Furthermore, the tail term in this sum, $(\mathbf{DF})^k$, approaches 0 as k becomes large:

$$\lim_{k\to\infty}(\mathbf{DF})^k = 0. \tag{1.6}$$

The key insight is that we want to invert a matrix $(\mathbf{I}-\mathbf{DF})$, because that will lead us to a power-minimal solution to achieving all the target SIRs:

$$\mathbf{p}^* = (\mathbf{I} - \mathbf{DF})^{-1}\mathbf{v} \tag{1.7}$$

is a solution of problem (1.3), *i.e.*, for any solution $\hat{\mathbf{p}}$ satisfying the constraints in problem (1.3), $\mathbf{p}^*$ is better:

$$\hat{\mathbf{p}} \geq \mathbf{p}^*. \tag{1.8}$$

Now, the matrix inversion step is not readily implementable in a distributed fashion, as we need in cellular network power control. Fortunately, it can be achieved by applying the following update.

(1) First, $\mathbf{p}^* = (\mathbf{I} - \mathbf{DF})^{-1}\mathbf{v}$ can be represented as a power series, as stated in Statement (c) above:

$$\mathbf{p}^* = \sum_{k=0}^{\infty} (\mathbf{DF})^k \mathbf{v}. \tag{1.9}$$

(2) Then, you can readily check that the following iteration over time gives exactly the above power series (1.9), as time t goes on:

$$\mathbf{p}[t+1] = \mathbf{DFp}[t] + \mathbf{v}. \tag{1.10}$$

One way to check this is to substitute the above recursive formula of $\mathbf{p}$ (1.10) all the way to $\mathbf{p}]0]$, the initialization of the iteration. Then, at any time t, we have

$$\mathbf{p}[t] = (\mathbf{DF})^t \mathbf{p}[0] + \sum_{k=0}^{t} (\mathbf{DF})^k \mathbf{v},$$

which converges, as $t \to \infty$, to:

$$\lim_{t\to\infty} \mathbf{p}[t] = 0 + \sum_{k=0}^{\infty} (\mathbf{DF})^k \mathbf{v} = \mathbf{p}^*,$$

since $(\mathbf{DF})^t \mathbf{p}[0]$ approaches 0 as $t \to \infty$. This also shows that it does not matter what the initialization vector $\mathbf{p}[0]$ is. Its effect will be washed away as time goes on. So, we know (1.10) is right.

(3) Finally, rewrite the vector form of update equation (1.10) in scalar form for each transmitter i, and you will see that it is exactly the DPC distributed algorithm (1.2):

$$p_i[t+1] = \frac{\gamma_i}{\mathrm{SIR}_i[t]} p_i[t], \quad \text{for each } i.$$

We just completed a development and convergence analysis of the DPC as the solution of a global optimization problem through the language of linear algebra. In general, for any square matrix $\mathbf{A}$, the following statements are equivalent:

1. The largest modulus eigenvalue is less than 1.
2. The limit of this matrix multiplying itself is 0: $\lim_{k\to\infty} \mathbf{A}^k = 0$.
3. The infinite sum of powers $\sum_{k=0}^{\infty} \mathbf{A}^k$ exists and equals $(\mathbf{I} - \mathbf{A})^{-1}$.

What we saw in DPC is an iterative **power method** (the word "power" here has nothing to do with transmit powers, but instead refers to raising a matrix to some power, *i.e.*, multiplying a matrix by itself many times). It is a common method used to develop iterative algorithms arising from a linear systems model. First we formulate the problem as a linear program, define the solution of the problem as the solution to a linear equation, implement the matrix inversion through a sequence of matrix powers, turn each of those steps into an easy computation at each time slot, and finally achieve the matrix inversion through an iteration in time. This will be used again when we talk about Google's pagerank algorithm in Chapter 3.

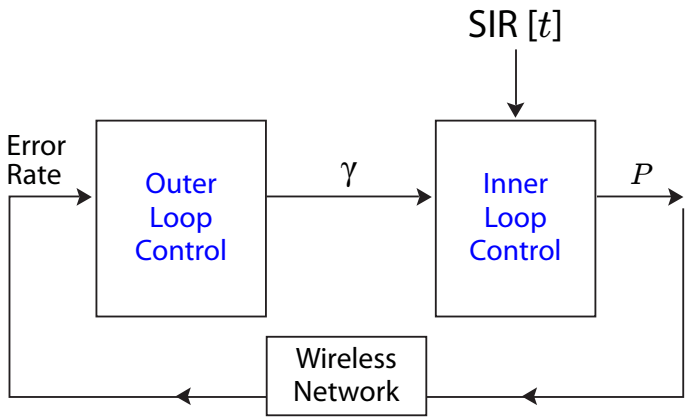


Figure 1.7 Inner and outer loops of power control in cellular networks. The inner loop takes in a fixed target SIR, compares with the current SIR, and updates the transmit power. The outer loop adjusts the target SIR based on the performance measured over a longer timescale. We have focused on the inner loop power control.

1.4.2 Outer loop power control

As shown in the block diagram in Figure 1.7, in cellular networks there are two timescales of power control. What we have been discussing so far is the **inner loop power control**. Nested outside of that is the **outer loop power control**, where the target SIRs $\{\gamma_i\}$ are determined.

One standard way to determine target SIRs is to measure the received signal quality, in terms of decoding error probabilities, at the receiver. If the error rate is too high, the target SIR needs to be increased. And if error rate is lower than necessary, the target SIR can be reduced.

Alternatively, we can also consider optimizing target SIRs as optimization *variables*. This is particularly useful for 3G and 4G networks where data traffic dominates voice traffic on the cellular networks. Higher SIR can provide a higher rate at the same signal quality. But every user wants to achieve higher SIR. So we need to model this either as a network-wide optimization, maximizing the sum of all users' payoff functions, or as a game, with each user maximizing its own payoff function of SIR.

Further Reading

This chapter introduces several foundational methodologies: optimization, games, and algorithms. As a result, there are many interesting readings.

1. The DPC algorithm in the core of this chapter appeared in the following seminal paper:

[FM93] G. J. Foschini and Z. Miljanic, "A simple distributed autonomous power control algorithm and its convergence," *IEEE Transactions on Vehicular Networks*, vol. 42, no. 3, pp. 641-646, November 1993.

2. Much more discussion on power control algorithms in cellular networks can be found in the following monograph:

[Chi+08] M. Chiang, P. Hande, T. Lan, and C. W. Tan, *Power Control for Cellular Wireless Networks*, Foundation and Trends in Networking, NOW Publisher, 2008.

3. A standard reference on linear algebra is the following mathematics textbook, which includes a chapter on non-negative matrices and Perron Frobenius theory:

[HJ90] R. A. Horn and C. R. Johnson, *Matrix Analysis*, Cambridge University Press, 1990.

4. A standard textbook on linear programming is

[BT97] D. Bertsimas and J. Tsitsiklis, *Introduction to Linear Optimization*, Athena Scientific, 1997.

5. There are many textbooks on game theory, in all types of styles. A comprehensive introduction is

[Mye97] R. B. Myerson, *Game Theory: Analysis of Conflict*, Harvard University Press, 1997.

Problems

1.1 *Distributed power control* ⋆

(a) Consider 3 pairs of transmitters and receivers in a cell, with the following channel gain matrix $\mathbf{G}$ and noise of 0.1 mW for all the receivers. The target SIRs are shown below. With an initialization of all transmit powers at 1 mW, run DPC for 10 iterations and plot the evolutions of transmit powers and of received SIRs.

You can use any programming language to calculate, or even write the steps out by hand. Please turn in the code or all the hand-written steps, respectively.

$$\mathbf{G} = \begin{bmatrix} 1 & 0.1 & 0.3 \\ 0.2 & 1 & 0.3 \\ 0.2 & 0.2 & 1 \end{bmatrix}, \qquad \gamma = \begin{bmatrix} 1 \\ 1.5 \\ 1 \end{bmatrix}.$$

(b) Now suppose the power levels for logical links 1,2,3 have converged as in the previous problem. A new pair of transmitter and receiver, labeled as logical link 4, shows up in the same cell, with an initial transmit power of 1 mW and demands a target SIR of 1. The new channel gain matrix is shown below. Similar to what you did for Problem 1 above, show what happens in the next 10 timeslots? What happens at the new equilibrium?

$$\mathbf{G} = \begin{bmatrix} 1 & 0.1 & 0.3 & 0.1 \\ 0.2 & 1 & 0.3 & 0.1 \\ 0.2 & 0.2 & 1 & 0.1 \\ 0.1 & 0.1 & 0.1 & 1 \end{bmatrix}.$$

1.2 *Power control infeasibility* ⋆⋆

Consider a 3-link system with the link gains G_{ij} shown below. The receivers request $\mathrm{SIR}_1 = 1, \mathrm{SIR}_2 = 2, \mathrm{SIR}_3 = 1$. The noise $n_i = 0.1,\ \forall i$. Show that this set of target SIRs is infeasible.

$$\mathbf{G} = \begin{bmatrix} 1 & 0.5 & 0.5 \\ 0.5 & 1 & 0.5 \\ 0.5 & 0.5 & 1 \end{bmatrix}$$

1.3 *Zero sum game* ⋆

In the following 2-user game, the payoffs of the users are exactly negative of each other in all the combinations of strategies. This models an extreme case of competition, and is called **zero-sum game**. Is there any pure strategy equilibrium? How many are there?

	a	b
a	(2,-2)	(3,-3)
b	(3,-3)	(4,-4)

1.4 *Mechanism design* ⋆⋆

Consider the game below. There are two players, each with two strategies, and the payoffs are shown below. Consider only pure strategy equilibria.

	a	b
a	(0,2)	(2,0)
b	(6,0)	(3,2)

(a) Is there a Nash equilibrium, and if so, what is it?

(b) We want to make this game a "better" one. What entries in the table

would you change to make the resulting Nash equilibrium unique and socially optimal?

This is an example of **mechanism design**: change the game so as to induce the players to move to a desirable equilibrium. We will see a lot more of mechanism design in future chapters.

1.5 *Repeating prisoner's dilemma* ⋆⋆⋆

(a) Suppose the two prisoners know that they will somehow be caught in the same situation 5 more times in future years. What will be each prisoner's strategy in choosing between confession and no confession?

(b) Suppose the two prisoners have infinite lifetimes, and there is always 90% chance that they will be caught in the same situation after each round of this game. What will be each prisoner's strategy now?

2 How does Google sell ad spaces?

2.1 A Short Answer

Much of the web services and online information is "free" today because of the advertisements shown on the websites. It is estimated that the online ad industry reached $26 billion in 2010. Compared to traditional media, online advertisements' revenue ranked right below TV and above newspaper.

In the early days of the web, *i.e.*, the mid 1990s, online advertisements were sold as banners on a per-thousand-impression basis. In 1997, GoTo (later became Overture) started selling advertisement spaces on a per-click basis. This middle ground between ad revenue (what the website cares about) and effectiveness of ad (what the advertisers care about) became a commonly accepted foundation for online advertising.

With the rise of Google came one of the most stable online ad market segments: **search ads**, also called *sponsored search*. In 2002, Google started the AdWords service where you can create your ad, attach keywords to it, and send it to Google's database. When someone searches for a keyword, Google will return a list of search results, as well as a list of ads on the right panel, or even the main panel, if that keyword matches any of the keywords of ads in its database. This process takes place continuously and each advertiser can adjust her bids frequently. There are often many ad auctions happening at the same time too. We will skip these important factors in the basic models in this chapter, focusing just on a single auction.

Now the question is: where will your ad appear on the list? We all know the order of appearance makes a big difference. You will have to pay more money to have your ad higher in the list. For example, when I searched "Banff National Park" on Google in September 2011, and I saw an ad for `www.banfflakelouise.com`, a vacation planning company. This ad was right on top of the main panel, above all the search results on websites and images of Banff National Park. (By the way, how those "real" search results are ordered is the subject of the next chapter.) You also see a list of ads on the right panel, starting with the top one for `www.rockymountaineer.com`, a tourist train company. These two companies probably get most of the clicks, and pay more than the other advertisers for each click. The rest of this chapter delves into the auction methods that allocate these ad spaces based on how much each advertiser is willing to pay.

When will these advertisers need to pay Google? Only when someone clicks on the link and visits their websites (like I just did, thus contributing to Google's revenue). The average number of times that a viewer of the search result page clicks an ad link, over say one hour, is called the **click through rate**. In a general webpage layout, it may be difficult to rank ad spaces by their positions along a line, but we can always rank them by their click through rates. Let us say the payment by advertisers to Google is proportional to the click through rates.

What is in it for the advertisers then? Their revenue derived from placing this particular ad is the product of two things: C, the number of clicks (per unit time, say, one hour), and R, the average revenue (in dollars) generated from each click.

- Let us assume that the number of clicks per day actually observed is indeed the estimated click through rate, both denoted as C. This is of course not true in general, but it is a reasonable assumption to make first. We also assume that C is independent of the content in the actual advertisement placed, again a shaky assumption to make the model more tractable.
- As for the average revenue R generated from each click (averaged over all the clicks), that highly depends on the nature of the goods or services being advertised and sold. R for each ad space buyer is assumed to be independent of which ad space she ends up buying, a more reasonable assumption than the one on the independence of C of the advertisement.

This product $C \times R$, the expected revenue from a particular ad space by a buyer, is the **valuation** of the ad space to the buyer. For example, if C is 20 clicks per hour for an ad space and R is \$8 generated per click for an ad space buyer, the valuation of that space to this buyer is \$160. For multiple ad spaces, the valuation of each buyer is a *vector*, one entry per ad space.

In this discussion, there is one seller, which is Google, and many buyers/bidders, which are the advertisers, and many "goods," which are the ad spaces. Each bidder can bid for the ad spaces, and Google will then allocate the ad spaces among the bidders according to some rule, and charge the bidders accordingly.

This process is an **auction**. In general, you can have S sellers, N bidders, and K items in an auction. We will only consider the case with $S = 1$. Ad space auction is a special case of general auctions. Auctions can be analyzed as games, *i.e.*, with a set of players, a strategy set per player, and a payoff function per player.

Depending on the rules, each bidder may choose to bid in different ways, possibly bidding her true valuation of the ad spaces. It would be nice to design the rules so that such a **truthful bidding** behavior is encouraged. But Google has other considerations too, such as maximizing its total revenue: the sum of the revenue from each bidder, which is in turn the product of the number of clicks (actually observed in real time) times the per-click charge (determined during the auction).

Before we analyze a K-item auction, we will first study auctions with only

$K = 1$ item. There are two main types of such 1-item auctions: ascending-price and descending-price. These require a public venue for announcing the bids, have been used since the Roman Empire, and are quite intuitive to most of us.

- In an **ascending price auction**, an auctioneer announces a base price, and then each bidder can raise her hand to bid a higher price. This price war keeps escalating until one bidder bids a price, and no other bidder raises a hand anymore as the auctioneer calls out "gone." The last bidder is the winning bidder, and she pays the price she bid in the last round.
- In a **descending price auction**, an auctioneer announces a high price first, so high that no bidder is willing to accept it. The auctioneer then starts to lower the price, until there is one bidder who shouts out OK. That bidder is allocated the item, and pays the price announced when she said OK.

The alternative to a public venue is private disclosure of bids, called **sealed envelope** auctions. This is much more practical in many settings, including selling ad spaces by Google and auctioning goods on eBay. There are two types of such auctions, but it turns out that their results are essentially equivalent to the two types of open auctions we just discussed.

Each bid b_i is submitted by bidder i in a sealed envelope. All bids are then revealed simultaneously to the auctioneer, who will then decide (1) the allocation and (2) how much to charge each item. The allocation part is easy: the highest bidder gets the item. But the amount charged can vary:

- In a **first price auction**, the winner pays the highest bid, *i.e.*, her own bid.
- In a **second price auction**, the winner pays the second highest bid, *i.e.*, the bid from the next highest bidder.

Second price auction sounds "wrong." If I know I will be paying the next highest bid, why not bid extremely high so that I can win the item, and then possibly pay a much lower price for it? It turns out this intuition *itself* is wrong. The assumption of "much lower prices by other bidders" does not hold when everyone engages in the same strategic thinking.

Instead, a second price auction is equivalent to the highly intuitive ascending price auction, and can induce truthful bidding behavior from the bidders. That is why second price auction is used so often, from auctioning major municipal projects to auctioning wireless spectrum.

Finally, we come back to auctions of K items (still with 1 seller and N bidders). If we follow the basic mechanism of second price auction, we obtain what is called the **Generalized Second Price** (GSP) for ad space auction: the ith ad space goes to the bidder that puts in the ith highest bid, and the charge, per click through rate, is the $(i+1)$th bid. If the webpage layout shows the ads vertically, the advertiser in a given ad space is paying the price that is the same as the bid from the advertiser in the ad space *right below* hers. This simple method is used by Google in selling its ad spaces.

But it turns out GSP is *not* an auction that induces truthful bidding, and there

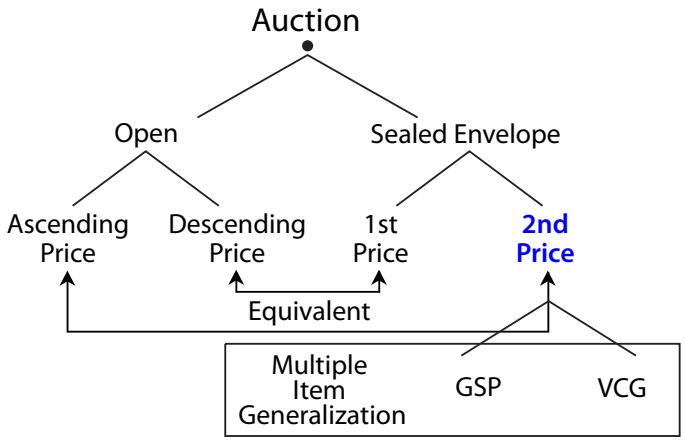


Figure 2.1 A taxonomy of major types of auction in this chapter. Second price sealed envelope is equivalent to (a certain simpler version of) ascending price open auction, and can be generalized in two different ways to multiple-item auctions: a simple extension to Generalized Second Price (GSP) auction and a more sophisticated extension to Vickrey Clarke Groves (VCG) auction that preserves the truthful bidding property.

can be many Nash equilibria if we analyze it as a game. An alternative is the **Vickrey Clarke Groves** (VCG) auction, which actually extends second pricing auction's property of truthful bidding to multiple-item auctions. VCG charges based on **externalities**, a principle that we will see many times throughout the book. The relationships between these types of auction are summarized in Figure 2.1.

Throughout the chapter, we focus on the simplest case, where there is a single round of bidding. In reality, there are multiple related bids going on at the same time, *e.g.*, `www.banfflakelouise.com` may be bidding for multiple related keywords "Banff," "Lake Louise," and "Canadian vacation" simultaneously. In a homework problem, we will go into a little more detail on one aspect of simultaneous auction in the context of spectrum auctioning.

2.2 A Long Answer

2.2.1 When do we need auctions?

No matter which format, an auction runs a resource allocation process. It allocates items among bidders and sets the prices. We assume each bidder has a valuation of the item, and that the valuation is private and independent. **Private valuation** means that the value is unknown to others (all the other bidders and

the auctioneer). **Independent valuation** means that one bidder's valuation does not depend on other bidders' valuations.

You will see that all assumptions are false, which is *why* we call them "assumptions" rather than "facts." But some assumptions are so false that the theory built on them loses predictive power. In this chapter, the private and independent valuation assumptions for auction will be used quite fruitfully, but it is still worthwhile to point out that they may be false in reality.

- In many instances, valuations often *depend* on each other, especially when there is a secondary market for you to resell the items. If you are bidding on a foreclosure house, you are probably dealing with dependent valuation.
- Valuations are sometimes *public*. In fact, some eBay bidding strategies attempt to reveal others' valuations, a particularly helpful strategy when you do not know how to value an item and you think other bidders might be experts who know more.
- Valuations in some cases are actually *not precisely known* even to the bidder herself.

There are several criteria to compare the different outcomes of an auction.

- One is seller's revenue, which clearly the seller would like to maximize. It often turns out that the revenue-maximizing strategy is very hard to characterize.
- Another criterion is the sum of payoffs received by each bidder, across all the bidders. This will become clearer as we model auctions as games. There is a tradeoff between seller revenue and bidders' payoff.
- A third criterion often used is truthful bidding, a property of an auction where each bidder bids her true valuation.

2.2.2 Auction as a game

We can view an auction as a game:

1. The set of players is the set of bidders, indexed by i.
2. The strategy is the bid b_i of each bidder, and each has a strategy space being the range of bids that she might put forward.
3. Each bidder's payoff function, U_i, is the difference between her valuation v_i of the item and the price p_i she has to pay, if she wins the auction:

$$U_i(\mathbf{b}) = v_i - p_i(\mathbf{b}).$$

Here, the coupling of the bidder's bidding choices is shown through the dependence of U_i on the entire *vector* $\mathbf{b}$, even though the valuation is independent: v_i only depends on i. Different auction rules lead to different $p_i(\mathbf{b})$, that is the mechanism design part. For a given mechanism, each bidder will pick her b_i to maximize U_i.

On the other hand, the payoff is

$$U_i(\mathbf{b}) = 0$$

if bidder i loses the auction, since she is not paying anything nor getting the item.

Obviously, winning or losing the auction is also an outcome determined by $\mathbf{b}$.

In the case of a first price sealed envelope auction, the price p_i one has to pay (when winning the auction) is her own bid value b_i, since that is the highest bid. So the payoff for bidder i, when winning, is $v_i - b_i$. From this bidder's perspective, she wants to pick the "right" b_i: not so big that the payoff is small (possibly zero) when winning, and not so small that she does not win the auction at all and receives 0 payoff. It is not easy to solve this optimization since it involves unknown valuation and the strategies of all other bidders. But one thing is clear: she should bid less than her true valuation, for otherwise the best payoff she can receive is zero.

In the case of a second price sealed envelope auction, the price p_i one has to pay (when winning the auction) is the second highest price, *i.e.*, the bid from say bidder j. The payoff is $v_i - b_j$ if bidder i wins, and 0 otherwise. In this game, it turns out that, *no matter* what other bidders might bid, each bidder can maximize her payoff by bidding her true valuation: $b_i = v_i$, *i.e.*, truthful bidding is a dominant strategy for the game. In other words, if $p_i(\mathbf{b})$ is the second highest entry in $\mathbf{b}$, then setting $b_i = v_i$ maximizes U_i, no matter what the values of the rest of the entries in $\mathbf{b}$ are.

2.2.3 Single item auction: Second price

There are several ways to view the somewhat counter-intuitive result of bidding behavior in a second price auction. Fundamentally, it is precisely the *decoupling* of the payoff amount (how much you gain if you win) from the auction result (whether you win at all) that induces each bidder to bid her true valuation. This is also what happens in the familiar format of an open auction with ascending price. Your bid determines how long you will stay in the price war. But when it stops, you pay the bid of the next highest bidder plus a small amount capped by the minimum increment per new bid (unless you overbid much more than the minimum increment), for that is the auctioneer's announcement when the price war stops as a result of the next highest bidder dropping out.

We can readily convince ourselves that truthful bidding is a dominant strategy. Say you want to bid your true valuation: $b = v$. Suppose that, as your advisor, I suggest lowering that bid to be something less than v. Call this new bid $\tilde{b}$. You should realize that such an action will only change the outcome (auction result or payment amount) if the next highest bid, say user 2's bid, b_2, is between b and $\tilde{b}$: $b > b_2 > \tilde{b}$. And in this case, you lose the auction, which you could have

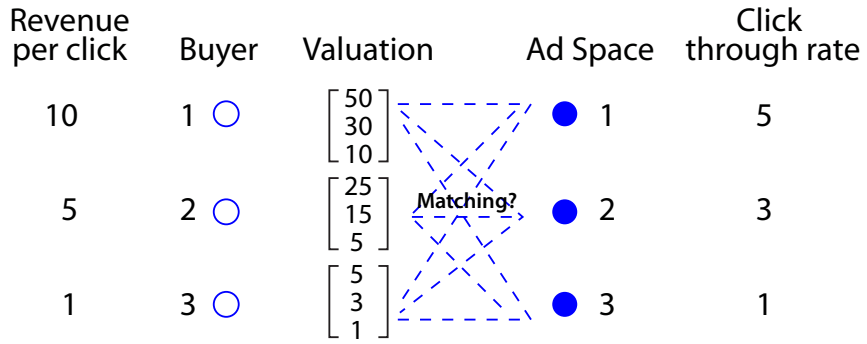


Figure 2.2 An example of 3 bidders and 3 ad spaces. Each bidder's valuation is a vector now. Each valuation vector is proportional to the vector of click through rates $[C_1, C_2, C_3]$, where the proportionality constant is the average revenue per click, R, for that bidder of ad space. We assume that the click through rates are independent of the content of the ad placed at each position. If the bidders and ad spaces are listed in descending order of their R and C, and each player bids true valuations, GSP simply matches "horizontally".

won and received a positive payoff of $v - b_2 = b - b_2 > 0$. So you would rather not take my advice to lower your bid.

Suppose I suggest raising your bid to be something more than v. Call this new bid $\hat{b}$. Again, such an action will only change the outcome if the highest bid, let us say user 2's bid, b_2, is in between b and $\hat{b}$: $\hat{b} > b_2 > b$. And in this case, you now win the auction, but receive a negative payoff, as you end up paying more than you value the item: $v - b_2 = b - b_2 < 0$. Had you bid b, you would have lost and received a payoff of 0, a better outcome when compared to a negative payoff. So you would rather not take my advice to raise your bid.

Therefore, no matter what other bidders do, you would rather neither lower nor raise your bid. You would rather simply bid v.

This argument seals the math, but what is the intuition behind the math? For example, while it is clear why first price is not truthful-bid-inducing, what about third price? We will see in Advanced Material that the intuition is that second price here can capture the negative externality, the "damage" done by the winner to the other bidders.

2.2.4 Multiple-item auction: Generalized second price (GSP)

So far in this section, we have been examining auctions with only one item to sell. Search ad markets have multiple ad spaces to sell in each auction: multiple items facing multiple bidders. For simplicity, we assume that there is only one

auction going on. Let us say there are three ad spaces in this auction, each with an average click through rate of 5, 3, and 1, as shown in Figure 2.2. And there are three advertisers (bidders of ad spaces), each with a different expected revenue per click, say 10, 5, and 1. Then the valuation of each advertiser is as follows: the first advertiser has valuations of [50, 30, 10] for the three spaces. The second advertiser has valuations of [25, 15, 5], and the third advertiser [5, 3, 1]. The job of a multiple-item auction is to *assign one item to each advertiser*.

If the number of advertisers and the number of ad spaces are different, some advertisers may get no ad space or some ad space will be left unsold. We will consider these cases as simple extensions in a homework problem.

We will later encounter other types of **bipartite graph**, like the one we saw between the auctioned items and bidders in Figure 2.2.

Now, back to multiple-ad-space auctions. A bidder, *i.e.*, an advertiser, i sends in a bid b_i, that is how much she is willing to pay per click. This is actually a significant simplification in ad auction. Since there are multiple ad spaces, why not ask each advertiser to submit many bids, one bid per ad space, effectively presenting a scale of their preferences? We will encounter such vector preferences later. But for ad auction, the industry thought that a scalar representation of this vector preference suffices. Each entry in the valuation vector is just a multiple of this scalar.

So, advertiser i sends in a vector of bids $[b_iC_1, b_iC_2, \ldots b_iC_K]$ for the K ad spaces with clickthrough rate $C_j, j = 1, 2, \ldots, K$. She may *not* pick b_i to be the same as v_i, the expected revenue per click.

In GSP, the ith ad space is allocated to the ith highest bidder. As we will see, this rule of ad space allocation is the same for VCG. But the amount each bidder is charged is different depending on whether GSP or VCG is used, and the strategic thinking in bidders' minds is consequently different too. In GSP, the charges are easy to determine: the ith ad space winner pays the per click price of the $(i + 1)$th highest bidder.

In summary, for GSP:

- The allocation part is easy to see: advertiser i's bidding vector is proportional to the vector of click through rates $[C_1, C_2, \ldots, C_K]$, where the proportionality constant is bid b_i. So the advertiser who sends in the ith highest bid for each click gets the ith most valuable ad space. Advertiser i then drops out of future bidding after winning an ad space.
- The charge part is simply a straight forward extension of the second price approach. (We could have also generalized the first price approach, as Overture did in the late 1990s, but soon realized that it was an unstable mechanism.)

Often, a minimum bid is also mandated, say, \$0.5. So in the above example in Figure 2.2, if all the advertisers bid their true valuations, advertiser 1 is matched to ad space 1, with a price of \$5 per click, advertiser 2 to ad space 2, with a price

of $1 per click, and advertiser 3 to ad space 3 with a price of $0.5 per click, the minimum bid allowed.

We can consider GSP as a game too. Despite the apparent similarities between GSP and second price auction, there are substantial differences. There can be multiple Nash equilibria in the GSP game, and it is possible that none of them involves a truthful bidding. The example in the next section illustrates these properties of GSP. This is in sharp contrast to second price auction's desirable property: truthful bidding as a dominant strategy. For a multiple-item auction that preserves this property, we will have to wait until Advanced Material, where we will describe the VCG auction. It is not "second price" that matters, but charging based on damage caused to others.

2.3 Examples

2.3.1 Single item auction on eBay

Started in 1995 and with over 40 million users now, eBay runs online auctions for all kinds of goods. The eBay auction style largely follows the second price auction, but is not exactly the same. There are four main differences:

- A seller can announce a start price, but can also choose to specify a secret *reserve price*: the minimum price below which she will not sell the good. bidders know the existence of the reserve price but not the actual value. This allows a seller to set a start price low enough to attract traffic but still maintains a minimum revenue. In the rest of this section, for simplicity, we will assume the reserve price is the same as the start price. There is also a *minimal increment* of δ dollars: the winner pays the second highest bid plus this δ (unless that exceeds her own bid, in which she just pays her own bid, the highest bid).
- An eBay auction is *not* sealed envelope. Some information about the current bids is continuously released to the public. This is particularly helpful when some bidders are not sure about the valuation of a good. It also generates more "fun" in the auction process. So what information is displayed? eBay looks at the price the winner would need to pay, had the auction been concluded right now, *i.e.*, the smaller of (1) the current highest bid b_1, and (2) the second highest bid b_2 plus δ. This price is announced in public. The next bid has to exceed it by the minimum increment δ, which becomes the new *ask price* displayed: $\min(b_1, b_2 + \delta) + \delta$.
- It has a fixed and publicly announced time horizon, *e.g.*, 3 days. This *hard closing* rule changes bidding behavior. For example, many bidders choose to wait until the last 10 minutes to enter their bids, knowing that it is the time period that really matters (unless they want to scare other bidders away with a large opening bid). There are even third-party tools for automated "sniping," where bids are sent on your behalf in the last seconds

before closing. It is advantageous to wait if you would like to surprise your competitors or want to learn something about their valuations. It is advantageous not to wait if you would like to scare the competitors away with a very high bid to start with and avoid a dragged-out bidding war.

- It allows automated "proxy agent" bidding to simplify the user interface. A bidder can enter the *maximum bid* she is willing to put in, and then let the proxy run the course. When the bidder is no longer the highest bidder, yet the displayed ask price is less than or equal to this indicated maximum bid, a bid is automatically entered to take the ask price.

Here is an example timeline illustrating an eBay auction with 3 bidders: Alice, Bob, and Chris.

- *Start*
  Sam, the seller, lists a lamp for sale on eBay, with a start price of $5.00 and a duration of 5 days. The minimum increment is $1.00. Reserve price is set to be the same as the start price.
  Highest bid: n/a; Ask price: $5.00

- *Day 1*
  The first bid is from Alice, who uses a proxy agent with the maximum bid set to $12.00. eBay therefore bids $5.00 on Alice's behalf. The ask price becomes $\$5.00 + \$1.00 = \$6.00$.
  Highest bid: Alice, $5.00; Ask price: $6.00

- *Day 2*
  The second bid is from Bob, who bids $8.00 even though the ask price is $6.00. eBay immediately raises bid to $\$8.00 + \$1.00 = \$9.00$ on Alice's behalf, since she is using proxy agent bidding and the ask price is not greater than her maximum bid. The ask price becomes $\min\{\$9.00, \$8.00 + \$1.00\} + \$1.00 = \$10.00$.
  Highest bid: Alice, $9.00; Ask price: $10.00

- *Day 3*
  Bob tries again, bidding $10.50 this time. eBay immediately raises the bid to $\$10.50 + \$1.00 = \$11.50$ on Alice's behalf. The ask price becomes $\min\{\$11.50, \$10.50 + \$1.00\} + \$1.00 = \$12.50$.
  Highest bid: Alice, $11.50; Ask price: $12.50

- *Day 4*
  Bob gets frustrated and raises his bid to $17.50. The highest bidder now changes to Bob and the ask price becomes $\min\{\$17.50, \$11.50 + \$1.00\} + \$1.00 = \$13.50$.
  Highest bid: Bob, $17.50; Ask price: $13.50

- *Day 5*
  It so happens that Chris enters the auction at the last moment and bids $18.00, even though he did not know $17.50 is the current highest bid. The ask price becomes $\min\{\$18.00, \$17.50 + \$1.00\} + \$1.00 = \$19.00$.
  Highest bid: Chris, $18.00; Ask price: $19.00. This displayed ask price gives a hint to Chris that his bid is now the highest one.

- *End*
  Nobody takes the ask price before the auction terminates at the hard closing time announced before. The auction ends and Chris wins the lamp with a price of $\min\{\$18.00, \$17.50 + \$1.00\} = \$18.00$.

The bidding history is summarized in Table 2.1:

Table 2.1 The bidding history and ask price evolution in an example of an eBay auction. There are three bidders, each using a different bidding strategy, to compete for a single item. The auction lasts 5 days.

	Day 1	Day 2	Day 3	Day 4	Day 5
Alice	$5.00	$9.00	$11.50	–	–
Bob	–	$8.00	$10.50	$17.50	–
Chris	–	–	–	–	$18.00
Ask price	$6.00	$10.00	$12.50	$13.50	$19.00

2.3.2 Multiple item auction by GSP in Google

Let us consider the bipartite graph in Figure 2.2 again.

- *Bidding*: Assume truthful bidding, *i.e.*, $\mathbf{b}$ is the same $\mathbf{v}$ for each of the three bidders as shown in the graph. For example, bidder 1 bids $50 (per hour) = $10 per click $\times$ 5 clicks per hour, for ad space 1, $30 for ad space 2, etc.
- *Auction outcome (matching, or allocation)*: By the GSP mechanism, the most valuable ad space is allocated to the highest bidder, which clearly is bidder 1 (since bidder 1 bids 50, bidder 2 bids 25, and bidder 3 bids 5 for this ad space). Similarly, ad space 2 is allocated to bidder 2, and ad space 3 to bidder 3. This matching is not surprising, since we have already ordered the bidders and the ad spaces in descending order.
- *Auction outcome (Charging, or pricing, or payment)*: Assume the actual number of clicks per hour is the same as the estimated (per hour) click through rate: bidder 1 pays the second highest bid, which is $5 (per click). So she pays $\$5 \times 5$ (the second 5 here refers to click through rate of 5 per hour for ad space 1)= $25 per hour for ad space 1. Bidder 2 payers $1 per click. So she pays $1*3=$3 per hour for ad space 2. Bidder 3 pays just the minimum bid, *e.g.*, $0.5 per click as set by Google. So she pays $0.5*1 = $0.5 per hour.

In summary, Google's collected a revenue of $\$25 + 3 + 0.5 = \$28.5$ per hour.

- *Payoffs for each bidder*: Payoff is valuation v minus price p. So bidder 1's payoff is $\$10 - \$5 = \$5$ per click, or equivalently, $\$5 \times 5 = \$25$ per hour. bidder 2's payoff is $\$5 - \$1 = \$4$ per click, or $\$4 \times 3 = \$12$ per hour. bidder 3's payoff is $\$1 - 0.5 = \$0.5$ per hour, or $\$0.5 \times 1 = \$0.5$ per hour.

  In summary, the total social payoff is $\$25 + 12 + 0.5 = \$37.5$ per hour.

2.3.3 Another example on GSP

Suppose there are two ad spaces on a webpage and three bidders. An ad in the first space receives 400 clicks per day, while the second space gets 200. Bidders 1, 2, and 3 have values per click of \$12, \$8, and \$4, respectively.

If all advertisers bid truthfully, then the bids are (in dollars) $[4800, 2400]$ from bidder 1, $[3200, 1600]$ from bidder 2, and $[1600, 800]$ from bidder 3. Bidder 1 wins the first ad space, paying \$8 per click, while bidder 2 wins the second space, paying \$4 per click. Bidder 3 does not get any ad space. If the actual click through rates are the same as estimated ones, payments of bidders 1 and 2 are \$3200 and \$800, respectively. And the payoffs are \$1600 and \$800, respectively.

Truth-telling is indeed an equilibrium in this example, as you can verify that no bidder can benefit by changing her bids.

But in general, truth-telling is not a dominant strategy under GSP. For example, consider a slight modification: the first ad space receives 400 clicks a day, and the second one 300. If all players bid truthfully, then bidder 1's payoff, as before, is (\$12 - \$8) * 400 = \$1600. If, instead, she shades her bid and bids only \$7 per click to get the second ad space, her payoff will be equal to (\$12 - \$4) *300 = \$2400 > \$1600. Therefore, she would bid below her valuation and receive a higher payoff. The difference between the first and second ad space's click through rate is simply not large enough relative to the difference in per click payment for her to bid truthfully.

2.4 Advanced Material

2.4.1 VCG Auction

As we just saw, the GSP auction does not guarantee truthful bidding for multiple-item auctions. If that property is desired, the proper generalization of second price auction is the VCG auction.

To search for the correct intuition, we revisit single item second price auctions. Had the highest bidder not been there in the auction, the second highest bidder would have obtained the item, while the other bidders face the same outcome of losing the auction. So the "damage" done by the highest bidder to the entire system (other than the highest bidder herself) is the valuation of the second

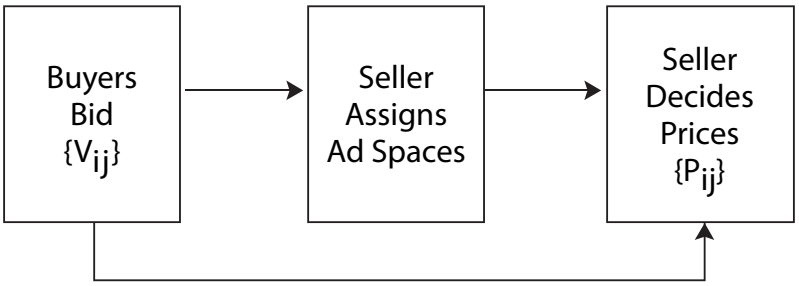


Figure 2.3 Three steps in VCG auctions. First, each bidder sends in a vector of bids for the K items. Then the seller solves an optimization problem of finding the matching between bidders and items and the corresponding maximized value V for the system. Finally, given the matching, the seller determines the price p_{ij} that bidder j pays for item i that she is matched to, based on the amount of negative externality caused.

highest bidder, which is exactly how much the highest bidder is charged. For example, suppose the three valuations are 10, 5, and 1, respectively from Alice, Bob, and Chris. If Alice were not there in the system, Bob would have won the item, and Chris would have lost the auction anyway. Now that Alice is in the system, Bob loses the chance to receive a valuation of 5, and Chris does not suffer any lost valuation. So the total valuation lost to the system (other than Alice) is 5, which is the price Alice pays according to second price auction's rule.

The key idea to realize is that, in a second price auction, the winner is charged for how much her winning reduces the payoffs of the other bidders. This is called the **negative externality** imposed by the winner on all the other bidders. It is this property that enables truthful bidding.

In Chapter 1, power control in wireless networks compensates for negative externality due to signal interference. We will later see negative externalities characterized and compensated for in voting, in tragedy of the commons, and in TCP congestion control. In multiple-item auctions, we are going to again compare two scenarios and take the difference as the quantification of the "damage" done by bidder i:

- The revenue generated by Google if bidder i were not in the system.
- The revenue generated by Google from all other bidders when bidder i is in the system.

In VCG auctions, there are three main steps, as summarized in Figure 2.3.

1. The first step is to invite each bidder to send in her bids, one for each of the

items. In Google's case, they are proportional to a common scalar. Let v_{ij} be the value of the bid submitted to the seller by bidder i for item j. Now, these bids might not be the true valuations. But as we will soon discover, by properly matching the bidders with the items and then charging the right prices, we can induce the bidders to submit the true valuations as $\{v_{ij}\}$. Again, auction design can be viewed as a mechanism design problem.

2. Second, an optimization is carried out by the seller. A **matching** is computed through this optimization, in which each item is matched to a bidder.

   For example, if we draw three horizontal lines from bidders to items in Figure 2.2, that would be a matching, denoted as $[(1,1),(2,2),(3,3)]$. This matching returns a total valuation of $50+15+1=66$ to the bidders.

   In VCG, we want the matching to maximize the total valuation of the *whole system*: we maximize $\sum_{(ij)} v_{ij}$ (where the sum is over all the matched pairs) over all the possible matchings. For example, a different matching of $[(1,2),(2,3),(3,1)]$ would return a total valuation of $\sum_{(ij)} v_{ij} = 30+5+5 = 40$, an inferior matching to $[(1,1),(2,2),(3,3)]$, as illustrated in Figure 2.4.

   Denote by V the resulting maximized total value. Suppose item j is allocated to bidder i in this matching. Obviously, we can write V as the sum of the value v_{ij} and $\hat{V}_{i\leftarrow j}$, the sum of the values of all the other (item, bidder) pairs in the matching:

   $$V = v_{ij} + \hat{V}_{i\leftarrow j}.$$

   This notation will become useful soon.

3. Matching decides the allocation, and pricing is determined only after the matching. For a given matching, each pair of (item, bidder) is charged a price of p_{ij}, which is the amount of damage caused by this matching. So, how do we quantify the damage done by a bidder i getting item j?

   Imagine two alternative systems. (a) A system *without* bidder i, and there is a corresponding $V_{\text{no i}}$ as the maximum total valuation for everyone else. (b) A system *with* bidder i, who will get item j, and the corresponding $\hat{V}_{i\leftarrow j}$ as the total maximum valuation for everyone else. The difference between these two valuations: $V_{\text{no i}} - V_{i\leftarrow j}$ is the "damage" caused to everyone else, and that is the price p_{ij} the seller charges user i for being matched to item j:

   $$p_{ij} = V_{\text{no i}} - \hat{V}_{i\leftarrow j}. \tag{2.1}$$

   This completes the description of VCG auction.

Similar to the GSP example we just saw in the last section, suppose there are two ad spaces on a webpage and three bidders. An ad in the first space receives 400 clicks per hour, while the second space gets 200. Bidders 1, 2, and 3 have valuations per click of \$12, \$8, and \$4, respectively. Suppose the bids are the true valuations for now, we will later prove taht is indeed a dominant strategy in the auction game.

The matching by VCG happens to be the same as by GSP in this case: bidder

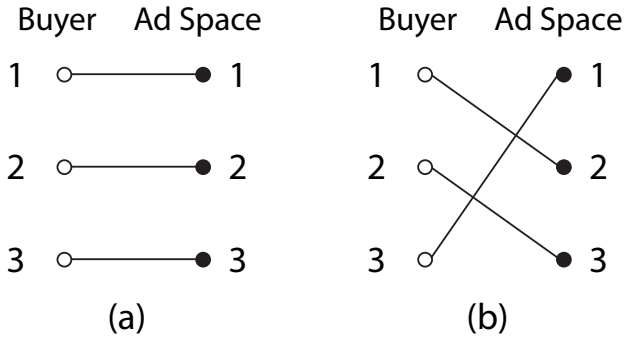


Figure 2.4 Two of the many possible matchings between bidders (on the left) and ad spaces (on the right) of a multi-item auction's bipartite graph. The weight of link (ij) is $v_{ij} = R_i C_j$. A maximum weight matching selects a subset of those links so that each bidder is matched to an item, and the sum of these links' weights is the largest possible among all matchings. By the parameter values in Figure **??**, matching (a) turns out to maximize $\sum_{(ij)} v_{ij}$, and is thus chosen in step 2 of a VCG auction, while matching (b) does not.

1 gets the first ad space, and bidder 2 the second, with bidder 3 not matched to any ad space. The second bidders payment is still \$800 because the damage it causes to the system is that bidder 3 lost the second ad space, thus \$4 $\times$ 200 = \$800 value. However, the payment of bidder 1 is now \$2400: \$800 for the damage to bidder 3 and \$1600 for the damage to bidder 2: bidder 2 moves from position 1 to position 2, thus causing her (400-200) = 200 clicks per day at the valuation of \$8 a click.

In this example, the seller's revenue under VCG is \$2400+\$800=\$3200, lower than that under GSP (\$3200 + \$800 = \$4000), if the advertisers bid truthfully in both cases.

2.4.2 Truthful bidding

So back to this question: what would each (rational) bidder i bid for item j in a VCG auction? We claim it must be the true valuation, *i.e.*, the expected revenue per click times the estimated click through rate.

Suppose that was not true, and bidder i bids some other number. In the VCG auction's matching, again it is important to realize that this would make a difference only if it resulted in bidder i getting a different item, say item h, *e.g.*, as in Figure 2.4(b). Obviously,

$$v_{ij} + \hat{V}_{i \leftarrow j} \geq v_{ih} + \hat{V}_{i \leftarrow h},$$

since the left side is V, the *maximized* total valuation over *all* possible matchings. Subtracting $V_{\text{no i}}$ from both sides:

$$v_{ij} + \hat{V}_{i \leftarrow j} - V_{\text{no i}} \geq v_{ih} + \hat{V}_{i \leftarrow h} - V_{\text{no i}},$$

and using the definition of p_{ij} in (2.1), we have

$$v_{ij} - p_{ij} \geq v_{ih} - p_{ih},$$

i.e., the payoff of bidding true valuation is higher. This argument shows that VCG induces truthful bidding as a dominant strategy for each bidder.

2.4.3 Other considerations

What about the seller's perspective: to maximize (over all the possible matchings) $\sum_{(ij)} p_{ij} C(j)$, the sum of per-click price times the estimated click through rate across the (item, bidder) pairs matched in a given matching. As we just saw in the example, GSP may generate more revenue than VCG, or less, depending on the problem parameter values. If the actual click through rate, as a function of the ad space location j, also depends on who is the winning bidder i, i.e., $C(j)$ becomes $C_i(j)$ and depends on i, then the seller's revenue maximization problem becomes even more challenging.

Which one to use: GSP or VCG? Google uses GSP while many web 2.0 companies use VCG. Companies like AppNexus run variants of VCG auctions on a large-scale (billions each day), real time (millisecond matching), and user-profile-based targeted ad placement. The comparison needs to include various considerations:

- GSP is simpler for Google to explain to advertisers than VCG.
- While GSP is simple when $C(j)$ is independent of winner i, it becomes more complicated when the click through rate of an ad space depends on the actual ad placed there.
- While VCG guarantees truthful bidding, that may not be the case if there are multiple auctions held in parallel across many websites.
- If Google switches from GSP to VCG, the advertisers may act irrationally. They may ignore this change, continue to shade their bids, and use the same bids as before. You can readily check that, for the same set of bids, VCG revenue is lower than GSP revenue. Relative to the potential benefit, the cost of transitioning from GSP to VCG has been too high for Google.

What is clear from this chapter is that different transaction mechanisms can induce very different behaviors from people. People react to different prices with different demands, react to different allocation methods with different strategies, and react to different expectations of tomorrow's events with different actions today. More generally, our design of a network and its functions may induce different optimization problems for the operators or the individual players. This is a recurring theme throughout the book.

Further Reading

Auction theory is a well studied discipline in economics and in computer science. There are many variations of the basic ones we covered here: reverse auction with one bidder and many sellers, double auctions with many bidders and many sellers, multiple winners per auction, revenue maximization strategies, and collusion among bidders.

1. The following is a standard textbook on the subject:
[Kri09] V. Krishna, *Auction Theory*, 2nd Ed., Academic Press 2009.

2. Another nice survey with special emphasis on applications is
[Mil04] P. Milgrom, *Putting Auction Theory to Work*, Cambridge University Press 2004.

3. The following book provides an in-depth discussion of eBay auction and the surprises in people's behavior there:
[Ste07] K. Steiglitz, *Snipers, Shills, and Sharks*, Princeton University Press, 2007.

4. The following short paper provides a concise survey to the key ideas in Google ad search:
[Var07] H. Varian, "The economics of Internet search," *The Angela Costa Lecture*, 2007.

5. A more technical discussion focusing on GSP can be found here, especially the viewpoint of auctions as a dynamic game:
[EOS07] B. Edelman, M. Ostrovsky, and M. Schwarz, "Internet Advertising and the Generalized Second-Price Auction: Selling Billions of Dollars Worth of Keywords," *The American Economic Review*, vol. 97, no. 1, pp. 242-259, 2007.

Problems

2.1 *A simple ad space auction* ⋆

Three advertisers, 1, 2, 3, bid for two ad spaces A, B. The average revenue per click are \$6, \$4, \$3 for the bidders respectively, and the click through rate of the ad spaces are 500, 300 clicks per hour respectively.

(a) Draw the bipartite graph with nodes indicating advertisers/ad spaces and edges indicating values per hour. Indicate the maximum matching with bold lines.

(b) What is the result of the auction, assuming truthful bidding: allocation,

prices charged, and payoffs received?

2.2 *eBay Auction* ⋆⋆

Kevin, the seller, lists a sofa for sale on eBay via auction with both start price and reserve price set to $7.00 and a duration of 5 days. The minimal increment is $0.25 and the following events happen during the auction:

- *Day 1* Bidder 1 uses a proxy agent setting the maximum bid up to $11.00
- *Day 2* Bidder 2 bids $9.25.
- *Day 3* Bidder 3 uses a proxy agent setting the maximum bid up to $17.25
- *Day 4* Bidder 2 bids $13.65.
- *Day 5* Bidder 1 bids $27.45.

List the bid history of the three bidders. Who is the winner and what price does she pay?

2.3 *More items than bidders* ⋆

Tom and Jimmy are bidding for three ad slots on a page, and one bidder can win at most one slot. Suppose an ad in the first slot receives 500 clicks per hour, while the second and third slot get 300, 200 clicks per hour respectively. Assume Tom receive r value per click.

(a) Denote by b, b' as the bids by Tom and Jimmy respectively. In GSP auction, discuss Tom's payoff f in terms of b, b'.

(b) Does Tom have a dominant strategy?

2.4 *Reverse auction* ⋆

Reverse auction is a type of auction where there are multiple sellers and only one bidder. The roles of bidders and sellers are reversed , that is, sellers lower their bids during auction and the one with the lowest bid sells his/her item.

Suppose there are three sellers in a reverse auction with one bidder. Denote b_i as the price seller i bids, v_i as the value seller i attaches to the item.

(a) In the case of second price auction, what is the payoff function f_i for seller i, as a function of b_1, b_2, b_3?

(b) Is truthful bidding a dominant strategy?

2.5 *Spectrum auction and package bidding* ⋆⋆

Wireless cellular technologies rely on spectrum assets. Around the world, auctions have emerged as the primary means of assigning spectrum licenses to companies wishing to provide wireless communication services. For example, from July 1994 to July 2011, the U.S. Federal Communications Commission (FCC) conducted 92 spectrum auctions, raising over $60 billion for the U.S. Treasury, and assigned thousands of licenses to hundreds of firms in different parts of the spectrum and different geographic regions of the country.

The U.S. FCC spectrum auctions use *simultaneous ascending auction*, in which groups of related licenses are auctioned simultaneously and the winner pays the highest bid. The British OfCom, in contrast, runs **package bidding**, where each potential spectrum bidder can bid on a joint set of frequency bands.

Among the many issues involved in spectrum auctioning is the debate between simultaneous ascending auction and package bidding auction. We will illustrate the inefficiency resulting from disallowing package bidding in a toy example. The root cause is bidder-specific complementarity and lack of competition.

Suppose that there are two bidders for two adjacent seats in a movie theater. Bidder 1 is planning to watch the movie together with her spouse as part of a date. She values the two spots *jointly* at $15, and a single spot is worth nothing. Bidder 2 plans to watch the movie by himself, and value each seat at $10, and the two seats together at $12 (since it is a little nicer to have no one sitting next to him on one side of his seat).

(a) Assume a simultaneous ascending auction is used for the seats, and Bidder 1 correctly guesses that Bidder 2 values $10 for one seat and $12 for two seats together. What strategy will Bidder 1 take? What is the result of the auction, in terms of the winner, the allocation, the price charged, and payoffs received?

(b) Repeat part (a) but now assume package bidding is used. In particular, Bidder 1 can bid on a package consisting of both seats. Explain the differences with (a).

3 How does Google rank webpages?

3.1 A Short Answer

Now we turn to the other links you see on a search result webpage; not the ads or sponsored search results, but the actual ranking of webpages by search engines such as Google. We will see that, each time you search on `www.google.com`, Google solves a very big linear equation to rank the webpages.

The idea of embedding links in text dates back to the middle of last century. As the Internet scaled up, and with the introduction of the web in 1989, the browser in 1990, and the web portal in 1994, this vision was realized with an unprecedented scale. The network of webpages is huge: over 1 trillion by 2008 in one estimate. And most of them are connected to each other in a giant component of this network. It is also sparse: most webpages only have a few hyperlinks pointing in or out. Google search organizes this huge and sparse network by ranking the webpages.

More important webpages should be ranked higher. But how do you quantify *how* important a webpage is? Well, if there are many other important webpages pointing towards a webpage A, probably A is important. This argument implicitly assumes two ideas:

- Webpages form a network, where a webpage is a node, and a hyperlink is a *directed* link in the network: webpage A may point to B without B pointing back to A.
- We can turn the seemingly circular logic of "important webpages pointing to you means you are important" into a set of equations that characterize the *equilibrium* based on a *recursive definition* of "importance." This importance score will act as an approximation of the ultimate test of search engines: how useful a user finds the search results.

Suppose there are N webpages. Each webpage i has O_i **outgoing links** and I_i **incoming links**. We cannot just count the number of webpages pointing to a given webpage A, because that number, the **in-degree** of the node in the hyperlinked graph, is often not the right measure of importance.

Let us denote the "importance score" of each webpage by π_i. If important webpages point to webpage A, maybe webpage A should be important too, *i.e.*, $\pi_A = \sum_{i \to A} \pi_i$, where the sum is over all the webpages pointing to A. But this

is not quite right either, since node i may be pointing to many other nodes in this graph, and that means each of these nodes only receives a small portion of node i's importance score.

Let us assume each node's importance score is *evenly* spread across all the outgoing links, *i.e.*, each of the outgoing neighbors of node i receives π_i/O_i importance score. Now each node's importance score can also be written as the sum of the importance scores received *from* all of the incoming neighbors, indexed by j, *e.g.*, for node A,

$$\sum_{j \to A} \frac{\pi_j}{O_j}.$$

If this sum is indeed also π_A, we have *consistency* of the scores. But it is not clear if we can readily compute these scores, or if there is a consistent set of scores at all.

It turns out that, with a couple of modifications to the basic idea above, there is always a unique set of consistent scores, denoted as $\{\pi_i^*\}$, and these scores determine the ranking of the webpages: the higher the score, the higher the webpage ranked.

For example, consider a very small graph with just four webpages and six hyperlinks, shown in Figure 3.1. This is a directed graph where each node is a webpage and each link a hyperlink. A consistent set of importance scores turns out to be [0.125, 0.125, 0.375, 0.375]: webpages 3 and 4 are more important than webpages 1 and 2. In this small example, it so happens that webpages 3 and 4, linking each other, push both webpages' rankings higher.

Intuitively, the scores make sense. First, by symmetry of the graph, webpages 1 and 2 should have the same importance score. We can view webpages 3 and 4 as if they form one webpage first, a supernode 3+4. Since node 3+4 has two incoming links, and each of nodes 1 and 2 only one incoming link, node 3+4 should have a higher importance score. Since node 3 points to node 4 and vice versa, these two nodes' importance scores mix into an equal division at equilibrium. This line of reasoning qualitatively explains the actual scores we see.

But how do we calculate the exact scores? In this small example, it boils down to two simple linear equations. Let the score of node 1 (and 2) be x, and that of node 3 (and 4) be y. Looking at node 1's incoming links, we see that there is only one such link, coming from node 4 that points to three nodes. So we know $x = y/3$. Since all scores must add up to: $2x + 2y = 1$, we have $x = 0.125$ and $y = 0.375$.

Now, how do we compute this set of consistent scores in a large, sparse, general graph of hyperlink connectivity?

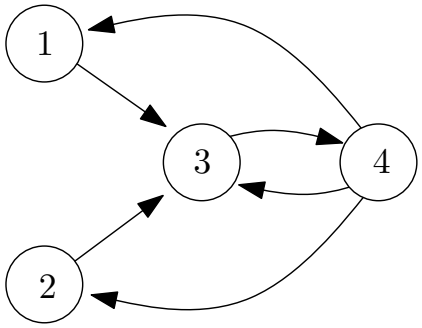


Figure 3.1 A simple example of importance score with 4 webpages and 6 hyperlinks. It is small with much symmetry, leading to a simple calculation of importance scores of the nodes.

3.2 A Long Answer

In any search engine, there are two main activities going on continuously behind the scene: (a) crawling the hyperlinked web space to get the webpage information, (b) indexing this information into concise representations and storing the indices.

When you search in Google, it triggers a ranking procedure that takes into account two main factors:

- How relevant the content is on each webpage, or the **relevance score**.
- How important the webpage is, or the **importance score**.

It is the composite score of these two factors that determines the ranking. We focus on the importance score ranking, since that usually determines the order of the top few webpages in any reasonably popular search, which has a tremendous impact on how people obtain information and how online businesses generate traffic.

We will be constructing several related matrices: $\mathbf{H}, \hat{\mathbf{H}}$, and $\mathbf{G}$, step by step (this matrix $\mathbf{G}$ is not the channel gain matrix of Chapter 1; it denotes the Google matrix in this chapter). Eventually, we will be computing an eigenvector of $\mathbf{G}$ as the importance score vector. Each matrix is $N \times N$, where N is the number of webpages. These are extremely large matrices, and we will discuss the computational challenge of scaling-up in Advanced Material.

3.2.1 Constructing $\mathbf{H}$

The first matrix we define is $\mathbf{H}$: its (i, j)th entry is $1/O_i$ if there is a hyperlink from webpage i to webpage j, and 0 otherwise. This matrix describes the network topology: which webpages points to which. It also evenly spreads the importance of each webpage among its outgoing neighbors, or the webpages that it points to.

Let π be an $N \times 1$ column vector denoting the importance scores of the N webpages. We start by guessing that the consistent score vector is $\mathbf{1}$, simply a vector of 1s as each webpage is equally important. So we have an initial vector $\pi[0] = \mathbf{1}$, where 0 denotes the 0th iteration, *i.e.*, the initial condition.

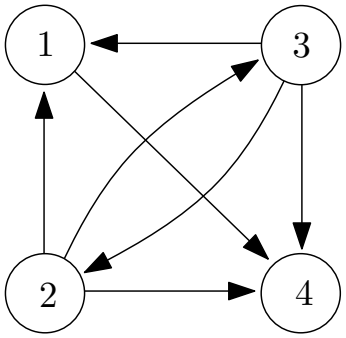


Figure 3.2 A network of hyperlinked webpages with a dangling node 4.

Then, multiply π^T on the right by matrix $\mathbf{H}$. (By convention, a vector is a column vector. So when we write a vector horizontally on a line, we put the transpose symbol T on top of the vector.) You can write out this matrix multiplication, and see this is spreading the importance score from the last iteration evenly among the outgoing links, and re-calculating the importance score of each webpage in this iteration by summing up the importance scores from the incoming links. For example, $\pi_1[2]$ (for webpage 1 in the second iteration) can be expressed as the following weighted sum of importance scores from the first iteration:

$$\pi_1[2] = \sum_{j \to 1} \frac{\pi_j[1]}{O_j},$$

i.e., the π vector from the previous iteration inner-producting the first column of $\mathbf{H}$:

$$\pi_1[2] = (\pi[1])^T (\text{column 1 of } \mathbf{H}).$$

If we index the iterations by k, the update at each iteration is simply:

$$\pi^T[k] = \pi^T[k-1]\mathbf{H}. \tag{3.1}$$

We followed the (visually a little clumsy) convention in this research field that defined $\mathbf{H}$ such that the update is a multiplication of row vector π^T by $\mathbf{H}$ from the right.

Since the absolute values of the entries in π do not matter, only the ranked order, we can also normalize the resulting π vector so that its entries add up to 1.

Do the iterations in (3.1) converge, *i.e.*, is there a K sufficiently large such that, for all $k \geq K$, the $\pi[k]$ vector is arbitrarily close to $\pi[k-1]$ (no matter the initial guess $\pi[0]$)? If so, we have a way to compute a consistent score vector as accurately as we want.

But the answer is "not quite yet." We need two adjustments to $\mathbf{H}$.

3.2.2 Constructing $\hat{\mathbf{H}}$

First, some webpages do not point to any other webpages. These are "dangling nodes" in the hyperlink graph. For example, in Figure 3.2, node 4 is a dangling node, and its row is all 0s in the $\mathbf{H}$ matrix:

$$\mathbf{H} = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 1/3 & 0 & 1/3 & 1/3 \\ 1/3 & 1/3 & 0 & 1/3 \\ 0 & 0 & 0 & 0 \end{bmatrix}.$$

There are no consistent scores. To see this, we write out the system of linear equations $\pi = \pi\mathbf{H}$:

$$\begin{cases} \frac{1}{3}(\pi_2 + \pi_3) = \pi_1 \\ \frac{1}{3}\pi_3 = \pi_2 \\ \frac{1}{3}\pi_2 = \pi_3 \\ \pi_1 + \frac{1}{3}(\pi_2 + \pi_3) = \pi_4. \end{cases}$$

Solving these equations gives $\pi_1 = \pi_2 = \pi_3 = \pi_4 = 0$, which violates the normalization requirement $\sum_i \pi_i = 1$.

One solution is to replace each row of 0's, like the last row in $\mathbf{H}$ above, with a row of $1/N$. Intuitively, this is saying that even if a webpage does not point to any other webpage, we will force it to spread its importance score evenly among all the webpages out there.

Mathematically, this amounts to adding the matrix $\frac{1}{N}(\mathbf{w}\mathbf{1}^T)$ to $\mathbf{H}$, where $\mathbf{1}$ is simply a vector of 1s, and $\mathbf{w}$ is an indicator vector with the ith entry being 1 if webpage i points to no other webpages (a dangling node) and 0 otherwise (not a dangling node). This is an *outer product* between two N-dimensional vectors, which leads to an $N \times N$ matrix. For example, if $N = 2$ and $\mathbf{w} = [1\ 0]^T$, we have

$$\frac{1}{2}\begin{pmatrix} 1 \\ 0 \end{pmatrix}(1\ 1) = \begin{pmatrix} 1/2 & 1/2 \\ 0 & 0 \end{pmatrix}.$$

This new matrix we add to $\mathbf{H}$ is clearly simple. Even though it is big, $N \times N$, it is actually the same vector $\mathbf{w}$ repeated N times. We call it a **rank-1 matrix**.

The resulting matrix:

$$\hat{\mathbf{H}} = \mathbf{H} + \frac{1}{N}(\mathbf{w}\mathbf{1}^T),$$

has all entries non-negative and each row adds up to 1. So we can think of each row as a probability vector, with the (i, j)th entry of $\hat{\mathbf{H}}$ indicating the probability that, if you are currently on webpage i, you will click on a link and go to webpage j.

Well, the structure of the matrix says that you are equally likely to click on any links shown on a webpage, and if there is no link at all, you will be equally likely to visit any other webpages. Such behavior is called a **random walk on graphs** and can be studied as **Markov chains** in probability theory. Clearly

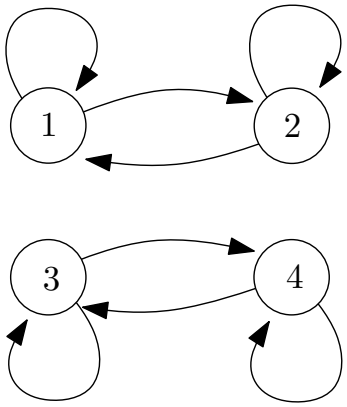


Figure 3.3 A network of hyperlinked webpages with multiple consistent score vectors.

this does not model web browsing behavior exactly, but it does strike a pretty effective balance between *simplicity* of the model and *usefulness* of the resulting webpage ranking. We will see a similar model for influence in social networks in Chapter 8.

3.2.3 Constructing $\mathbf{G}$

Second, there might be many consistent score vectors, all compatible with a given $\hat{\mathbf{H}}$. For example, for the graph in Figure 3.3, we have

$$\mathbf{H} = \begin{bmatrix} 1/2 & 1/2 & 0 & 0 \\ 1/2 & 1/2 & 0 & 0 \\ 0 & 0 & 1/2 & 1/2 \\ 0 & 0 & 1/2 & 1/2 \end{bmatrix}.$$

Different choices of $\pi[0]$ result in different π^*, which are all consistent. For example, if $\pi[0] = [1\ 0\ 0\ 0]^T$, then $\pi^* = [0.5\ 0.5\ 0\ 0]^T$. If $\pi[0] = [0\ 0.3\ 0.7\ 0]^T$, then $\pi^* = [0.15\ 0.15\ 0.35\ 0.35]^T$.

One solution to this problem is to add a little *randomization* to the iterative procedure and the recursive definition of importance. Intuitively, we say there is a chance of $(1-\theta)$ that you will jump to some other random webpage, without clicking on any of the links on the current webpage.

Mathematically, we add yet another matrix $\frac{1}{N}\mathbf{1}\mathbf{1}^T$, a matrix of 1s scaled by $1/N$ (clearly a rank-1 matrix), to $\hat{\mathbf{H}}$. But this time it is a *weighted* sum, with a weight $\theta \in [0,1]$. $(1-\theta)$ describes how likely you will randomly jump to some other webpage. The resulting matrix is called the **Google matrix**:

$$\mathbf{G} = \theta\hat{\mathbf{H}} + (1-\theta)\frac{1}{N}\mathbf{1}\mathbf{1}^T. \tag{3.2}$$

Now we can show that, independent of the initialization vector $\pi[0]$, the iterative procedure below:

$$\pi^T[k] = \pi^T[k-1]\mathbf{G} \tag{3.3}$$

will converge as $k \to \infty$, and converge to the unique vector π^* representing the consistent set of importance scores. Obviously, π^* is the left eigenvector of $\mathbf{G}$ corresponding to the eigenvalue of 1:

$$\pi^{*T} = \pi^{*T}\mathbf{G}. \tag{3.4}$$

One can then normalize π^*: take $\pi_i^*/\sum_j \pi_j^*$ as the new value of π_i^*, and rank the entries in descending order, before outputting them on the search result webpage in that order. The matrix $\mathbf{G}$ is designed such that there is a solution to (3.4) and that (3.3) converges from any initialization.

However you compute π^*, taking (the normalized and ordered version of) π^* as the basis of ranking is called the **pagerank algorithm**. Compared to DPC for wireless networks in Chapter 1, the matrix $\mathbf{G}$ in pagerank is much larger, but we can afford a centralized computation.

3.3 Examples

Consider the network in Figure 3.4 with 8 nodes and 16 directional links. We have

$$\mathbf{H} = \begin{bmatrix} 0 & 1/2 & 1/2 & 0 & 0 & 0 & 0 & 0 \\ 1/2 & 0 & 0 & 0 & 1/2 & 0 & 0 & 0 \\ 0 & 1/2 & 0 & 0 & 0 & 0 & 0 & 1/2 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1/2 & 0 & 0 & 0 & 1/2 \\ 0 & 0 & 0 & 1/2 & 1/2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1/2 & 0 & 1/2 & 0 & 0 \\ 1/3 & 0 & 0 & 1/3 & 0 & 0 & 1/3 & 0 \end{bmatrix}.$$

Here $\hat{\mathbf{H}} = \mathbf{H}$ since there is no dangling node. Taking $\theta = 0.85$, we have

$$\mathbf{G} = \begin{bmatrix} 0.0188 & 0.4437 & 0.4437 & 0.0188 & 0.0188 & 0.0188 & 0.0188 & 0.0188 \\ 0.4437 & 0.0188 & 0.0188 & 0.0188 & 0.4437 & 0.0188 & 0.0188 & 0.0188 \\ 0.0188 & 0.4437 & 0.0188 & 0.0188 & 0.0188 & 0.0188 & 0.0188 & 0.4437 \\ 0.0188 & 0.0188 & 0.8688 & 0.0188 & 0.0188 & 0.0188 & 0.0188 & 0.0188 \\ 0.0188 & 0.0188 & 0.0188 & 0.4437 & 0.0188 & 0.0188 & 0.0188 & 0.4437 \\ 0.0188 & 0.0188 & 0.0188 & 0.4437 & 0.4437 & 0.0188 & 0.0188 & 0.0188 \\ 0.0188 & 0.0188 & 0.0188 & 0.4437 & 0.0188 & 0.4437 & 0.0188 & 0.0188 \\ 0.3021 & 0.0188 & 0.0188 & 0.3021 & 0.0188 & 0.0188 & 0.3021 & 0.0188 \end{bmatrix}.$$

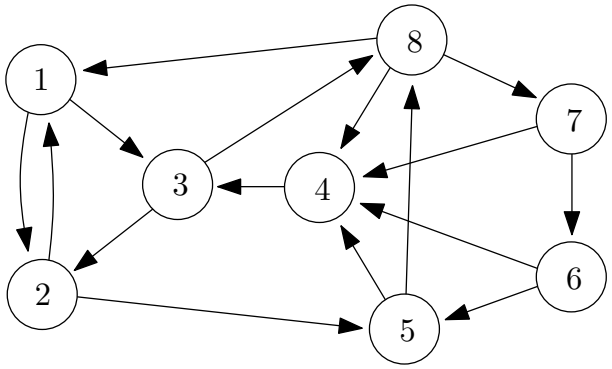


Figure 3.4 An example of the pagerank algorithm with 8 webpages and 16 hyperlinks. Webpage 3 is ranked the highest even though webpage 4 has the largest in-degree.

Initializing $\pi[0] = [1/8\ 1/8\ \cdots 1/8]^T$, iteration (3.3) gives

$$\begin{aligned}
\pi[1] &= [0.1073\ 0.1250\ 0.1781\ 0.2135\ 0.1250\ 0.0719\ 0.0542\ 0.1250]^T \\
\pi[2] &= [0.1073\ 0.1401\ 0.2459\ 0.1609\ 0.1024\ 0.0418\ 0.0542\ 0.1476]^T \\
\pi[3] &= [0.1201\ 0.1688\ 0.2011\ 0.1449\ 0.0960\ 0.0418\ 0.0606\ 0.1668]^T \\
\pi[4] &= [0.1378\ 0.1552\ 0.1929\ 0.1503\ 0.1083\ 0.0445\ 0.0660\ 0.1450]^T \\
\pi[5] &= [0.1258\ 0.1593\ 0.2051\ 0.1528\ 0.1036\ 0.0468\ 0.0598\ 0.1468]^T \\
\pi[6] &= [0.1280\ 0.1594\ 0.2021\ 0.1497\ 0.1063\ 0.0442\ 0.0603\ 0.1499]^T \\
&\vdots
\end{aligned}$$

and, to 4 decimal places, $\pi^* = [0.1286\ 0.1590\ 0.2015\ 0.1507\ 0.1053\ 0.0447\ 0.0610\ 0.1492]^T$. This means the ranked order of the webpages are: 3, 2, 4, 8, 1, 5, 7, 6.

The node with the largest in-degree, *i.e.*, with the largest number of links pointing to a node, is node 4, which is *not* ranked the highest. This is in part because its importance score is spread exclusively to node 3. As we will see again in Chapter 8, there are many more useful metrics measuring node importance than just the degree, pagerank being one of them.

3.4 Advanced Material

3.4.1 Generalized pagerank and some basic properties

The Google matrix $\mathbf{G}$ can be generalized if the randomization ingredient is more refined. First, instead of the matrix $\frac{1}{N}\mathbf{1}\mathbf{1}^T$, we can add the matrix $\mathbf{1}\mathbf{v}^T$ (again, the outer product of two vectors), where $\mathbf{v}$ can be *any* probability distribution. Certainly, $\frac{1}{N}\mathbf{1}^T$ is a special case of that.

We can also generalize the dangling node treatment: instead of adding $\frac{1}{N}\mathbf{w}\mathbf{1}^T$

to $\mathbf{H}$, where $\mathbf{w}$ is the indicator vector of dangling nodes, we can add $\mathbf{w}\mathbf{v}^T$. Again, using $\frac{1}{N}\mathbf{1}$ is a special case.

Now, the Google update equation can be written in the long form (not using the shorthand notation $\mathbf{G}$) as a function of the given webpage connectivity matrix $\mathbf{H}$, vector $\mathbf{w}$ indicating the dangling webpages, and the two algorithmic parameters: scalar θ and vector $\mathbf{v}$:

$$\pi^T\mathbf{G} = \theta\pi^T\mathbf{H} + \pi^T(\theta\mathbf{w} + (1-\theta)\mathbf{1})\mathbf{v}^T. \tag{3.5}$$

You should verify that the above equation is indeed the same as (3.3).

There are many viewpoints to further interpret (3.3) and connect it to matrix theory, to Markov chain theory, and to linear systems theory. For example:

- π^* is the left eigenvector corresponding to the dominant eigenvalue of a positive matrix.
- It represents the so-called stationary distribution of a Markov chain whose transition probabilities are in $\mathbf{G}$.
- It represents the equilibrium of an economic growth model according to $\mathbf{G}$ (more on this viewpoint later in this section).

The major operational challenges of running the seemingly simple update (3.3) are *scale* and *speed*: there are billions of webpages and Google needs to return the results almost instantly.

Still, the power method (3.3) offers many numerical advantages compared to a direct computation of the dominant eigenvector of $\mathbf{G}$. First, (3.3) can be carried out by multiplying a vector by the sum of $\mathbf{H}$ and two rank-1 matrices. This is numerically simple: $\mathbf{H}$ is very large but also very *sparse*: each webpage usually links to just a few other webpages, so almost all the entries in $\mathbf{H}$ are zero. Multiplying by rank-1 matrices is also easy. Furthermore, at each iteration, we only need to store the current π vector.

While we have not discussed the speed of convergence, it is clearly important to speed up the computation of π^*. As we will see again in Chapter 8, the convergence speed in this case is governed by the second largest eigenvalue $\lambda_2(\mathbf{G})$ of $\mathbf{G}$, which can be shown to be approximately θ here. So this parameter θ controls the tradeoff between convergence speed and the relevance of the hyperlink graph in computing the importance scores: smaller θ (closer to 0) drives the convergence faster, but also de-emphasizes the relevance of the hyperlink graph structure. This is hardly surprising: if you view the webpage importance scores more like random objects, it is easier to compute the equilibrium. Usually $\theta = 0.85$ is believed to be a pretty good choice. This leads to convergence in about tens of iterations while still giving most of the weight to the actual hyperlink graph structure rather than the randomization component in $\mathbf{G}$.

3.4.2 Pagerank as a solution to a linear equation

Pagerank sounds similar to the distributed power control in Chapter 1. They both apply the power method to solve a system of linear equations. The solutions to those equations capture the right engineering configuration in the network, whether that is the relative importance of webpages in a hyperlink graph or the best transmit power vector in a wireless interference environment. This conceptual connection can be sharpened to an exact, formal parallel below.

First, we can rewrite the characterization of π^* as the solution to the following linear equation (rather than as the dominant left eigenvector of matrix $\mathbf{G}$ (3.4), the viewpoint we have been taking so far):

$$(\mathbf{I} - \theta\mathbf{H})^T\pi = \mathbf{v}. \tag{3.6}$$

Compare (3.6) with the characterization of optimal power vector in the distributed power control algorithm in Chapter 1:

$$(\mathbf{I} - \mathbf{DF})\mathbf{p} = \mathbf{v}.$$

Of course, the vectors $\mathbf{v}$ are defined differently in these two cases: based on webpage viewing behavior in pagerank and on receiver noise in power control. But we see a striking parallel: the consistent score vector π and the optimal transmit power vector $\mathbf{p}$ are both solutions to a linear equation with the following structure: identity matrix minus a scaled version of the network connectivity matrix.

In pagerank, the network connectivity is represented by the hyperlink matrix $\mathbf{H}$. This makes sense since the key factor here is the hyperlink connectivity pattern among the webpages. In power control, the network connectivity is represented by the normalized channel gain matrix $\mathbf{F}$. This makes sense since the key factor here is the strength of the interference channels.

In pagerank, the scaling is done by one scalar θ. In power control, the scaling is done by many scalars in the diagonal matrix $\mathbf{D}$: the target SIR for each user. To make the parallelism exact, we may think of a generalization of the Google matrix $\mathbf{G}$ where each webpage has its own scaling factor θ.

The general theme for solving these two linear equations can be stated as follows. Suppose you want to solve a system of linear equations $\mathbf{Ax} = \mathbf{b}$, but do not want to directly invert the square matrix $\mathbf{A}$. You might be able to split $\mathbf{A} = \mathbf{M} - \mathbf{N}$, where $\mathbf{M}$ is invertible and its inverse $\mathbf{M}^{-1}$ can be much more easily computed than $\mathbf{A}^{-1}$.

The following **linear stationary iteration** ("linear" because the operations are all linear, and "stationary" because the matrices themselves do not vary over iterations) over times indexed by k:

$$\mathbf{x}[k] = \mathbf{M}^{-1}\mathbf{N}\mathbf{x}[k-1] + \mathbf{M}^{-1}\mathbf{b}$$

will converge to the desired solution:

$$\lim_{k\to\infty} \mathbf{x}[k] = \mathbf{A}^{-1}\mathbf{b},$$

from any initialization $\mathbf{x}[0]$, provided that the largest eigenvalue of $\mathbf{M^{-1}N}$ is smaller than 1. Both DPC and pagerank are special cases of this general algorithm.

But we still need to show that (3.6) is indeed equivalent to (3.4): a π that solves (3.6) also solves (3.4), and vice versa. First, starting with a π that solves (3.6), we can easily show the following string of equalities:

$$\begin{aligned}\mathbf{1}^T\mathbf{v} &= \mathbf{1}^T(\mathbf{I}-\theta\mathbf{H})^T\pi \\ &= \mathbf{1}^T\pi - \theta(\mathbf{H}\mathbf{1})^T\pi \\ &= \mathbf{1}^T\pi - \theta(\mathbf{1}-\mathbf{w})^T\pi \\ &= \pi^T(\theta\mathbf{w}+(1-\theta)\mathbf{1}),\end{aligned}$$

where the first equality uses (3.6) and the third equality uses the fact that summing each row of $\mathbf{H}$ gives a vector of 1s (except those rows corresponding to dangling webpages). The other two equalities are based on simple algebraic manipulations.

But $\mathbf{1}^T\mathbf{v} = 1$ by design, so we know

$$\pi^T(\theta\mathbf{w}+(1-\theta)\mathbf{1}) = 1.$$

Now we can readily check that $\pi^T\mathbf{G}$, using its definition in (3.5) and the above equation, equals $\theta\pi^T\mathbf{H}+\mathbf{v}^T$.

Finally, using one more time the assumption that π satisfies (3.6), *i.e.*, $\mathbf{v} = (\mathbf{I}-\theta\mathbf{H})^T\pi$, we complete the argument:

$$\pi^T\mathbf{G} = \theta\pi^T\mathbf{H} + \pi^T(\mathbf{I}-\theta\mathbf{H}) = \theta\pi^T\mathbf{H} - \theta\pi^T\mathbf{H} + \pi^T = \pi^T.$$

Therefore, any π solving the linear equation (3.6) is also a dominant left eigenvector of $\mathbf{G}$ that solves (3.4).

Vice versa, a π that solves (3.4) also solves (3.6), can be similarly shown.

3.4.3 Scaling up and speeding up

It is not easy to adjust the parameters in pagerank computation. We discussed the role of θ before, and we know that when θ is close to 1, pagerank results become very sensitive to small changes in θ, since the importance matrix $(\mathbf{I}-\theta\mathbf{H})^{-1}$ approaches infinity.

There is also substantial research going into designing the right randomization vector $\mathbf{v}$, *e.g.*, the entries of the $\mathbf{H}$ matrix: a web surfer likely will not pick all of the hyperlinked webpages equally likely, and their actual behavior can be recorded to adjust the entries of $\mathbf{H}$.

But the biggest challenge to running pagerank is *scale*: how to scale up to really large matrices? How to quickly compute and update the rankings? There are both storage and computational challenges. And there are many interesting approaches developed over the years, including the following five. The first one is a computational acceleration method. The other four are *approximations*, two

change the notion of optimality and two restructure the graph of the hyperlinked webpages.

1. *Decompose* **H**. A standard triangular decomposition gives $\mathbf{H} = \mathbf{DL}$, where **D** is a diagonal matrix with entries being 1 over the number of links from webpage i, and **L** is a binary adjacency matrix. So only integers, instead of real numbers, need to be stored to describe **H**. Now, suppose there are N webpages, and on average, each webpage points to M webpages. N is huge: in the billions, and M is very small: often 10 or less. Instead of NM multiplications, this matrix decomposition reduces it to just M multiplications.
2. *Relax the meaning of convergence.* It is not the values of importance scores that matter to most people, it is just the *order* of the webpages, especially the top ones. So once the computation of pagerank is sure about the order, there is no need to further improve the accuracy of computing π towards convergence.
3. *Differentiate among the webpages.* The pagerank algorithm as applied to most webpages quickly converge, and can be locked while the other webpages' pageranks are refined. This is an approximation that works particularly well when the pageranks follow the power law that we will discuss in Chapter 10.
4. *Leave the dangling nodes out.* There are many dangling nodes out there, and their behavior in the matrices and computations involved are pretty similar. So they might be grouped together and not be updated to speed up the computation.
5. *Aggregation of webpages.* When many nodes are lumped together into a cluster, then *hierarchical* computation of pageranks can be recursively computed, first treating each cluster as one webpage, and then distributing the pagerank of that cluster among the actual webpages within that cluster. We will visit an example of this important principle of building hierarchy to reduce computation (or communication) load in a homework problem.

3.4.4 Beyond the basic search

There is another player in the game of search: companies that specialize in increasing a webpage's pagerank, possibly pushing it to the top few search results, or even to the very top spot. This industry is called **Search Engine Optimization** (SEO). There are many proprietary techniques used by SEO companies. Some techniques enhance content relevance scores, sometimes by adding bogus tags in the html files. Other techniques increase the importance score, sometimes by adding links pointing to a customer's site, and sometimes by creating several truly important webpages and then attaching many other webpages as its outgoing neighbors.

Google is also playing this game by detecting SEO techniques and then updating its ranking algorithm so that the artificial help from SEO techniques

is minimized. For example, in early 2011, Google made a major update to its ranking algorithm to counter the SEO effects.

There are also many important variants to the basic type of search we discussed, *e.g.*, personalized search based on user's feedback on how useful she finds the top webpages in the search result. Multimedia search is another challenging area: searching through images, audios, and video clips require very different ways of indexing, storing, and ranking the content than text-based search.

Further reading

The pagerank algorithm is covered in almost every single book on network science these days. Some particularly useful references are as follows.

1. The Google founders wrote the following seminal paper explaining the pagerank algorithm back in 1998:

[BP98] S. Brin and L. Page, "The anatomy of a large-scale hypertextual Web search engine," *Computer Networks and ISDN Systems* vol. 33, pp. 107-117, 1998.

2. The standard reference book devoted to pagerank is

[LM06] A. N. Langville and C. D. Meyer, *Google's Pagerank and Beyond*, Princeton University Press, 2006.

3. Here is well written website explaining pagerank:

[R] C. Ridings, "Pagerank explained: Everything you've always wanted to know about Pagerank," `http://www.rankwrite.com`.

4. Dealing with non-negative matrices and creating linear stationary iterations are well documented, *e.g.*, in the following reference book:

[BP79] A. Berman and R. J. Plemmons, *Nonnegative Matrices in the Mathematical Sciences*, Academic Press, 1979.

5. Computational issues in matrix multiplication is treated in textbooks like this one:

[GV96] G. Golub and C. F. van Van Loan, *Matrix Computations*, 3rd Ed., The Johns Hopkins University Press, 1996.

Problems

3.1 *Pagerank sink* ⋆

Write out the $\mathbf{H}$ matrix of the graph in Figure 3.5. Iterate $\pi[k]^T = \pi[k-1]^T\mathbf{H}$,

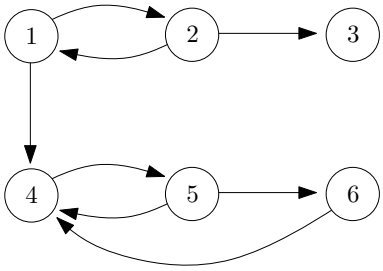


Figure 3.5 A simple network of webpages with a sink node.

where $k = 0, 1, 2, \ldots$, and let

$$\pi[0] = \begin{bmatrix} 1/6 & 1/6 & 1/6 & 1/6 & 1/6 & 1/6 \end{bmatrix}^T.$$

What problem do you observe with the converged $\pi[k]$ vector?

3.2 *Cyclic ranking* ⋆

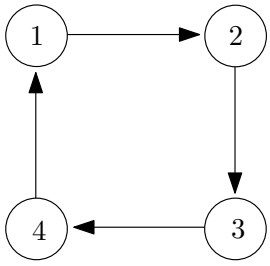


Figure 3.6 A simple network of webpages with a cycle.

Write out the $\mathbf{H}$ matrix of the graph in Figure 3.6. Iterate $\pi[k]^T = \pi[k-1]^T\mathbf{H}$, where $k = 0, 1, 2, \ldots$, and let

$$\pi[0] = \begin{bmatrix} 1/2 & 1/2 & 0 & 0 \end{bmatrix}^T.$$

(a) Do the vectors $\{\pi[k]\}$ converge?
(b) Solve for π^* such that $\pi^{*T} = \pi^{*T}\mathbf{H}$ and $\sum_i \pi_i^* = 1$.

3.3 *Basic pagerank with different* θ ⋆⋆

Compute the pagerank vector π^* of the graph in Figure 3.7, for $\theta = 0.1, 0.3, 0.5, 0.85$. What do you observe?

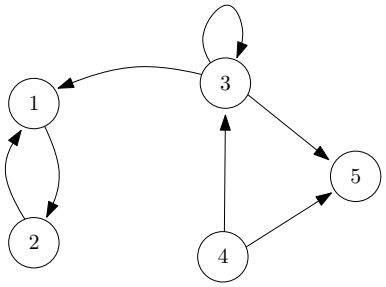


Figure 3.7 A simple example for pagerank with different θ.

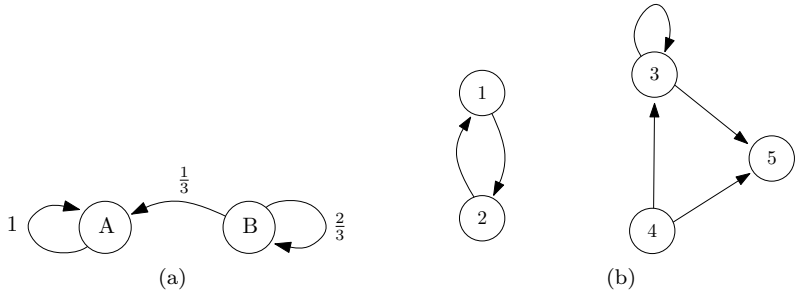


Figure 3.8 An example of hierarchical pagerank. Two different graphs that will be superimposed later.

3.4 *Block aggregation in pagerank* ⋆⋆

Set $\theta = 0.85$ and start with any normalized initial vector $\pi[0]$.

(a) Compute the pagerank vector $\begin{bmatrix}\pi^*_A & \pi^*_B\end{bmatrix}^T$ of the graph in Figure 3.8(a)

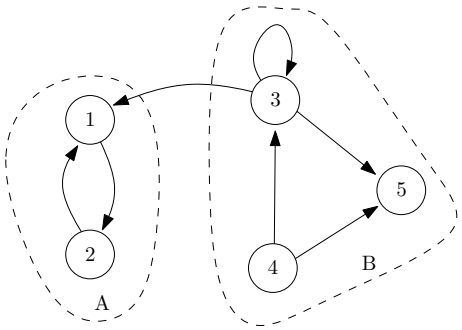


Figure 3.9 Subgraphs A and B are combined into a single graph.

with

$$\mathbf{H} = \begin{bmatrix} 1 & 0 \\ 1/3 & 2/3 \end{bmatrix}.$$

Note the uneven splitting of link weights from node B. This will be useful later in the problem.

(b) Compute the pagerank vectors $\begin{bmatrix} \pi_1^* & \pi_2^* \end{bmatrix}^T$ and $\begin{bmatrix} \pi_3^* & \pi_4^* & \pi_5^* \end{bmatrix}^T$ of the two graphs in Figure 3.8(b).

(c) If we divide the graph in Figure 3.7 into two blocks as shown in Figure 3.9, we can approximate π^* in the previous question by

$$\tilde{\pi}^* = \begin{bmatrix} \pi_A^* \cdot \begin{bmatrix} \pi_1^* & \pi_2^* \end{bmatrix}, & \pi_B^* \cdot \begin{bmatrix} \pi_3^* & \pi_4^* & \pi_5^* \end{bmatrix} \end{bmatrix}^T.$$

Compute this vector. Explain the advantage, in terms of computational load, of using this approximation instead of directly computing π^*.

3.5 *Personalized ranking (Open-ended)*

How would you solicit and aggregate individual user feedback to improve personalized ranking?

4 How does Netflix recommend movies?

We just saw three beautiful equations in the last three chapters, each used at least a billion times every single day:

$$p_i[t+1] = \frac{\gamma_i}{\mathrm{SIR}_i[t]} p_i[t]$$
$$U_i(\mathbf{b}) = v_i - p_i(\mathbf{b})$$
$$\pi^{*T} = \pi^{*T}\mathbf{G}.$$

We continue with our first block of four chapters that present four fundamental algorithms: Distributed power control, second price auction, pagerank, and now, collaborative filtering. These four chapters also introduce the basic languages of optimization, game, graph, and learning theories. A word of caution: as a chapter that introduces both the basic ideas in convex optimization and in machine learning, this chapter is among the longest in the book; you have to wait about 14 pages before we get to the most important idea on collaborative filtering for Netflix. This chapter is also mathematically more demanding than most of the other chapters.

4.1 A Short Answer

4.1.1 Recommendation problem

Netflix started its DVD rental business in 1997: instead of going to rental stores, you can just wait for DVDs to arrive by mail. Instead of incurring a late fee for each day you hold the DVD beyond the return date, you can keep holding the DVD as long as you continue to pay the monthly subscription fee, but you cannot receive a new DVD without returning the old one. This is similar in spirit to the sliding window mechanism of congestion control in Chapter 14, or the tit-for-tat incentive mechanism of P2P in Chapter 15. Netflix also maintained an efficient inventory control and mail delivery system. It operated with great *scalability* (the per-customer cost is much lower as the number of customers go up) and *stickiness* (users are reluctant to change the service). By 2008, there were about 9 million users in North America.

Then Netflix moved on to the next mode of delivering entertainment. This time it was streaming movies and TV programs from video servers, through the

Internet and wireless networks, to your Internet-connected devices: TVs, set-top boxes, game consoles, smart phones, and tablets. With its branding, choice of content, and aggressive pricing, Netflix's subscriber base nearly tripled to 23 million by April 2011. Netflix video streaming generated so much Internet traffic that over one in every four bits going through the Internet that month was Netflix traffic. In September 2011, Netflix announced that it would separate the DVD rental and online streaming businesses. Soon afterwards, Netflix reversed the decision, although the pricing for DVD rental and for online streaming became separated.

We will look at the cloud-based video distribution technology, including Netflix, Amazon Prime, Hulu, HBO Go, etc., in Chapter 17, and how that is changing the future of both entertainment and networking. In this chapter, we instead focus on the social network dimension used by Netflix: How does it recommend movies for you to watch (either by mail or by streaming)? It is like trying to read your mind and predict your movie rating. An effective **recommendation system** is important to Netflix because it enhances user experience, increases royalty and volume, and helps with inventory control.

A recommendation system is a helpful feature for many applications beyond video distribution. Similar to search engines in Chapter 3, recommendation systems give rise to structures in a "sea" of raw data and reduce the impact of information "explosion." Here are some representative systems of recommendation:

- You must have also noticed how Amazon recommends products to you based on your purchase and viewing history, adjusting its recommendation each time you browse a product. Amazon recommendation runs **content-based filtering**, in contrast to **collaborative filtering** used by Netflix. (A related question is when you can trust the averaged rating of a product on Amazon? It is a different variant of the recommendation problem, and will be taken up in Chapter 5.)
- You must have also been swayed by recommendations on YouTube that followed each of the videos you watched. We will look at YouTube recommendation in Chapter 7.
- You may have used Pandora's online music selection, where recommendation is developed by experts of music selection. But you get to thumb-up or thumb-down the recommendation; an explicit binary feedback.

Netflix instead wants to develop a recommendation system that does *not* depend on any expert, but uses the rich history of *all* the user behaviors to profile *each* user's taste in movies. This system has the following inputs, outputs, and criteria of success:

- Among the inputs to this system is the history of star ratings across all the users and all the movies. Each data point consists of four numbers: (a) user ID, indexed by u, (b) movie title, indexed by i, (c) number of stars, 1-5,

of the rating, denoted as r_{ui}, (d) date of the rating, denoted as t_{ui}. This is a really large data set: think of millions of users and tens of thousands movies. But only a fraction of users have watched a given movie, and only a fraction of that fraction actually bothered to rate the movie. Still, the size of this input is on the order of billions for Netflix. And the data set is also biased: which users have watched and rated which movie already provide much information about people's movie taste.

- The output is, first of all, a set of predictions $\hat{r}_{ui}$, one for each movie i that user u has not watched yet. These can be real numbers, not just integers like an actual rating r_{ui}. We can interpret a predicted rating of, say, 4.2 as saying that the user will rate this movie 4 stars with 80% probability and 5 stars with 20% probability. The final output is a short, ranked list of movies recommended to each user u, presumably those movies receiving $\hat{r}_{ui} \geq 4$, or the top 5 movies with the highest predicted $\hat{r}_{ui}$.
- The real test of this mind-reading system is whether user u actually likes the recommended movies. This information, however, is hard to collect. So a proxy used by Netflix is the **Root Mean Squared Error** (RMSE), measured for those (u, i) pairs where we have both the prediction and the actual rating. Let us say there are C such pairs. Each rating prediction's error is squared: $(r_{ui} - \hat{r}_{ui})^2$, and then averaged over all the predictions. Since the square was taken, to scale the numerical value back down, a square root is taken over this average:

$$\text{RMSE} = \sqrt{\sum_{(u,i)} \frac{(r_{ui} - \hat{r}_{ui})^2}{C}}.$$

The smaller the RMSE, the better the recommendation system. Netflix could have used the absolute value of the error instead of the squared error, but for our purposes we will stick to RMSE as the metric that quantifies the accuracy of a recommendation system. More importantly, regardless of the error metric it uses, in the end only the ranked order of the movies matter in recommendation. Only the top few in that rank ordered list are relevant as only they will be recommended to the user. The ultimate test is whether the user decides to watch the recommended movies, and whether she likes them or not. So, RMSE minimization is just a tractable approximation of the real problem of recommendation.

4.1.2 The Netflix Prize

Could recommendation accuracy be improved by 10% over what Netflix was using? That was the question Netflix presented to the research community in October 2006, through an open, online, international competition with a $1 million prize called the **Netflix Prize**.

The competition's mechanism is interesting in its own right. Netflix made

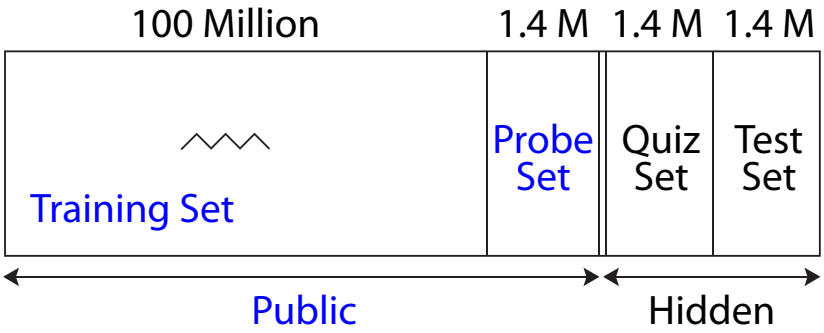


Figure 4.1 The Netflix Prize's four data sets. The training set and probe set were publicly released, whereas the quiz set and test set were hidden from the public and known only to Netflix. The probe, quiz, and test sets have similar statistical properties, but the probe set can be used by each competing team as often as they want, and the quiz set at most once a day. The final decision is based on comparison of RMSE on the test set.

available a set of over 100 million ratings, as part of its records from 1999 to 2005. That amount of data could really fit in the memory of standard desktops in 2006, making it easy for anyone in the world to participate in the competition. The rating data came from more than 480,000 users and 17,770 movies. On average, each movie was rated by more than 5000 users and each user rated more than 200 movies. But those average numbers disguise the real difficulty here: many users only rated a few movies, and very few users rated a huge number of movies (one user rated over 17,000 movies). Whatever recommendation system we use, it must predict well for *all* users.

The exact distribution of the data is shown in Figure 4.1:

- A little less than 100 million ratings were made public as the *training set*.
- About 1.4 million additional ratings were also made public and they had similar statistical properties as the test set and the quiz set described next. It was called the *probe set*, which competitors in Netflix Prize could use to test their algorithms.
- About 1.4 million additional ratings were hidden from the competitors, called the *quiz set*. Each competing team could submit an algorithm that would run on the quiz test, but not more than once a day. The RMSE scores continuously updated on the leaderboard on Netflix Prize website were based on this set's data.
- Another 1.4 million ratings, also hidden from the competitors, called the *test*

set. This was the real test. The RMSE scores on this set would determine the winner.

Each competing team first came up with a model for its recommendation system, then decided their model parameters' values based on minimizing the RMSE between known ratings in the training set and their model's predictions, and finally, used this model with tuned parameters to predict the unknown ratings in the quiz set. Of course, Netflix knows the actual ratings in the quiz set, and can evaluate RMSE between those ratings and the predictions from each team.

This was a smart arrangement. No team could reverse engineer the actual test set, since only scores on the quiz set were shown. It was also helpful to have a probe set where the competing teams could run their own tests as many times as they wanted.

Netflix had its own algorithm called Cinematch that gave an RMSE of 0.9514 on the quiz set if its parameters were tuned by the training set. Improving RMSE by even 0.01 can sometimes make a difference in the top 10 recommendations for a user. If the recommendation accuracy is improved by 10% over Cinematch, it would push RMSE to 0.8563 on the quiz set, and 0.8572 on the test set.

This Netflix Prize ignited the most intense and high-profile surge of activities in the research communities of machine learning, data mining, and information retrieval in recent years. To some researchers, the quality and sheer size of the available data was as attractive as the hype and prize. Over 5000 teams worldwide entered more than 44,000 submissions. Both Netflix and these research fields benefitted from the three year quest towards the goal of 10%. It turned out that setting the target as 10% improvement was a really good decision. For the given training set and quiz set, getting 8% improvement was reasonably easy, but getting 11% would have been extremely difficult.

Here are a few highlights in the history of Netflix Prize, also shown in the timeline in Figure 4.2: (a) Within a week of the start of the competition, Cinematch was beaten. (b) By early September, 2007, team BellKor made an 8.26% improvement over Cinematch, but first place changed hands a couple of times, until (c) in the last hour before the first year of competition ended, the same team got 8.43% improvement and won the $50,000 annual progress prize for leading the pack during the first year of the competition.

Then teams started merging. BellKor and BigChaos, two of the leading teams, merged and (d) received the 2008 progress prize for pushing the RMSE down to 0.8616. They further merged with Pragmatic Theory, and (e) the new team BellKor's Pragmatic Chaos became the first team that achieved more than 10% improvement in June 2009, beating Cinematch by 10.06%, on the quiz set. Then the competition entered the "last call" period: all teams had 30 days to make their final submissions. (f) At the end of this period, two teams beat Cinematch by more than 10% on the quiz set: BellKor's Pragmatic Chaos had an RMSE

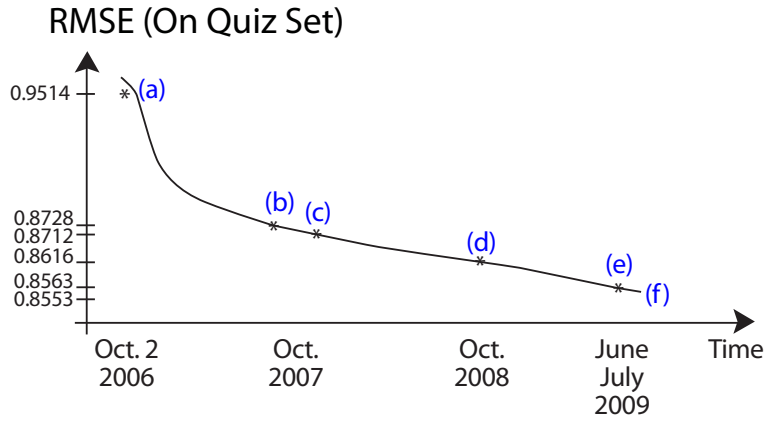


Figure 4.2 The Netflix Prize's timeline and some of the highlight events. It lasted for almost three years and the final decision came down to a twenty-minute differential. The y-axis shows the progress towards reducing RMSE on the quiz set data by 10% over the benchmark.

of 0.8554, and The Ensemble had an RMSE of 0.8553, slightly better. The final winner was to be declared by comparing their RMSEs on the test set.

Here is the grand finale: both teams beat Cinematch by more than 10% on the test set, and actually got the same RMSE on that set: 0.8567. But BellKor's Pragmatic Chaos submitted their algorithm 20 minutes earlier, and thus became the winner of the grand prize. A world-class science race lasting almost 3 years concluded with a 20 minute differential.

You must be wondering what algorithm BellKor's Pragmatic Chaos used in the final winning solution. The answer is documented in detail in a set of three reports, one from each component of this composite team, linked from the Netflix Prize website. But what you will find is that the winning solution is really a cocktail of many methods combined, with hundreds of ingredient algorithms blended together and thousands of model parameters fine-tuned specifically to the training set provided by Netflix. That was what it took to get that last 1% of improvement. But if you are only interested in the main approaches, big ideas, and getting 8-9% improvement over Cinematch, there are actually just a few key methodologies. And those are what we will focus on in the rest of this chapter.

4.1.3 Key ideas

To start with, take a look at the table in Figure 4.3. We can also think of the table as a matrix $\mathbf{R}$, or as a *weighted* bipartite graph where the user nodes are on the left column and the movie nodes on the right. A link connecting user node u and movie node i if u rated i, and the value of the rating as the weight of the

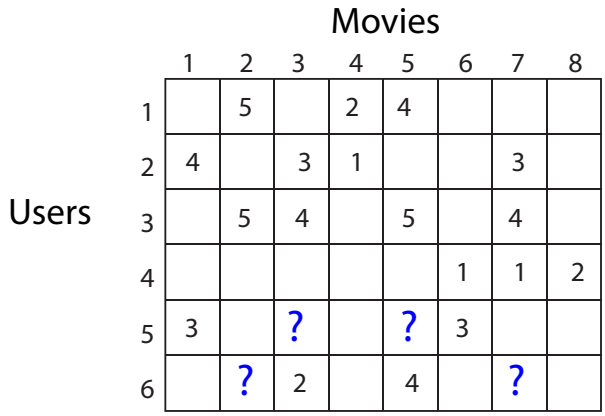


Figure 4.3 Recommendation system's problem: predicting missing ratings from given ratings in a large yet sparse table. In this small example of 6 users and 8 movies, there are 18 known ratings as a training set, and 4 unknown ratings to be predicted. Real problems are much larger, with billions of entries in the table, and sparse: only about 1% filled with known ratings.

link. In Chapter 8 we will discuss other matrices that describe the structure of different graphs.

Each column in this table is a movie (or an item in general), each row is a user, and each cell's number is the star rating by that user for that movie. Most cells are empty since only a few users rated a given movie. You are given a large yet sparse table like this, and asked to predict some missing entries like those four indicated by question marks in the last two rows in this table.

There are two main types of techniques for any recommendation system: content-based filtering and collaborative filtering.

Content-based filtering only looks at each row in isolation and attaches labels to the columns: if you like a comedy with Rowan Atkinson, you will probably like another comedy with Rowan Atkinson. This straightforward solution is often inadequate for Netflix.

In contrast, collaborative filtering exploits all the data in the entire table, across all the rows and all the columns, trying to discover *structures* in the pattern and values of the entries in this table.

Drawing an imperfect analogy with search engines in Chapter 3, content-based filtering is like the relevance score of individual webpages, and collaborative filtering is like the importance score determined by the connections among the webpages.

In collaborative filtering, there are in turn two main approaches.

- The intuitively simpler one is the **neighborhood model**. Here, two users are "neighbors" if they share similar tastes in movies. If Alice and Bob both

like "Schindler's List" and "Life is Beautiful," but not as much "E.T." and "Lion King", then knowing that Alice likes "Dr. Zhivago" would make us think Bob likes "Dr. Zhivago", too. In the neighborhood method, we first compute a **similarity score** between each pair of users. The larger the score, the closer these two users are in their taste for movies. Then for a given user whose opinion of a movie we would like to predict, we select, say, 50 of the most similar users who have rated that movie. Then take a weighted sum of these ratings and call that our prediction.

- The second approach is called the **latent factor model**. It assumes that underneath the billions of ratings out there, there are only a few hundred key factors on which users and movies interact. Statistical similarities among users (or among movies) are actually due to some hidden, low-dimensional structure in the data. This is a big assumption: there are many fewer *types* of people and movies than there are people and movies, but it sounds about right. It turns out that one way to represent a low-dimensional model is to factorize the table into two sets of *short vectors* of "latent factors."

Determining baseline predictors for the neighborhood model, or finding just the right short vectors in the latent factor model, boils down to solving **least squares** problems, also called **linear regressions**.

Most of the mileage in the leading solutions to the Netflix Prize was obtained by combining variants of these two approaches, supplemented by a whole bag of tricks. Two of these supplementary ideas are particularly interesting.

One is **implicit feedback**. A user does not have to rate a movie to tell us something about her mind. Which movies she browsed, which ones she watched, and which ones she bothered to rate at all are all helpful hints. For example, it is useful to leverage information in a binary table where each entry simply indicates whether this user rated that movie or not.

Another idea played an important role in pushing the improvement to 9% in the Netflix Prize: incorporating **temporal dynamics**. Here, the model parameters become time-dependent. This allows the model to capture changes in a person's taste and in trends of the movie market, as well as the mood of the day when a user rated movies, at the expense of dramatically increasing the number of model parameters to optimize over. One interesting observation is that when a user rates many movies on the same day, she tends to give similar ratings to all of these movies. By discovering and discounting these temporal features, the truly long-term structures in the training set are better quantified.

In the next section, we will present baseline predictor training and the neighborhood method, leaving the latent factor model to Advanced Material.

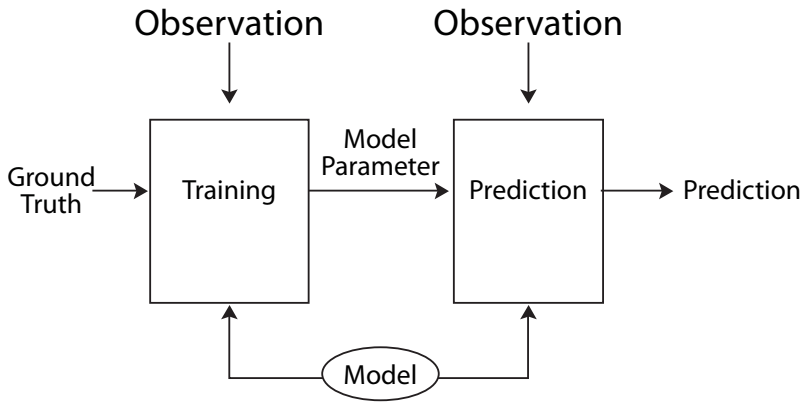


Figure 4.4 The main flow of two steps: first training and then prediction. The training module optimizes over model parameters using the known ground truth. Then the prediction module uses the models with optimized model parameters to make predictions.

4.2 A Longer Answer

Before diving into any specific predictors, let us take a look at the generic workflow consisting of two phases, as shown in Figure 4.4:

- *Training*: we put in a model (a mathematical representation of what we want to understand) with its parameters to work on the observed data, and then compare the predictions with the ground truth that we know in the training data. Then we tune the model parameters so as to minimize some error metric, like RMSE, relative to the ground truth that we know.
- *Prediction*: now we use the optimized model parameter to work on data observed beyond those in the training set. These predictions are then used in practice (or, in the case of the Netflix Prize, compared against a quiz set of ground truth that only Netflix knew).

4.2.1 Baseline predictor through least squares

Collaborative filtering extracts and leverages *structures* from the data set such as those in the table in Figure 4.3. But here is an easy method that does not even require any understanding of the structure: just use a simple averaging to compute the biases in the aggregate data. Let us start with this **baseline predictor**, and along the way introduce two important topics: the least squares problem and its solution, and time-dependent parameters that capture temporal shifts.

Suppose you look at the table of ratings in Figure 4.3 and decide *not* to

study the user-movie interactions. Instead, you just take the average of all the ratings out there, denoted as $\bar{r}$, and use that as the predictor for all $\hat{r}_{ui}$. In the Netflix Prize's training set, $\bar{r} = 3.6$. That is an extremely lazy and inaccurate recommendation system.

So how about incorporating two parameters: b_i to model the quality of each movie i relative to the average $\bar{r}$, and b_u to model the bias of each user u relative to $\bar{r}$? Some movies are of higher quality than average, while some users tend to give lower ratings to all movies. For example, "The Godfather" might have $b_i = 1.2$, but Alice tends to be a harsh reviewer with a $b_u = -0.5$. Then you might predict that Alice would rate "The Godfather" with $3.6 + 1.2 - 0.5 = 4.3$ stars.

If you think along this line, you will be using a model of baseline predictor:

$$\hat{r}_{ui} = \bar{r} + b_u + b_i. \tag{4.1}$$

We could have used the $b_u = \frac{\sum_i r_{ui}}{M_u}$, where M_u is the number of movies rated by user u, and $b_i = \frac{\sum_u r_{ui}}{M_i}$, where M_i is the number of users rated movie i. But that may not minimize RMSE. Instead, we choose the model parameters $\{b_u, b_i\}$ so that the resulting prediction's RMSE (for the training set's rating data) is minimized in the training phase. This is equivalent to:

$$\text{minimize}_{\{b_u, b_i\}} \sum_{(u,i)} (r_{ui} - \hat{r}_{ui})^2, \tag{4.2}$$

where the sum is, of course, only over (u, i) pairs in the training set where user u actually rated movie i, and the minimization is over all the $N + M$ parameters, where N is the number of users and M the number of movies in the training set. This type of optimization problem is called **least squares**. It is extensively studied and will be encountered later.

Least squares minimizes a convex *quadratic* function with *no* constraints, where the objective function is the sum of squares of some linear function of the variables. That is exactly what we have in (4.2).

To simplify the notation, just consider a very small example with one user (user 1) rating two movies (A and B) in the training set: r_{1A} and r_{1B}, with average rating $\bar{r} = (r_{1A} + r_{1B})/2$. The model parameters we have are b_1 (for b_u) and b_A and b_B (for b_i). The RMSE minimization boils down to minimizing the following convex, quadratic function

$$(b_1 + b_A + \bar{r} - r_{1A})^2 + (b_1 + b_B + \bar{r} - r_{1B})^2$$

over (b_1, b_A, b_B).

We can rewrite this minimization (4.2) in the standard form of a least squares problem, where $\mathbf{A}$ and $\mathbf{c}$ are given constant matrix and vector, respectively: minimize the sum of squares of all the elements in a vector $\mathbf{Ab} - \mathbf{c}$, *i.e.*, the square of the l-2 norm of $\mathbf{Ab} - \mathbf{c}$:

$$\|\mathbf{Ab} - \mathbf{c}\|_2^2,$$

where the subscript 2 denotes the l-2 norm. In this case, we have

$$\left\| \begin{pmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \end{pmatrix} \begin{pmatrix} b_1 \\ b_A \\ b_B \end{pmatrix} - \begin{pmatrix} r_{1A} - \bar{r} \\ r_{1B} - \bar{r} \end{pmatrix} \right\|_2^2 .$$

More generally, the variable vector $\mathbf{b}$ contains all the user biases and movie biases, and is thus $N + M$ elements long. Each entry in the constant vector $\mathbf{c}$ is the difference between the known rating r_{ui} in the training set and $\bar{r}$, thus C elements long. The constant matrix $\mathbf{A}$ is $C \times (N + M)$, with each entry being 0, unless the corresponding user has rated the corresponding movie, in which case it is 1.

You can take the first derivatives of this $\|\mathbf{Ab} - \mathbf{c}\|_2^2$ with respect to $\mathbf{b}$, and set them to zero. We know this is a minimizer because the objective function is convex.

By the definition of the l-2 norm, we have $\|\mathbf{x}\|_2^2 = \sum_i x_i^2$. So we can write $\|\mathbf{Ab} - \mathbf{c}\|_2^2$ as

$$(\mathbf{Ab} - \mathbf{c})^T (\mathbf{Ab} - \mathbf{c}) = \mathbf{b}^T \mathbf{A}^T \mathbf{Ab} - 2\mathbf{b}^T \mathbf{A}^T \mathbf{c} + \mathbf{c}^T \mathbf{c}.$$

Taking the derivative with respect to $\mathbf{b}$ and setting to 0 gives:

$$2\left(\mathbf{A}^T \mathbf{A}\right) \mathbf{b} - 2\mathbf{A}^T \mathbf{c} = 0. \tag{4.3}$$

You could also write out the quadratic function in long hand and take the derivative to validate the above expression.

So the least-squares solution $\mathbf{b}^*$ is the solution to the following system of linear equations:

$$\left(\mathbf{A}^T \mathbf{A}\right) \mathbf{b} = \mathbf{A}^T \mathbf{c}. \tag{4.4}$$

There are many ways to numerically solve this system of linear equations to obtain the result $\mathbf{b}^*$. This is a useful fact: minimizing (convex) quadratic functions boils down to solving linear equations, because we take the derivative and set it to zero.

In the above example of estimating three parameters (b_1, b_A, b_B) from two ratings (r_{1A}, r_{1B}), this linear equation (4.4) becomes:

$$\begin{pmatrix} 2 & 1 & 1 \\ 1 & 1 & 0 \\ 1 & 0 & 1 \end{pmatrix} \begin{pmatrix} b_1 \\ b_A \\ b_B \end{pmatrix} = \begin{pmatrix} r_{1A} + r_{1B} - 2\bar{r} \\ r_{1A} - \bar{r} \\ r_{1B} - \bar{r} \end{pmatrix}.$$

It so happens that there are an infinite number of solutions to the above linear equations. But in the realistic case where the number of model parameters is much less than the number of known ratings, like the one in the next section, we will not have that problem.

As explained in Advanced Material, least squares solutions often suffer from the **overfitting** problem. It fits the known data in the training set so well that

it loses the flexibility to adjust to a new data set. Then you have a super-refined explanatory model that loses its predictive power.

To avoid overfitting, a standard technique is called **regularization**. We simply minimize a weighted sum of (a) the sum of squared error and (b) the sum of squared parameter values:

$$\text{minimize}_{\{b_u,b_i\}} \sum_{(u,i)} (r_{ui} - \hat{r}_{ui})^2 + \lambda \left(\sum_u b_u^2 + \sum_i b_i^2 \right), \tag{4.5}$$

where $\hat{r}_{ui}$ is the baseline predictor (4.1), as a function of model parameters $\{b_u, b_i\}$. This balances the need to fit known data with the desire to use small parameters. The weight λ is chosen to balance these two goals. Bigger λ gives more weight to regularization and less to fitting the training data.

Since we picked the L-2 norm as the penalty function of $\{b_u, b_i\}$ in the regularization terms, (4.5) still remains a least squares problem: minimizing a convex quadratic function in the variables $\{b_u, b_i\}$, even though it is a different function from the one in (4.2).

In the rest of the chapter, we will subtract the baseline predictors from the raw rating data:

$$\tilde{r}_{ui} = r_{ui} - \hat{r}_{ui} = r_{ui} - (\bar{r} + b_u + b_i).$$

After this bias removal, in matrix notation, we have the following error matrix:

$$\tilde{\mathbf{R}} = \mathbf{R} - \hat{\mathbf{R}}.$$

4.2.2 Quick detour and generalization: Convex optimization

Least squares is clearly *not* a linear programming problem as introduced in Chapter 1. But it is a special case of **convex optimization**: minimizing a **convex function** subject to a **convex set** of constraints. Detecting convexity in a problem statement takes some experience, but defining the convexity of a set and of the convexity of a function is straightforward.

We call a *set* convex if the following is true: whenever two points are in the set, the line segment connecting them is also entirely in the set. So in Figure 4.5, (a) is a convex set but (b) is not. An important property of convex sets is that you can use a straight line (or, a hyperplane in more than 2 dimensions) to separate two (non-intersecting) convex sets.

We call a (twice differentiable) *function* convex if its second derivative is positive (and the domain of the function is a convex set). So in Figure 4.6, (a) is a convex function but (b) is not. In the case of a multivariate function f, the second derivative is a matrix, the **Hessian matrix**, where the $(i,j)th$ entry is the partial derivative of f with respect to the ith and jth argument of the function.

For example, the Hessian of $f(x,y) = x^2 + xy + y^2$ is

$$\begin{pmatrix} 2 & 1 \\ 1 & 2 \end{pmatrix}.$$

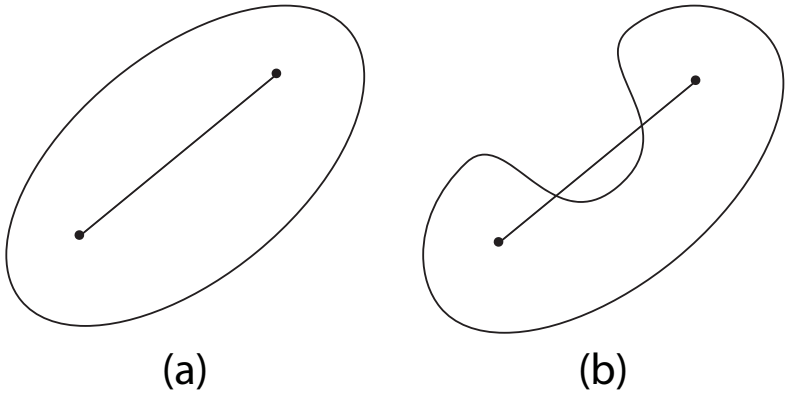


Figure 4.5 (a) is a convex set. (b) is not. A set is convex if for any two points in the set, the line segment in between them is also in the set. A key property of convex sets is that we can use a straight line (or a hyperplane in higher than two-dimensional spaces) to separate two (non-intersecting) convex sets. The constraint set of a convex optimization problem is a convex set. The domain of a convex function must be a convex set.

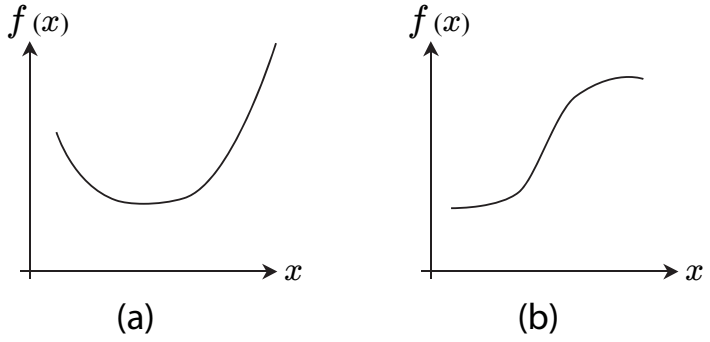


Figure 4.6 (a) is a convex function. (b) is not. A (twice differentiable) function is convex if its second derivative Hessian matrix is positive semidefinite. A function f is concave if $-f$ is convex. Some functions, like (b) above, have a convex part and a concave part. Linear functions are both convex and concave.

For the functions we will deal with in this book, there is a simple test of convexity. If the Hessian's eigenvalues are all non-negative, it is called a **positive semidefinite** matrix (not to be confused with positive matrix in Chapter 1);

then we say the "second derivative" is "positive," and the multivariate function is convex. For example, $f(x, y)$ above is convex.

Least squares is a convex quadratic minimization, since the Hessian of $\|\mathbf{Ab} - \mathbf{c}\|_2^2$ is $2\mathbf{A}^T\mathbf{A}$ (just take the derivative with respect to $\mathbf{b}$ of the left side of (4.3)), which is always positive semidefinite no matter what $\mathbf{A}$ is.

We can also define convexity for functions that are not smooth enough to have second derivatives. Roughly speaking, convex functions curve *upwards*. But since all functions we will see have first and second derivatives, let us stick to this easily-verifiable second derivative definition.

It turns out that convex optimization problems are easy to solve, both in theory and in practice, almost as easy to solve as linear programming problems. One justification is that, for a convex optimization problem, any locally optimal solution (no worse than other feasible solutions in a small neighborhood) is also a globally optimal solution (no worse than any other feasible solution). We will see several more justifications in later chapters. The "watershed" between easy and hard optimization is *convexity*, rather than linearity. This has been a veryuseful realization in the optimization theory community over the past two decades.

4.2.3 Quick detour: Baseline predictor with temporal models

If you compute the model parameters $\{b_u, b_i\}$ of this baseline predictor through the above least squares optimization (4.5) using the training set data $\{r_{ui}\}$, the accuracy of the prediction in the probe set or the quiz set will not be that impressive. But here is an idea that can substantially reduce the RMSE: incorporate temporal effects into the baseline model parameters.

Movies went in and out of fashion over the period of more than 5 years in the Netflix Prize's datasets. So b_i should not be just one number for user i, but a function that depends on what day it is. These movie trends do not shift substantially within the course of a day, but if we bin all the days in 5 years into 30 bins, each about 10 weeks long, we might expect some shift in b_i across the bins. Let us denote that shift, whether positive or negative, as $b_{i,\mathrm{bin}(t)}$, where time t is measured in days but then binned into 10-week periods. Then we have 30 additional model parameters for each movie i, in addition to the time-independent b_i:

$$b_i(t) = b_i + b_{i,\mathrm{bin}(t)}.$$

Each user's taste also changes over time. But taste is trickier to model than a movie's temporal effect. There is a continuous component $\sigma_u(t)$: user u's rating standard changes over time, and this deviation can be measured in several ways. There is also a *discrete* component $b_{u,t}$: big fluctuations on each day that a user rates movies. You might wonder why that is so. One reason is that one single user account on Netflix is often shared by a family, and the actual person giving the rating might be a different one on any given day. Another is the "batch rating effect:" when a user decides to rate many movies on the same day, these

ratings often fall into a much narrower range than they would have if not batch processed. This was one of the key insights among the hundreds of tricks used by the leading teams in the Netflix Prize. Putting these together, we have the following time-dependent bias term per user:

$$b_u(t) = b_u + \sigma_u(t) + b_{u,t}.$$

Now our baseline predictor becomes time-dependent too:

$$\hat{r}_{ui}(t) = \bar{r} + b_i(t) + b_u(t).$$

The least squares problem in (4.5) remains the same if we substitute $\hat{r}_{ui}$ by $\hat{r}_{ui}(t)$, and add regularization terms for all the additional time-dependent parameters defining $b_i(t)$ and $b_u(t)$.

We just added a lot of model parameters to the baseline predictor, but this is well worth it. Even if we stop right here with just the baseline predictor, *without* any discovery and exploitation of user-movie interactions in the training set, we can already achieve an RMSE of 0.9555, not far from the benchmark Cinematch's RMSE of 0.9514. Furthermore, the two families of methods we are going to describe next can also benefit from temporal effect models.

4.2.4 Neighborhood method: similarity measure and weighted prediction

We have said nothing about collaborative filtering yet. Everything so far is about taking averages in a row or a column of the rating matrix. For Netflix recommendations, we now move on to extract structures from user-movie interactions in the table shown in Figure 4.3. The neighborhood method is one of the two main approaches. It relies on *pairwise statistical correlation*:

- *User-user correlation*: Two users with similar ratings of movies in the training set are row-wise "neighbors:" they are likely going to have similar ratings for a movie in the quiz set. If one of the users rates movie i with 4 stars, the other user is likely going to rate it with 4 stars too.
- *Movie-movie correlation*: Two movies that got similar ratings from users in the training set are column-wise "neighbors:" they are likely going to have similar ratings by a user in the quiz set. If one of the movies is rated 4 stars by user u, the other movie is likely going to be rated 4 stars by this user too.

Both arguments sound intuitive. We mentioned user-user correlation in the first section, and we will focus on movie-movie correlation now.

Since there are so many (u, i) pairs in the training data, of course we want to leverage more than just one neighbor. "Neighbor" here refers to closeness in movies' styles. Given a movie i, we want to pick its L nearest neighbors, where "distance" among movies is measured by a **similarity metric**. A standard similarity metric is the **cosine coefficient** shown in Figure 4.7, where we view

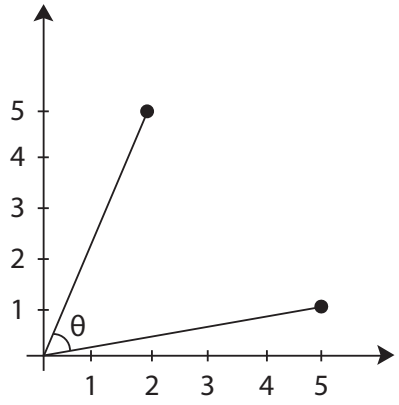


Figure 4.7 Given two points (in this case, in 2-dimensional space), we can measure how close they are by the sizes of the angle between the lines connecting each point to the origin. The cosine of this angle θ is the cosine similarity measure if we view these two points as two users' ratings on 2 movies, one on each axis.

each pair of columns in the $\tilde{\mathbf{R}}$ matrix as two vectors $\mathbf{r}_i$ and $\mathbf{r}_j$ in the Euclidean space, and the cosine of their angles is

$$d_{ij} = \frac{\mathbf{r}_i^T \mathbf{r}_j}{\|\mathbf{r}_i\|_2 \|\mathbf{r}_j\|_2} = \frac{\sum_u \tilde{r}_{ui} \tilde{r}_{uj}}{\sqrt{\sum_u (\tilde{r}_{ui})^2 \sum_u (\tilde{r}_{uj})^2}}, \tag{4.6}$$

where the summation is of course only over those users u that rated both movies i and j in the training set. We can now collect all the $\{d_{ij}\}$ in an M by M matrix $\mathbf{D}$ that summarizes all pairwise movie-movie similarity values.

Now for a given movie i, we rank all the other movies, indexed by j, in descending order of $|d_{ij}|$, and pick those top L movies as the neighbors that will count in neighborhood modeling. L can be some integer between 1 and $M-1$. Call this set of neighbors (for movie i) $\mathcal{L}_i$.

We say the predicted rating is simply the baseline predictor plus a weighted sum of ratings from these neighbor movies (and normalized by the weights). There can be many choices for these weights w_{ij}. One natural, though suboptimal, choice is to simply let $w_{ij} = d_{ij}$. There is actually no particular reason why similarity measure must also be the prediction weight, but for simplicity let us stick with that.

Then we have the following predictor, where N in $\hat{r}_{ui}^N$ denotes "neighborhood method:"

$$\hat{r}_{ui}^N = (\bar{r} + b_u + b_i) + \frac{\sum_{j \in \mathcal{L}_i} d_{ij} \tilde{r}_{uj}}{\sum_{j \in \mathcal{L}_i} |d_{ij}|}. \tag{4.7}$$

This equation is the most important one for solving the Netflix recommendation problem in this chapter. So we pause a little to examine it.

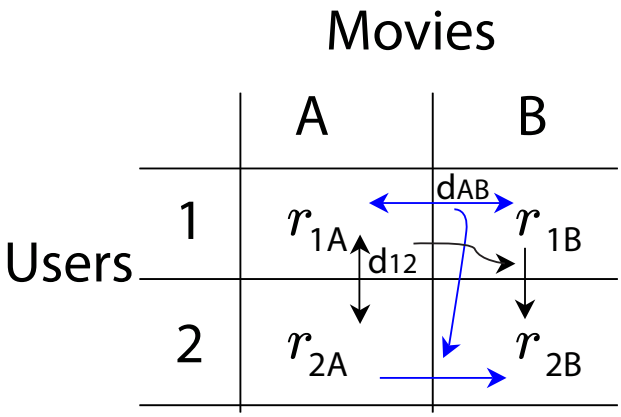


Figure 4.8 Two coordinates of collaborative filtering: user-user correlation vs. movie-movie correlation. Suppose we want to predict user 2's rating of movie B. We can either exploit the similarity between movies A and B and r_{2A}, or exploit the similarity between users 1 and 2 and r_{1B}.

- The three terms inside the bracket simply represent the estimate before taking advantage of similarities of users or movies.

- The quotient term is the weighted sum of the intelligence we gathered from collaborative filtering. The weights $\{d_{ij}\}$ can be positive or negative, since it is just as helpful to know two movies are very different as to know they are very similar. Of course, the magnitude normalization term in the denominator needs to take the absolute value of $\{d_{ij}\}$ to avoid a cancellation of the effects.

There are quite a few extensions to the neighborhood predictor shown above. Choosing an appropriate L and generalizing the choice of weights w_{ij} are two of those. We can also throw away those neighbors that only have very few overlaps, *e.g.*, if very few users rated both movies 1 and 2, then we might not want to count movie 2 as a neighbor of movie 1 even if d_{12} is large. We also want to discount those users that rate too many movies as all high or all low. These users tend to be not so useful in recommendation systems.

Now we can again collect all these predictors into a matrix $\hat{\mathbf{R}}^N$ and use that instead of just the baseline predictors $\hat{\mathbf{R}}$.

We have seen two styles of neighborhood models: user-user and movie-movie. But as you might suspect, these two are really two sides of the same coin, creating different "chains" of connection in the table of ratings. This is illustrated in Figure 4.8.

4.2.5 Summary

The procedure developed so far can be summarized into 5 steps:

1. Train a baseline predictor by solving least squares.
2. Obtain the baseline prediction matrix $\hat{\mathbf{R}}$, and shift $\mathbf{R}$ by $\hat{\mathbf{R}}$ to get $\tilde{\mathbf{R}}$.
3. Compute movie-movie similarity matrix $\mathbf{D}$.
4. Pick a neighborhood size L to construct a neighborhood of movies $\mathcal{L}$ for each movie i.
5. Compute the baseline predictor plus neighborhood predictor (4.7) as the final prediction for each (u, i) pair. This gives us $\hat{\mathbf{R}}^N$.

At the end of the above steps, Netflix can pick the top few unwatched and highly rated (by its prediction) movies to recommend to each user u. Or, as in the case of the Netflix Prize, it can compute RMSE against the ground truth in quiz and test sets to determine the prize winner.

4.3 Examples

We illustrate the baseline predictor and the neighborhood model with a simple example. The matrix $\mathbf{R}$ in this example consists of 40 hypothetical ratings from 10 users on 5 movies. This is an 80% dense matrix, *much* denser than a real system's data, but we are constrained by the size of a matrix that can be written out on one page. We will use 30 randomly chosen ratings as the training set, and the remaining 10, in boldface, as the test set. The other 10 missing ratings are denoted by "-". The average rating of this example matrix is $\bar{r} = 3.83$.

$$\mathbf{R} = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \\ 9 \\ 10 \end{array} \begin{array}{c} \begin{array}{ccccc} A & B & C & D & E \end{array} \\ \begin{pmatrix} 5 & 4 & 4 & - & \mathbf{5} \\ - & 3 & 5 & \mathbf{3} & 4 \\ 5 & 2 & - & \mathbf{2} & 3 \\ - & \mathbf{2} & 3 & 1 & 2 \\ 4 & - & \mathbf{5} & 4 & 5 \\ \mathbf{5} & 3 & - & 3 & 5 \\ 3 & \mathbf{2} & 3 & 2 & - \\ 5 & \mathbf{3} & 4 & - & 5 \\ \mathbf{4} & 2 & 5 & 4 & - \\ \mathbf{5} & - & 5 & 3 & 4 \end{pmatrix} \end{array}$$

4.3.1 Baseline predictor

The baseline predictor minimizes the sum of squares of all the elements in the following vector $\mathbf{Ab} - \mathbf{c}$:

$$
\begin{array}{c} \\ 1 \\ 2 \\ \vdots \\ 30 \end{array}
\begin{array}{c}
\begin{array}{cccccccccc} 1 & 2 & 3 & \cdots & 10 & A & B & \cdots & E \end{array} \\
\begin{pmatrix}
1 & 0 & 0 & \cdots & 0 & 1 & 0 & \cdots & 0 \\
0 & 0 & 1 & \cdots & 0 & 1 & 0 & \cdots & 0 \\
\vdots & & & & \ddots & & & \vdots & \\
0 & 0 & 0 & \cdots & 1 & 0 & 0 & \cdots & 1
\end{pmatrix}
\end{array}
\begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_{10} \\ b_A \\ b_B \\ \vdots \\ b_E \end{pmatrix}
-
\begin{pmatrix} r_{1A} - \bar{r} \\ r_{3A} - \bar{r} \\ \vdots \\ r_{10E} - \bar{r} \end{pmatrix}.
$$

Solving the above system of linear equations with 30 equations and 15 variables, we find the optimal user bias:

$$\mathbf{b}_u^* = [0.62, 0.42, -0.28, -1.78, 0.52, 0.49, -1.24, 0.45, 0.40, 0.23]^T,$$

and the optimal movie bias

$$\mathbf{b}_i^* = [0.72, -1.20, 0.60, -0.60, 0.33]^T.$$

These values quantify the intuition from what we observe from the training data. For example:

1. Users 1, 2, 5, 6 and 8 tend to give higher ratings;
2. Users 4 and 7 tend to give lower ratings;
3. Movies A and C tend to receive higher ratings;
4. Movies B and D tend to receive lower ratings.

We clip any predicted rating lower than 1 to 1 and any higher than 5 to 5, since it is ranking data we are dealing with. The rating matrix estimated by the baseline predictor (after clipping) is as follows:

$$
\hat{\mathbf{R}} = \begin{array}{c}
\begin{array}{ccccc} A & B & C & D & E \end{array} \\
\begin{pmatrix}
5.00 & 3.09 & 4.90 & - & \mathbf{4.62} \\
- & 2.89 & 4.69 & \mathbf{3.49} & 4.42 \\
4.10 & 2.19 & - & \mathbf{2.78} & 3.71 \\
- & \mathbf{1.00} & 2.49 & 1.29 & 2.22 \\
4.90 & - & \mathbf{4.79} & 3.58 & 4.51 \\
\mathbf{4.88} & 2.96 & - & 3.56 & 4.48 \\
3.15 & \mathbf{1.23} & 3.03 & 1.82 & - \\
4.84 & \mathbf{2.92} & 4.72 & - & 4.44 \\
\mathbf{4.84} & 2.92 & 4.72 & 3.51 & - \\
\mathbf{4.61} & - & 4.49 & 3.29 & 4.22
\end{pmatrix}
\end{array}
\begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \\ 9 \\ 10 \end{array}.
$$

We compute the RMSE between $\mathbf{R}$ and $\hat{\mathbf{R}}$ above. It is 0.51 for the training set and 0.64 for the test set.

The baseline predictor with parameters trained through least squares suffers

from the overfitting problem. Take the predicted rating for movie B by user 4 as an example: $\hat{r}_{4B} = \bar{r} + b_4 + b_B = 3.67 - 1.78 - 1.20 = 0.69$, significantly lower than the real rating $r_{4B} = 2$. Indeed, while the overall training RMSE is 0.51, the test error is 0.64.

4.3.2 Neighborhood model

The neighborhood model goes one step further to extract the user-movie interactions. We start with the differences between the raw ratings of the training set and the corresponding biases captured by the (unregularized) baseline predictor:

$$\tilde{\mathbf{R}} = \mathbf{R} - \hat{\mathbf{R}} = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \\ 9 \\ 10 \end{array} \begin{array}{c} \begin{array}{ccccc} A & B & C & D & E \end{array} \\ \begin{pmatrix} 0 & 0.91 & -0.90 & - & ? \\ - & 0.11 & 0.31 & ? & -0.42 \\ 0.90 & -0.19 & - & ? & -0.71 \\ - & ? & 0.51 & -0.29 & -0.22 \\ -0.90 & - & ? & 0.42 & 0.49 \\ ? & 0.040 & - & -0.56 & 0.52 \\ -0.15 & ? & -0.031 & 0.18 & - \\ 0.16 & ? & -0.72 & - & 0.56 \\ ? & -0.87 & 0.33 & 0.54 & - \\ ? & - & 0.51 & -0.29 & -0.22 \end{pmatrix} \end{array}$$

We used ? to indicate test set data, and $-$ to denote unavailable data (*i.e.*, user 1 never rated movie D).

We use the cosine coefficient to measure the similarity between movies represented in $\tilde{\mathbf{R}}$. Take the calculation of the similarity between movie B and movie C as an example. According to the training data in $\tilde{\mathbf{R}}$, users 1, 2 and 9 rated both movies. Therefore,

$$\begin{aligned} d_{BC} &= \frac{\tilde{r}_{1B}\tilde{r}_{1C} + \tilde{r}_{2B}\tilde{r}_{2C} + \tilde{r}_{9B}\tilde{r}_{9C}}{\sqrt{(\tilde{r}_{1B}^2 + \tilde{r}_{2B}^2 + \tilde{r}_{9B}^2)(\tilde{r}_{1C}^2 + \tilde{r}_{2C}^2 + \tilde{r}_{9C}^2)}} \\ &= \frac{(0.91)(-0.90) + (-0.11)(0.31) + (-0.87)(0.33)}{\sqrt{(0.91^2 + 0.11^2 + 0.87^2)(0.90^2 + 0.31^2 + 0.33^2)}} \\ &= -0.84. \end{aligned}$$

Similarly, we can calculate the entire similarity matrix, a 5×5 symmetric matrix (where the diagonal entries are not of interest since they concern the same movie):

$$\mathbf{D} = \begin{array}{c} \\ A \\ B \\ C \\ D \\ E \end{array} \begin{array}{c} \begin{array}{ccccc} A & B & C & D & E \end{array} \\ \begin{pmatrix} - & -0.21 & -0.41 & -0.97 & -0.75 \\ -0.21 & - & -0.84 & -0.73 & 0.51 \\ -0.41 & -0.84 & - & -0.22 & -0.93 \\ -0.97 & -0.73 & -0.22 & - & 0.068 \\ -0.75 & 0.51 & -0.93 & 0.068 & - \end{pmatrix} \end{array}.$$

With the above pairwise movie-movie similarity values, we can carry out the following procedure to compute $\hat{r}_{ui}$:

1. Find $L = 2$ movie neighbors with the largest absolute similarity values, $|d_{ik}|$ and $|d_{il}|$.
2. Check if user u has rated both movies k and l. If so, use both, as in the formula below. If the user rated only one of them, then just use that movie. If the user rated neither, then do not use the neighborhood method.
3. Calculate the predicted rating $\hat{r}_{u,i} = (\bar{r} + b_u + b_i) + \frac{d_{ik}\tilde{r}_{uk} + d_{il}\tilde{r}_{ul}}{|d_{ik}| + |d_{il}|}$.

Take $\hat{r}_{3D}$ for an example. The two nearest neighbors for movie D are movie A and movie B, whose cosine coefficients are -0.97 and -0.73 respectively. User 3 has rated both movie A and movie B. Hence,

$$\hat{r}_{3D} = (\bar{r}+b_3+b_D)+\frac{d_{DA}\tilde{r}_{3A} + d_{DB}\tilde{r}_{3B}}{|d_{DA}| + |d_{DB}|} = 2.78+\frac{-0.97 * 0.90 + (-0.73) * (-0.19)}{0.97 + 0.73} = 2.35.$$

In the baseline predictor, $\hat{r}_{3D} = 2.78$. So the neighborhood predictor reduces the error compared to the ground truth of $r_{3D} = 2$.

Similarly, the closest neighbors for movie B are movies C and D. From the training set, we know that user 2 only rated movie C but not D. Hence $\hat{r}_{2B} = (\bar{r} + b_2 + b_B) + d_{BC}\tilde{r}_{2C}/|d_{BC}| = 2.89 - 0.31 = 2.58$.

The predictions by the neighborhood model (after clipping) are given below:

$$\hat{\mathbf{R}}^N = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \\ 9 \\ 10 \end{array} \begin{array}{c} \begin{array}{ccccc} A & B & C & D & E \end{array} \\ \begin{pmatrix} 5.00 & 3.99 & 3.99 & - & \mathbf{5.00} \\ - & 2.58 & 4.86 & \mathbf{3.38} & 4.11 \\ 4.81 & 2.19 & - & \mathbf{2.35} & 2.81 \\ - & \mathbf{1.00} & 2.71 & 1.29 & 1.71 \\ 4.46 & - & \mathbf{4.30} & 4.49 & 5.42 \\ \mathbf{4.97} & 3.52 & - & 3.52 & 4.48 \\ 2.97 & \mathbf{1.16} & 3.03 & 1.97 & - \\ 4.28 & \mathbf{3.64} & 4.16 & - & 4.77 \\ \mathbf{4.25} & 2.44 & 5.54 & 4.33 & - \\ \mathbf{4.87} & - & 4.71 & 3.29 & 3.71 \end{pmatrix} \end{array}$$

The training error and test error using the neighborhood model are 0.34 and 0.54 respectively, compared to 0.51 and 0.64 using the baseline predictor. This represents a 16% improvement in RMSE for the test set.

We could also take a look at the errors by *Hamming distance*. If we just look at the wrong predictions (after rounding up or down to integers), it turns out all of them are off by 1 star. But the baseline predictor gets 5 wrong, whereas the neighborhood predictor gets 4 wrong, a 20% reduction in error.

Even in this extremely small example, the movie-based neighborhood method manages to take advantage of the statistical relationship between the movies and outperform the baseline predictor for both the training and test data sets.

In much larger and realistic cases, neighborhood methods have much more information to act on and can reduce RMSE much further as compared to a baseline predictor.

4.4 Advanced Material

4.4.1 Regularization: Robust learning without overfitting

Learning is both an exercise of *hindsight* and one of *foresight*: it should generate a model that fits training data, but also make predictions that match unseen data. There is a tradeoff between the two. You need reasonable hindsight to have foresight, but a perfect hindsight often means you are simply re-creating history. This is called overfitting, and is a particularly common problem when there is a lot of noise in your data (you end up modeling the noise), or there are too many parameters relative to the training data size (you end up creating a model tailor-made for the training data). Eventually it is the foresight that matters.

Overfitting is intuitively clear: you can always develop a model that fits training data perfectly. For example, my recommendation system for Netflix might be as follows: Alice rates 4 stars to a comedy starring Julia Roberts if she watches it on April 12, and 5 stars to a drama starring Al Pacino if she watches on March 25, etc. I can make it fit the training data perfectly by adding enough model parameters, such as lead actors and actresses, date of the user watching the movie, etc. But in doing so, the model is no longer simple, robust, or predictive.

We will go through an example of overfitting in learning in a homework problem. There are several ways to avoid overfitting. Regularization is a common one: add a penalty term that reduces the sensitivity to model parameters by rewarding smaller parameters. We measure size by some norm, and then we add this penalty term to the objective function quantifying learning efficiency. In our case, that efficiency is captured by RMSE. To maintain the least squares structure of the optimization over model parameters, it is a common practice to add the l-2 norm of the parameter values as the penalty term to control hindsight:

$$\text{minimize}_{\{\text{model parameters}\}} \ (\text{Squared error term}) + \lambda\,(\text{Parameter size squared}).$$

This results in the **regularized least squares** problem, and it can be efficiently solved, just like the original least squares. Statistically, this process will introduce bias to the parameter values.

The theme in regularization is to use parameters with smaller magnitudes to avoid "over optimization." Using fewer parameters is another path.

In Netflix recommendation, adding the regularization term $\lambda(\sum_u b_u^2 + \sum_i b_i^2)$ limits the magnitudes of the user and movie biases, which in turn helps bring the test error more in line with the training error. For the same example as in the last section, Figure 4.9 shows how the training and test errors vary with respect

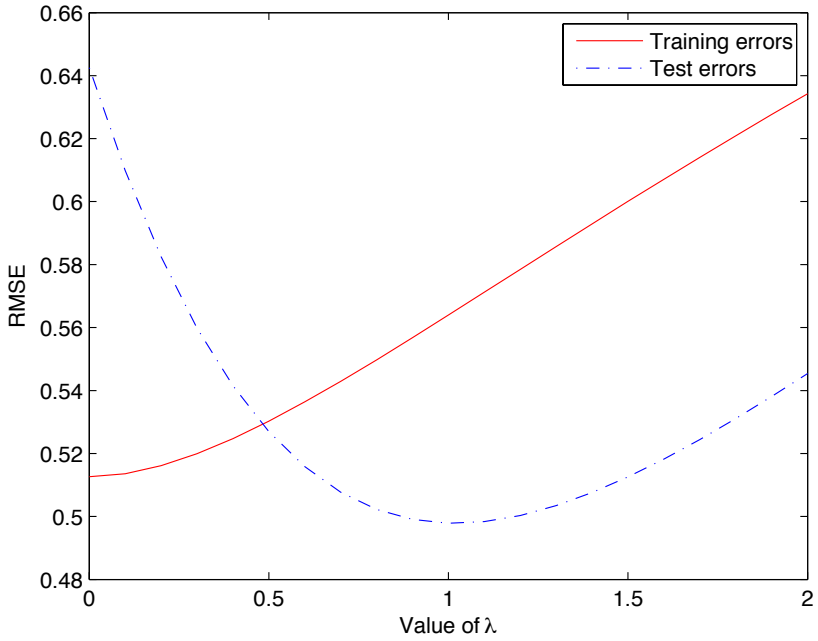


Figure 4.9 Effect of regularization on overfitting for baseline predictor. As the regularization weight parameter λ increases, the training set's RMSE increases but the test set's RMSE drops first. In hindsight, we see that $\lambda = 1$ turns out to be the right regularization weight value in this numerical example. Setting the right λ in each regularized least squares problem can be tricky.

to different values of λ. The test error first falls with more regularization and starts to rise once λ exceeds an optimal value, which happens to be at about 1 in this example.

After regularization with the best $\lambda = 1$, we get $\hat{\mathbf{R}}$ as

$$\hat{\mathbf{R}} = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \\ 9 \\ 10 \end{array} \begin{array}{c} \begin{array}{ccccc} A & B & C & D & E \end{array} \\ \begin{pmatrix} 4.70 & 3.23 & 4.61 & - & \mathbf{4.40} \\ - & 3.05 & 4.44 & \mathbf{3.38} & 4.23 \\ 4.01 & 2.53 & - & \mathbf{2.85} & 3.71 \\ - & \mathbf{1.47} & 2.86 & 1.80 & 2.65 \\ 4.67 & - & \mathbf{4.58} & 3.52 & 4.38 \\ \mathbf{4.54} & 3.07 & - & 3.39 & 4.25 \\ 3.37 & \mathbf{1.90} & 3.28 & 2.22 & - \\ 4.66 & \mathbf{3.18} & 4.57 & - & 4.36 \\ \mathbf{4.49} & 3.02 & 4.40 & 3.34 & - \\ \mathbf{4.45} & - & 4.36 & 3.29 & 4.15 \end{pmatrix} \end{array},$$

and the resulting RMSE as 0.56 for the training set and 0.50 for the test set.

The tricky part is to set the weight λ just right so as to strike the balance between hindsight and foresight. There are several techniques to do that, such as **cross-validation**, where we divide the available training data into K parts. In each of K rounds of cross-validation, we leave out one of the K parts as test data. Of course, a larger K provides more opportunities for validation, but leaves a smaller test data sample per round. Sometimes, a reasonable λ can be found by using any K picked from a wide range of values.

4.4.2 Latent factor method: matrix factorization and alternating projection

The latent factor method relies on *global structures* underlying the table in Figure 4.3. One of the challenges in recommendation system design is that the table is both *large* and *sparse*. In the case of the Netflix Prize, the table has about $N = 480,000$ times $M = 17,770$, *i.e.*, more than 8.5 billion cells, yet only a little more than 1% of those are occupied with ratings.

But we suspect there may be structures that can be captured by two much smaller matrices. We suspect that the similarities among users and movies are not just a statistical fact, but are actually induced by some *low-dimensional* structures hidden in the data. Therefore, we want to build a low-dimensional model for this high-dimensional data.

Towards this goal, we propose that for each user u, there is a K-dimensional vector $\mathbf{p}_u$ explaining her movie taste. And for each movie i, there is also a K-dimensional vector $\mathbf{q}_i$ explaining the movie's appeal. The inner product between these two vectors, $\mathbf{p}_u^T \mathbf{q}_i$, is the prediction $\hat{r}_{ui}$.

Typical numbers of K for the Netflix Prize were between 10 and 200. Let us consider a toy example where K is 2. One dimension is whether the movie is romantic vs. action, and the other is long vs. short. Let us say Alice likes romantic movies but is not too much into long movies: $\mathbf{p}_u = [1,\ 0.7]^T$. Let us say "Gone With the Wind" is pretty romantic and long: $\mathbf{q}_i = [2.5,\ 2]^T$. Then their inner product is $\mathbf{p}_u^T \mathbf{q}_i = 1 \times 2.5 + 0.7 \times 2 = 3.9$.

Now our job in the latent factor method is to fix K, the number of dimensions, and then optimize over all these latent factor vectors $\{\mathbf{p}_u, \mathbf{q}_i\}$ to minimize the Mean Squared Error (MSE) based on the given ratings $\{r_{ui}\}$ in the training set:

$$\text{minimize}_{\mathbf{P},\mathbf{Q}} \sum_{(u,i)} \left(r_{ui} - \mathbf{p}_u^T \mathbf{q}_i\right)^2. \tag{4.8}$$

Here, we collect the $\mathbf{p}_u$ vectors across all the users to form an $N \times K$ matrix $\mathbf{P}$, and we collect the $\mathbf{q}_i$ vectors across all the movies to form an $K \times M$ matrix $\mathbf{Q}$. Of course, we should also add quadratic regularization terms, penalizing the sum of squares of all the entries in $(\mathbf{P}, \mathbf{Q})$ as well.

Solving (4.8) we obtain $\mathbf{P}^*$ and $\mathbf{Q}^*$, and the resulting latent-factor prediction matrix is

$$\hat{\mathbf{R}}^L = \mathbf{P}^* \mathbf{Q}^*.$$

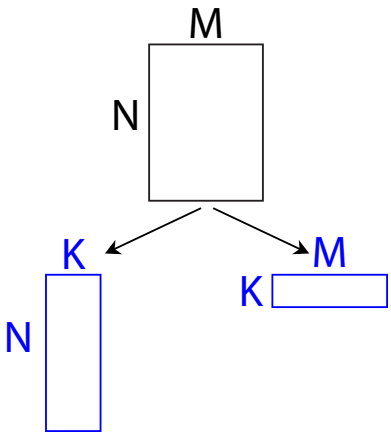


Figure 4.10 Factorize matrix $\mathbf{R}$ into a skinny matrix $\mathbf{P}$ and a thin matrix $\mathbf{Q}$. In the case of the Netflix Prize, $N = 480,000$ and $M = 17,770$, both very large, whereas the latent factor dimension is between $K = 10$ to 200, much smaller than either N or M.

How many model parameters are we talking about here? If $K = 200$, then we have $200 \times (480,000 + 17,770)$, a little less than 100 million parameters. That is almost the same as the number of the training set's rating data points. We can view $(\mathbf{P}, \mathbf{Q})$ as a matrix factorization of the much larger $N \times M$ matrix $\mathbf{R}$, as illustrated in Figure 4.10. By picking a K value so that $K(N + M)$ is about the same as the number of entries in the large, sparse matrix $\mathbf{R}$, we turn the sparsity in the high-dimensional data (given to us) into structures in a low-dimensional model (constructed by us).

How do we solve problem (4.8), or its regularized version? It might look like a least squares problem, but it is not. *Both* $\mathbf{p}_u$ and $\mathbf{q}_i$ are optimization variables now. In fact, it is not even a convex optimization. But if we hold $\mathbf{P}$ as constants and vary over $\mathbf{Q}$, or vice versa, it reduces to a least squares problem that we know how to solve. So a standard way to solve (4.8) is by **alternating projections**: hold $\mathbf{P}$ as constants and vary over $\mathbf{Q}$ in a least squares problem, then hold $\mathbf{Q}$ as constants and vary over $\mathbf{P}$ in another least squares problem, and repeat until convergence. Since this is a nonconvex optimization problem, the converged point $(\mathbf{P}^*, \mathbf{Q}^*)$ may not be globally optimal.

Coming back to those latent factors, what exactly are they? Many models may be able to fit given observations, but we hope some of them are also explainable through intuition. In the neighborhood method, we have an intuitive explanation of statistical correlation. But in the latent factor method, precisely where we expect to see some meaning of these latent factors, we actually do not have intuitive labels attached to all these K dimensions. If the prediction works out, these K dimensions are telling us something, but we might not be able to explain what that "something" is. Some people view this as a clever strength: ex-

tracting structures even when words fail us. Others consider it an unsatisfactory explanation of this "trick."

In any case, a straight-forward implementation of the latent factor model, with the help of temporal effect modeling, can give an RMSE of 0.8799. That is about an 8% of improvement over Cinematch, and enough to get you close to the leading position in the 2007 progress prize.

The difference between the neighborhood model and the latent factor model lies not just in statistical vs. structural understanding, but also in that the neighborhood model only utilizes pairwise (local) interactions, whereas the latent factor model leverages global interactions. But we can expand the neighborhood model to incorporate global interactions too. In fact, it has recently been shown that neighborhood models become equivalent to a particular matrix factorization model.

By combining the neighborhood method and latent factor method, with the help of temporal effect models and implicit feedback models, and after blending other tricks and carefully tuning all the parameters with the training set, in about two years after the 8% improvement was achieved, we finally saw the 10% mark barely reached by the two teams in July 2009.

Further Readings

The Netflix Prize was widely reported in popular science magazines from 2006 to 2009.

1. The original announcement by Netflix in October 2006 can be found here:
[Net06] The Netflix Prize, `http://www.netflixprize.com//rules`, 2006.

2. A non-technical survey of the first two years of progress can be found at
[Bel+09] R. Bell, J. Bennett, Y. Koren, and C. Volinsky, "The million dollar programming prize," *IEEE Spectrum*, May 2009.

3. A more technical survey can be found as a book chapter below:
[KB11] Y. Koren and R. Bell, "Advances in collaborative filtering," *Recommender Systems Handbook*, Springer, 2011.

4. Another survey written by part of the winning team of the Netflix Prize is the following one on factorization methods, which we followed in Advanced Material:
[KBV09] Y. Koren, R. Bell, and C. Volinsky, "Matrix factorization techniques for recommender systems," *IEEE Computer*, 2009.

5. This chapter introduces the basic notions of convex optimization. An excellent textbook for convex optimization and applications is

[BV04] S. Boyd and L. Vandenberghe, *Convex Optimization*, Cambridge University Press, 2004.

Problems

4.1 *Baseline predictor* ⋆

Compute the baseline predictor $\hat{\mathbf{R}}$ for the following $\mathbf{R}$:

$$\mathbf{R} = \begin{bmatrix} 5 & - & 5 & 4 \\ - & 1 & 1 & 4 \\ 4 & 1 & 2 & 4 \\ 3 & 4 & - & 3 \\ 1 & 5 & 3 & - \end{bmatrix}.$$

(Hint: This involves a least squares problem with 16 equations and 9 variables. Feel free to use any programming language. For example, backslash operator or `pinv()` in Matlab. If there are multiple solutions to the least squares problem, take any one of these.)

4.2 *Neighborhood predictor* ⋆⋆⋆

Using the $\mathbf{R}$ and the computed $\hat{\mathbf{R}}$ from the previous question, compute the neighborhood predictor $\hat{\mathbf{R}}^N$ with $L = 2$.

4.3 *Least squares* ⋆⋆

(a) Solve for $\mathbf{b}$ in the following least squares problem, by hand or programming in any language:

$$\text{minimize}_{\mathbf{b}} \qquad \|\mathbf{A}\mathbf{b} - \mathbf{c}\|_2^2,$$

where

$$\mathbf{A} = \begin{bmatrix} 1 & 0 & 2 \\ 1 & 1 & 0 \\ 0 & 2 & 1 \\ 2 & 1 & 1 \end{bmatrix} \qquad \text{and} \qquad \mathbf{c} = \begin{bmatrix} 2 \\ 1 \\ 1 \\ 3 \end{bmatrix}.$$

(b) Solve the above least squares problem again with regularization. Vary the regularization parameter λ for $\lambda = 0, 0.2, 0.4, \ldots, 5.0$, and plot both $\|\mathbf{A}\mathbf{b} - \mathbf{c}\|_2^2$ and $\|\mathbf{b}\|_2^2$ against λ.

(Hint: take derivative of

$$\|\mathbf{A}\mathbf{b} - \mathbf{c}\|_2^2 + \lambda\|\mathbf{b}\|_2^2$$

with respect to $\mathbf{b}$ to obtain a system of linear equations.)

4.4 *Convex functions* ⋆

Determine whether the following functions are convex, concave, both, or neither:
(a) $f(x) = 3x + 4$, for all real x;
(b) $f(x) = 4\ln\left(\frac{x}{3}\right)$, for all $x > 0$;
(c) $f(x) = e^{2x}$, for all real x;
(d) $f(x, y) = -3x^2 - 4y^2$, for all real x and y;
(e) $f(x, y) = xy$, for all real x and y.

(Hint: f is convex if and only if its Hessian matrix $\mathbf{H}$ is positive semidefinite. Similarly, f is concave if and only if its Hessian $\mathbf{H}$ is negative semidefinite.)

4.5 *Log-Sum-Exp and geometric programming* ⋆⋆

(a) Is $\exp(x + y)$ convex or concave in (x, y)?

(b) Is $\exp(x + y) + \exp(2x + 5y)$ convex or concave in (x, y)?

(c) Is $\log(\exp(x+y)+\exp(2x+5y))$ convex or concave in (x, y)? This log-sum-exp function is heavily used in a class of convex optimization called **geometric programming** in fields like statistical physics, communication systems, and circuit design.

(d) Can you turn the following problem in variables $x, y, z > 0$ into convex optimization?

$$\begin{array}{ll} \text{minimize} & xy + xz \\ \text{subject to} & x^2yz + xz^{-1} \leq 10 \\ & 0.5x^{-0.5}y^{-1} = 1. \end{array}$$

5 When can I trust an average rating on Amazon?

Starting with this chapter, in the next four chapters we will walk through a remarkable landscape of intellectual foundations. But sometimes we will also see significant gaps between theory and practice.

5.1 A Short Answer

We continue with the theme of recommendation. Webpage ranking in Chapter 3 turns a graph into a ranked order list of nodes. Movie ranking in Chapter 4 turns a weighted bipartite user-movie graph into a set of ranked order lists of movies, with one list per user. We now examine the aggregation of a vector of rating scores by reviewers of a product or service, and turn that vector into a scalar, one per product. These scalars may in turn be used to rank order a set of similar products. In Chapter 6, we will further study aggregation of many vectors into a single vector.

When you shop on Amazon, likely you will pay attention to the number of stars shown below each product. But you should also care about the number of reviews behind that averaged number of stars. Intuitively, you know that a product with 2 reviews, both 5 stars, may not be better than a competing product with 100 reviews and an average of 4.5 stars, especially if these 100 reviews are all 4 and 5 stars and the reviewers are somewhat trustworthy. We will see how such intuition can be sharpened.

In most online review systems, each review consists of three fields:

1. Rating (a numerical score often on the scale of 1-5 stars). This is the focus of our study.
2. Review (text).
3. Review of review (often a binary up or down vote).

Rarely do people have time to read through all the reviews, so a summary review is needed to aggregate the individual reviews. What is a proper aggregation? That is the subject of this chapter.

Ratings are often not very trustworthy, and yet they are important in so many contexts, from peer reviews in academia to online purchases of every kind. The hope is that the following two approaches can help:

First, we need methods to ensure some level of accuracy, screening out the really bad ones. Unlimited and anonymous reviews have notoriously poor quality, because a competitor may enter many negative reviews, the seller herself may enter many positive reviews, or someone who has never even used the product or service may enter random reviews. So before anything else, we should first check the mechanism used to enter reviews. How strongly are customers encouraged, or even rewarded, to review? Do you need to enter a review of reviews before you are allowed to upload your own review? Sometimes a seemingly minor change in formatting leads to significant differences: Is it a binary review of thumbs up or down, followed by a tally of up vs. down votes? What is the dynamic range of the numerical scale? It has been observed that the scale of 1-10 often returns 7 as the average and then a bimodal distribution around it. A scale of 1-3 gives a very different psychological hint to the reviewers compared to a scale of 1-5, or a scale of 1-10 compared to -5 to 5.

Second, the review population size needs to be large enough to wash out the inaccurate ones. But *how* large is large enough? And can we run the raw ratings through some signal processing to get the most useful aggregation?

These are tough questions with no good answers yet, not even well formulated problem statements. The first question depends on the nature of the product being reviewed. Movies (*e.g.*, on IMDB) are very subjective, whereas electronics (*e.g.*, on Amazon) are much less so, with hotels (*e.g.*, on tripadvisor) and restaurants (*e.g.*, on opentable) somewhere in between. It also depends on the quality of the review, although reputation of the reviewer is a difficult metric to quantify in its own right.

The second question depends on the metric of "usefulness." Each user may have a different metric, and the provider of the service or product may use yet another one. This lack of clarity in what should be optimized is the crux of the ill-definedness of the problem at hand.

With these challenges, it may feel like opinion aggregation is unlikely to work well. But there have been notable exceptions recorded for some special cases. A famous example is Galton's 1906 observation on a farm in Plymouth, UK, where 787 people in a festival there participated in a game of guessing the weight of an ox, each writing down a number *independent* of others. There was also no common bias; everyone could take a good look at the ox. While the estimates by each individual were all over the place, the average was 1197 pounds. It turned out the ox weighed 1198 pounds. Just a simple averaging worked remarkably well. For the task of guessing the weight of an ox, 787 was more than enough to get the right answer (within a margin of error of 0.1%).

But in many other contexts, the story is not quite as simple as Galton's experiment. There were several key factors here that made simple averaging work so well:

- The task is relatively easy; in particular, there is a correct objective answer with a clear numerical meaning.

- The estimates are both unbiased and independent of each other.
- There are enough people participating.

More generally, three factors are important in aggregating individual action:

- *Definition of the task*: Guessing a number is easy. Consensus formation in social choice is hard. Reviewing a product on Amazon is somewhere in between. Maybe we can define "subjectivity" by the size of the review population needed to reach a certain "stabilization number."
- *Independence of reviews*: As we will see, the wisdom of crowds, if there is one to the degree we can identify and quantify, stems not from having many smart individuals in the crowd, but from the independence of each individual's view from the rest. Are Amazon reviews independent of each other? Kind of. Even though you can see the existing reviews before entering your own, usually your rating number will not be significantly affected by the existing ratings. Sometimes, reviews are indeed entered as a reaction to recent reviews posted on the website, either to counter-argue or to reinforce points made there. This influence from the sequential nature of review systems will be partially studied in Chapter 7.
- *Review population*: For a given task and degree of independence, there is correspondingly a minimum number of reviews, a threshold, needed to give a target confidence of trustworthiness to the average. If these ratings pass through some signal processing filters first, then this threshold may be reduced.

What kind of signal processing do we need? For text reviews, there needs to be natural language tools, *e.g.*, detecting inconsistencies or extreme emotions in a review and discounting it and its associated rating. We in academia face this problem in each decision on a peer-reviewed paper, a funding proposal, a tenure-track position interview, and a tenure or promotion case.

For rating numbers, some kind of weighting is needed, and we will discuss a particularly well-studied one soon. In Chapter 6, we will also discuss voting methods, including majority rule, pairwise comparison, and positional counting. These voting systems require each voter to provide a complete ranking, possibly implicitly, with a numerical rating scale. Therefore, we will have more information, perhaps too much information, as compared to our current problem.

5.2 Challenges of rating aggregation

Back to rating aggregation. Here are several examples illustrating three of the key challenges in deciding when to trust ratings on Amazon.

Example 1. Many online rating systems use a naive averaging method for their product ratings. Moreover, given that different products have different numbers

of reviews, it is hard to determine which product has a better quality. For example, in Figure 5.1, Philips 22PFL4504D HDTV has 121 ratings with a mean of 4, while Panasonic VIERA TC-L32C3 HDTV has 55 ratings with a mean of 4.5. So the customer is faced with a tradeoff between choosing a product with a lower average rating and a larger number of reviews versus one with a higher average rating and a smaller number of reviews.

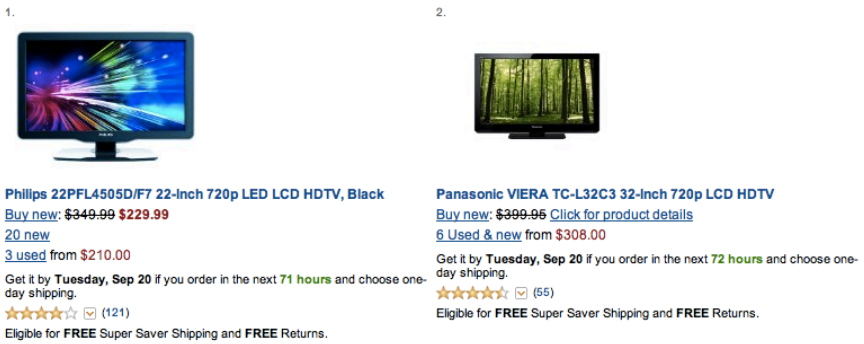


Figure 5.1 Tradeoff between review population and average rating score. Should a product with fewer reviews but higher average rating be ranked higher than a competing product with more ratings but lower average rating?

Example 2. Consider 2 speaker systems for home theater. Both RCA RT151 and Pyle Home PCB3BK have comparable mean scores around 4. 51.9% of users gave RCA RT151 a rating of 5 stars while 7.69% gave 1 star. On the other hand, 54.2% of users gave 5 stars to Pyle Home PCB3BK while 8.4% gave 1 star. So Pyle Home PCB3BK speaker has not only a higher percentage of people giving it 5 stars than RCA RT151, but also has a higher percentage of people giving it 1 star. There is a larger *variation* in the ratings of Pyle Home PCB3BK than RCA RT151.

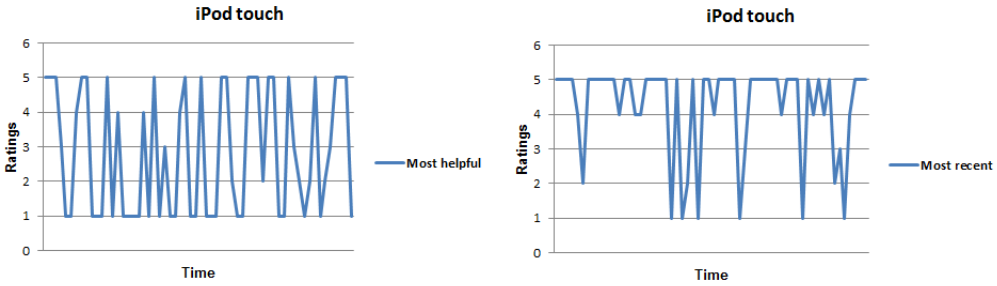


Figure 5.2 How to view the aggregated ratings: should it be based on helpful ratings or on the latest trend? Here the same set of iPod touch ratings on Amazon is used to extract two different subsets of ratings, and their values are quite different.

Example 3. In Figure 5.2, we compare the mean rating of the first 60 "most

helpful" reviews of iPod3 Touch (32 GB) on Amazon with the mean from the 60 most recent ratings. The ratings are on the y-axis and the times of the ratings (index) are on the x-axis. The mean of the most recent ratings is 1.5 times greater than the mean corresponding to the most helpful reviews. Is this a "real" change or just noise and normal fluctuation? What should the timescale of averaging be?

At the heart of these problems is the challenge of turning *vectors* into *scalars*, which we will meet again in Chapter 6. This can be a "lossy compression" with very different results depending on how we run the process, *e.g.*, just look at the difference between mean and median.

5.3 Beyond basic aggregation of ratings

We may run a time-series analysis to understand the dynamics of rating. In Figure 5.3, the three curves of ratings entered over a period of time give the same average, but "clearly" some of them have not converged to a stable average rating. What kind of *moving window size* should we use to account for cumulative average and variance over time?

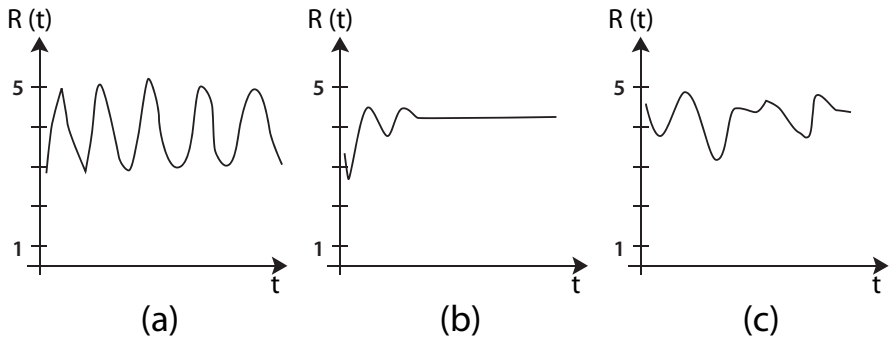


Figure 5.3 Three time series with the same long-term average rating but very different stabilization behavior. The time axis scale is in the order of weeks. (a) shows continued cyclic fluctuations of ratings R over time t. (b) shows a clear convergence. (c) shows promising signs of convergence but it is far from clear that the ratings have converged. Of course, the timescale also matters.

We may consider detecting anomalous ratings and throwing out the highly suspicious ones. If we detect a trend change, that may indicate a change of ownership or generational upgrade. And if such detection is accurate enough, we can significantly discount the ratings before this time. For ratings on the scale of 1-5, the coarse granularity makes this detection more difficult.

We may consider zooming into particular areas of this vector of ratings, *e.g.*, the very satisfied customers and the very dissatisfied ones, although it is often the case that those who care enough to enter ratings are either extremely satisfied or reasonably dissatisfied. There might be a bimodal distribution in the underlying customer satisfaction for certain products, but for many products there is often another bimodal distribution on the biased sampling based on who cared enough to write reviews.

Across all these questions, we can use the cross-validation approach from Chapter 4 to train and test the solution approach. If we can stand back 1 year and predict the general shape of ratings that have unfolded since then, that would be a strong indicator of the utility of our signal processing method.

But these questions do not have well-studied answers yet, so we will now focus instead on some simpler questions as proxies to our real questions: Why does simple averaging sometimes work, and what to do when it does not?

5.4 A Long Answer

5.4.1 Averaging a crowd

We start from a significantly simplified problem. Take the Galton example, and say the number that a crowd of N people wants to guess is x, and each person i in the crowd makes a guess y_i:

$$y_i(x) = x + \epsilon_i(x),$$

i.e., the true value plus some error ϵ_i. The error depends on x but not other j; it is independent of other errors. This error can be positive or negative, but we assume that it averages across different x to be 0; it has no bias. In reality, errors are often neither independent nor unbiased. Sequential estimates based on publicly announced estimates made by others may further exacerbate the dependence and bias. We will see examples of such information cascades in Chapter 7.

We measure error by mean squared error (MSE), just like what we did in Chapter 4. We want to compare the following two quantities:

- The average of individual guesses' errors.
- The error of the averaged guess.

The average of errors and the error of the average are not the same, and we will see how much they differ. Since x is a number that can take on different values with different probabilities, we should talk about the *expected MSE*, where the expectation $\mathbf{E}_x$ is the averaging procedure over the probability distribution of x.

The average of (expected, mean squared) errors (AE), by definition, is

$$E_{AE} = \frac{1}{N}\sum_{i=1}^{N} \mathbf{E}_x \left[\epsilon_i^2(x)\right]. \tag{5.1}$$

On the other hand, the (expected, mean squared) error of the average (EA) is

$$E_{EA} = \mathbf{E}_x \left[\left(\frac{1}{N} \sum_{i=1}^{N} \epsilon_i(x) \right)^2 \right] = \frac{1}{N^2} \mathbf{E}_x \left[\left(\sum_{i=1}^{N} \epsilon_i(x) \right)^2 \right] \tag{5.2}$$

since the error term is now

$$\frac{1}{N} \sum_i y_i - x = \frac{1}{N} \left(\sum_i y_i - Nx \right) = \frac{1}{N} \left(\sum_i (y_i - x) \right) = \frac{1}{N} \left(\sum_i \epsilon \right).$$

It looks like (5.1) and (5.2) are the same, but they are not: sum of squares and square of sum are different. Their difference is a special case of Jensen's inequality on convex quadratic functions. There are many terms in expanding the square in (5.2), some are ϵ_i^2, and others are $\epsilon_i \epsilon_j$ where $i \neq j$. For example, if $N = 2$, we have one cross-term:

$$(\epsilon_1 + \epsilon_2)^2 = \epsilon_1^2 + \epsilon_2^2 + 2\epsilon_1 \epsilon_2.$$

These cross terms $\{\epsilon_i \epsilon_j\}$ take on different values depending on whether the estimates $\{y_i\}$ are independent or not. If they are independent, we have

$$\mathbf{E}_x[\epsilon_i(x)\epsilon_j(x)] = 0, \quad \forall i \neq j.$$

In that case, all the cross terms in expanding the square are zero, and we have

$$E_{EA} = \frac{1}{N} E_{AE}. \tag{5.3}$$

If you take the square root of MSE to get RMSE, the scaling in (5.3) is then $1/\sqrt{N}$.

This may appear to be remarkable. In such a general setting and using such an elementary derivation, we have mathematically crystallized (a type of) the wisdom of crowds in terms of efficiency gain: error is reduced by a factor as large as the size of the crowd if we average the estimates first, provided that the estimates are *independent* of each other. But this result holds for a crowd of 2 as much as it holds for a crowd of 1000. Some people think there must be something beyond this analysis, which is essentially the Law of Large Numbers at work: variance is reduced as the number of estimates go up. There should be some type of the wisdom of crowd that only shows up for a large enough crowd. Furthermore, we have so far assumed there is no systematic bias; averaging will not help reduce any bias that is in everyone's estimate.

The $1/N$ factor is only one dimension of the wisdom of crowds, what we refer to as "multiplexing gain" from independent "channels." We will later see "diversity gain," symbolically summarized as $1 - (1 - p)^N$.

What if the estimates are completely *dependent*? Then the averaged estimate is just the same as each estimate, so the error is exactly the same, and it does not matter how many reviews you have:

$$E_{EA} = E_{AE}. \tag{5.4}$$

In most cases, the estimates are somewhere in between completely independent and completely dependent. We have seen that each person in the crowd can be quite wrong. What is important is that they are wrong in independent ways. In statistical terms, we want their pairwise correlations to be small. Of course, if we could identify who in the crowd have the correct estimates, we should just use their estimates. So the wisdom of crowds we discussed is more about achieving robustness arising out of independent randomization than getting it right by identifying the more trustworthy estimates.

5.4.2 Bayesian estimation

Bayesian analysis can help us quantify the intuition that the number of ratings, N, should matter. Let us first get a feel for the Bayesian view with a simple, illustrative example. We will then go from one product to many products being ranked in the Bayesian way.

Suppose you run an experiment that returns a number, and you run it n times (the same experiment and independent runs). Suppose that s times it returns an outcome of 1. What do you think is the chance that the next experiment, the $(n+1)$th one, will return an outcome of 1 too? Without going into the foundation of probability theory, the answer is the intuitive one:

$$\frac{s}{n}.$$

Now if you know the experiment is actually a flip of a biased coin, what do you think is the chance that the next experiment will be positive? Hold on, is that the same question?

Actually, it is not. Now you have a *prior* knowledge: you know there are two possible outcomes, one with probability p for head, and the other $1-p$ for tail. That prior knowledge changes the derivation, and this is the essence of the Bayesian reasoning.

We first write down the probability distribution of p given that s out of n flips showed heads. Intuitively, the bigger s is, the more likely the coin is biased towards head, and the larger p is. This is the essence of the Bayesian view: more observations makes the model better.

Now, if p were *fixed*, then the probability of observing s heads and $n-s$ tails follows the Binomial distribution:

$$\left(\begin{array}{c} n \\ s \end{array} \right) p^s (1-p)^{n-s}. \tag{5.5}$$

So the probability distribution of p must be *proportional* to (5.5). This is the key step in Laplace's work that turned Bayes insights into a systematic mathematical language.

This is perhaps less straightforward that it may sound. We are "flipping the table" here. Instead of looking at the probability of observing s out of n heads

for a given p, we looking at the probability distribution of p that gave rise to this observation in the first place, since we have the observation but not p.

Once the above realization is internalized in your brain, the rest is easy. The probability distribution of p is proportional to (5.5), but we need to divide it by a normalization constant so that it is between 0 and 1. Knowing $p \in [0, 1]$, the normalization constant is simply:

$$\int_0^1 \left(\begin{array}{c} n \\ s \end{array} \right) p^s (1-p)^{n-s} dp.$$

Using beta function to get the above integral, we have:

$$f(p) = \frac{\left(\begin{array}{c} n \\ s \end{array} \right) p^s (1-p)^{n-s}}{\int_0^1 \left(\begin{array}{c} n \\ s \end{array} \right) p^s (1-p)^{n-s} dp} = \frac{(n+1)!}{s!(n-s)!} p^s (1-p)^{n-s}.$$

Finally, since *conditional* probability of seeing a head given p is just p, the *unconditional* probability of seeing a head is simply

$$\int_0^1 p f(p) dp,$$

an integral that evaluates to

$$\frac{s+1}{n+2}.$$

A remarkable and remarkably simple answer, this is called the **rule of succession** in probability theory. It is perhaps somewhat unexpected. The intermediate step to understanding p's distribution is not directly visible in the final answer, but that was in the core of the innerworking of Bayesian analysis. It is similar in spirit to the latent factor model in Chapter 4, and to other hidden-factor models like hidden Markov models used in many applications, from voice recognition to portfolio optimization.

Why is it *not* s/n? One intuitive explanation is that if you know the outcome must be success or failure, it is as if you have already seen 2 experiments "for free", 1 success and 1 failure. If you incorporate the prior knowledge in this way, then the same intuition on the case without prior knowledge indeed gives you $(s+1)/(n+2)$.

5.4.3 Bayesian ranking

So why is Bayesian analysis related to ratings on Amazon? Because ratings' population size matters. Back to our motivating question: should a product with only 2 reviews, even though both are 5 stars, be placed higher than a competing product with 100 reviews that averages 4.5 stars? Intuitively, this would be wrong. We should somehow weight the raw rating scores with the population sizes, just like we weighted a node's importance by the in-degree in Chapter

3. Knowing how many reviews there are gives us a prior knowledge, just like knowing a coin shows up heads 100 times out of 103 flips is a very different observation than knowing it shows up heads 3 times out of 13 flips.

More generally, we can think of a "sliding ruler" between the average rating of all the products, R, and the averaged rating of brand i, r_i. The more reviews there are for brand i relative to the total number of reviews for all the brands, the more trustworthy r_i is relative to R. The resulting Bayesian rating for brand i is

$$\tilde{r}_i = \frac{NR + n_i r_i}{N + n_i}. \tag{5.6}$$

We may also want to put an upper bound on N, for otherwise as time goes by and N monotonically increases, the dynamic range of the above ratio can only shrink.

Quite a few websites adopt Bayesian ranking. The Internet Movie DataBase (IMDB)'s top 250 movies ranking follows (5.6) exactly. So the prior knowledge used is the average of all the movie ratings.

Beer Advocate's top beer ranking *e.g.*, `http://beeradvocate.com/lists/popular` uses the following formula:

$$\frac{N_{min}R + n_i r_i}{N_{min} + n_i},$$

where N_{min} is the minimum number of reviews needed for a beer to be listed there. Perhaps a number in between N and N_{min} would have been a better choice, striking a tradeoff between following the Bayesian adjustment exactly and avoiding the saturation effect (when some beers get a disproportionately large numbers of reviews).

All of the above suffer from a drawback in their assuming that there is a single, "true" value of a product's ranking, as the mean of some Gaussian distribution. But some products simply create bipolar reactions: some love it and some hate it. The idea of Bayesian ranking can be extended to a multinomial model and the corresponding Dirichlet prior.

Of course, this methodology only applies to adjusting the ratings of each brand within a comparable family of products, so that proper ranking can be achieved based on $\{\tilde{r}_i\}$. It cannot adjust ratings without this backdrop of a whole family of products that provides the *scale* of relative trustworthiness of ratings. It is good for *ranking*, but not for *rating* refinement, and it does not take into account time series analysis.

5.5 Examples

5.5.1 Bayesian ranking changes order

Consider Table 5.5.1, a compilation of ratings and review populations for MacBooks. The items are listed in descending order of their average rating. Following

(5.6), the Bayesian rankings for each of the five items can be computed. First, we compute the product NR as follows:

$$NR = \sum_i n_i r_i = 10{\times}4.920{+}15{\times}4.667{+}228{\times}4.535{+}150{\times}4.310{+}124{\times}4.298 = 2332.752$$

Then, with $N = \sum_i n_i = 527$, we apply (5.6) to each of the items:

$$\bar{r}_1 = \frac{2332.752 + 10 \times 4.920}{527 + 10} = 4.436,$$

$$\bar{r}_2 = \frac{2332.752 + 15 \times 4.667}{527 + 15} = 4.433,$$

$$\bar{r}_3 = \frac{2332.752 + 228 \times 4.535}{527 + 228} = 4.459,$$

$$\bar{r}_4 = \frac{2332.752 + 150 \times 4.310}{527 + 150} = 4.401,$$

$$\bar{r}_5 = \frac{2332.752 + 124 \times 4.298}{527 + 124} = 4.402.$$

These calculations and the Bayesian-adjusted rankings are shown in Table 5.5.1. All of the MacBook's ranking positions change after the adjustment is applied, because Bayesian adjustment takes into account the number of reviews as well as the average rating for each item. The third MacBook (MB402LL/A) rises to the top because the first and second were rated by far less people, and as a result, the ratings of both of these items drop significantly.

MacBook	Total Ratings	Average Rating	Rank	Bayes Rating	Bayes Rank
MB991LL/A	10	4.920	1	4.436	2
MB403LL/A	15	4.667	2	4.433	3
MB402LL/A	228	4.535	3	4.459	1
MC204LL/A	150	4.310	4	4.401	5
MB061LL/A	124	4.298	5	4.402	4

Table 5.1 An example where average ratings and Bayesian-adjusted ratings lead to entirely different rankings of the items. For instance, though the first listed MacBook (MB991LL/A) has the highest average, this average is based on a small number of ratings (10), which lowers its Bayes ranking by two places.

5.5.2 Bayesian ranking quantifies subjectivity

Sometimes Bayesian adjustment does not alter the ranked order of a set of comparable products, but we can still look at the "distance" between the original average rating and the Bayesian adjusted average rating across these products, *e.g.*, using l-2 norm of the difference between these two vectors.

For example, the adjusted rating for a set of digital cameras and women's shoes is computed in Table 5.2 and Table 5.3, respectively. This distance, normalized by the number of products in the product category, is 0.041 for digital cameras and 0.049 for shoes. This difference of almost 20% is a quantified indicator about the higher subjectivity and stronger dependence on review population size for fashion goods compared to electronic goods.

Digital Camera	Number of reviews, n_i	Mean Rating, r_i	Bayesian Rating, $\tilde{r}_i$
Canon Powershot	392	4.301	4.133
Nikon S8000	163	3.852	4.008
Polaroid 10011P	168	3.627	3.965

Table 5.2 Bayesian adjustment of ratings for 3 digital cameras, with a total of 723 ratings. The L-2 distance between the vector of mean ratings and that of Bayesian ratings is 0.023.

Women's Shoes	Number of reviews, n_i	Mean Rating, r_i	Bayesian Rating, $\tilde{r}_i$
Easy Spirit Traveltime	150	3.967	4.182
UGG Classic Footwear	148	4.655	4.289
BearPaw Shearling Boots	201	4.134	4.204
Skechers Shape-Ups	186	4.344	4.245
Tamarac Slippers	120	3.967	4.189

Table 5.3 Bayesian adjustment of ratings for 5 women's fashion shows, with a total of 805 ratings. The L-2 distance between the vector of mean ratings and that of Bayesian ratings is 0.041, about twice the difference in the case of digital camera example.

5.5.3 What does Amazon do

On Amazon, each individual product rating shows only the raw scores (the averaged number of stars), but when it comes to ranking similar products by "average customer review," it actually follows some secret formula that combines raw score with three additional elements:

- Bayesian adjustment by review population

- Recency of the reviews
- Reputation score of the reviewer (or quality of review, as reflected in review of review). See for example `www.amazon.com/review/top-reviewers-classic` for the hall of fame of Amazon reviewers.

The exact formula is not known outside of Amazon. In fact, even the reviewer reputation scores, which leads to a ranking of Amazon reviewers, follows some formula that apparently has been changed three times in the past years and remains a secret. Obviously, how high a reviewer is ranked depends on how many yes/useful votes (say, x) and no/not-useful votes (say, y) are received by each review she writes. If x is larger than a threshold, either x itself or the fraction $x/(x+y)$ can be used to assess this review's quality. And the reviewer reputation changes as some moving window average of these review quality measures change over time. The effect of fan vote or loyalty vote, where some people always vote yes on a particular reviewer's reviews, is then somehow subtracted. We will see in Chapter 6 that Wikipedia committee elections also follow some formula that turns binary votes into a ranking.

Let us consider the list of the top 20 LCD HDTVs of size 30 to 34 inches in April 2012, "top" according to average customer reviews. It can be obtained from Amazon by the following sequence of filters: Electronics > Television & Video > Televisions > LCD > 30 to 34 inches.

There are actually three rank ordered lists:

- The first is the ranking by Amazon, which orders the list in Table 5.5.3.
- Then there are the numerical scores of "average customer review", which, interestingly enough, does not lead to the actual ranking provided by Amazon.
- There are also the averaged rating scores, which lead to yet another rank order.

First, we look at the difference between the Amazon ordering and how the HDTVs would have been ranked had they been sorted based only on the average customer ratings. The two lists are as follows:

- 1, 2, 7, 12, 14, 3, 4, 5, 8, 9, 15, 16, 20, 6, 10, 11, 18, 13, 17, 19.
- 1, 7, 12, 14, 2, 5, 4, 8, 3, 9, 15, 16, 20, 18, 6, 10, 11, 13, 17, 19.

Clearly, the average customer review ranking is closer to the actual ranking. It follows the general trend of the actual ranking, with a few outliers: 7, 12, 14, 20, and 13 all seem to be strangely out of order. Let us try to reverse-engineer what other factors might have contributed to the actual ranking:

1. *Bayesian adjustment*: The population size of the ratings matter. The raw rating scores must be weighted with the population size in some way.
2. *Recency of the reviews*: Perhaps some of the reviewers rated their HDTVs as soon as they purchased them, and gave them high ratings because the products worked initially. But, especially with electronics, sometimes faulty

HDTV	Total reviews	5 star	4 star	3 star	2 star	1 star	Avg. review	Avg. rating
1	47	37	8	1	1	0	4.7	4.723
2	117	89	19	0	3	6	4.6	4.556
3	315	215	61	19	9	11	4.5	4.460
4	180	116	47	9	2	6	4.5	4.472
5	53	36	12	3	1	1	4.5	4.528
6	111	71	19	6	6	9	4.2	4.234
7	22	16	4	2	0	0	4.6	4.636
8	56	43	5	3	1	4	4.5	4.464
9	130	89	22	8	4	7	4.4	4.400
10	155	96	26	11	9	13	4.2	4.181
11	231	135	48	17	15	16	4.2	4.173
12	8	5	3	0	0	0	4.6	4.625
13	116	55	35	9	5	12	4.0	4.000
14	249	175	60	3	3	8	4.6	4.570
15	8	5	1	2	0	0	4.4	4.375
16	34	20	8	4	0	2	4.3	4.294
17	47	20	14	6	5	2	4.0	3.957
18	44	20	20	1	1	2	4.2	4.250
19	56	24	17	4	5	6	3.9	3.857
20	7	3	3	1	0	0	4.3	4.286

Table 5.4 List of the top twenty 30 to 34 inch LCD HDTVs on Amazon when sorted by average customer review.

components cause equipment to stop working over time. As a result, recent reviews should be considered more credible.

3. *Quality of the reviewers or reviews*: (a) Reputation score of the reviewer: Reviewers with higher reputations should be given more "say" in the average customer review of a product. (b) Quality of review: The quality of a review can be measured in terms of its length or associated keywords in the text. (c) Review of review: Higher review scores indicate that customers found the review "helpful" and accurate. (d) Timing of reviews: Review spamming from competing products can be detected based upon review timing.

Bayesian adjustment will be performed using equation (5.6). Here, R is the averaged rating of all the products (here assumed to be the top 20), and N is either the total number of reviews or, as in the Beer Advocate's website, is the *minimum* number of reviews necessary for a product to be listed, or possibly some mid-range number. Some number in-between these extremes is most likely, as lots of reviews are entered on Amazon, and the Bayesian adjustment will saturate as N keeps increasing and become simply R. From the above table, we can compute $R = \sum_i n_i r_i / \sum_i n_i = 4.36$. Now, what to choose for N? We compare the Bayesian adjustment rankings for $N_{min} = 7$, which is the lowest number of reviews for any product (20), $N_{max} = 249$, which is the highest number of reviews for any product (14), $N_{avg} = 100$, which is the average of

the reviews across the products, and $N_{sum} = 1986$, the total number of reviews entered for the top 20 products. The results are as follows:

- N_{min}: 1, 7, 14, 2, 5, 12, 4, 3, 8, 9, 15, 20, 16, 18, 6, 10, 11, 13, 17, 19
- N_{max}: 1, 14, 2, 3, 4, 5, 7, 8, 9, 12, 15, 20, 16, 18, 6, 17, 10, 11, 19, 13
- N_{avg}: 14, 1, 2, 3, 4, 5, 7, 8, 9, 12, 15, 20, 16, 18, 6, 10, 11, 17, 19, 13
- N_{sum}: 14, 18, 2, 1, 3, 4, 7, 8, 9, 20, 6, 13, 10, 17, 5, 11, 12, 19, 15, 16

Clearly, having N too large or too small is undesirable: Both result in rankings that are far out of order. Both N_{max} and N_{avg} give better results, at least in terms of grouping clusters together. Still, there are some outliers in each case: 14, 15, 20, 6, 10, and 11 are considerably out of order.

We notice that products 12, 15, and 20 all have a very small number of ratings, specifically 8, 8, and 7, respectively. But even so, why would they be placed so far apart in the top 20? To answer this, we take into account the review of reviews: In the case of product 12, for instance, the "most helpful" review had 26 people find it helpful, whereas in the cases of products 15 and 20 it was 6 and 3, respectively. In addition, we can look at the recency of the reviews: The "most helpful" review for product 12 was made on November of 2011. Product 15's "saving grace" is that its corresponding review was more recent, made in December of 2011, which would push it closer to product 12 in the rankings. On the other hand, Amazon may have deemed that product 20's review in July of 2011 was too outdated. Finally, product 12 had an extremely high quality of review in its descriptive listing the pros and cons in each case. Amazon probably trusts this integrity.

On the other hand, why would Amazon decide to rank an item such as 6 so high, given that the Bayesian adjustment places it around rank 15? Well, when we look at item 6, we see that its "most helpful" review had 139 out of 144 people find it helpful, and similar percentages exist for all reviews below it. Further, the reviewers all have high ratings, one of which is an Amazon "top reviewer".

The final point of discussion is why product 14 is ranked so low in the top 20, but has one of the first three positions on each of the Bayesian adjustments. This can be explained by a few factors:

- The most helpful review was from 2010, extremely outdated.
- The 8 reviewers who gave it 1 star all said that the TV had stopped working after a month, many of whom were high up in the "helpful" rankings. These reviewers dramatically increased the spread of the ratings and opinions for this product.

To summarize, the following set of guidelines is inferred from this (small) sample on how Amazon comes up with its rankings:

1. An initial Bayesian ranking is made, with N chosen to be somewhere around N_{max} or N_{avg}.

2. Products that have small numbers of reviews or low recency of their most helpful reviews are ranked separately amongst themselves, and re-distributed in the top 20 at lower locations (*e.g.*, products 12, 15, and 20).
3. Products that have very high quality, positive reviews from top reviewers are bumped up in the rankings (*e.g.*, product 6).
4. Products that could cause a potential risk to their sales due to the possibility of faulty electronics (*e.g.*, product 14) are severely demoted in the rankings.

5.6 Advanced Material

As we just saw, review of review can be quite helpful: higher scores means more confidence in the review, and naturally leads to a heavier weight for that review. If a review does not have any reviews, we may take some number between the lowest score for a review and the average score for all reviews. More generally, we may take the Bayesian likelihood approach to determine the trustworthiness of a review based on the observation of the reviews it receives.

While a systematic study of the above approach for rating analytics is still ongoing, this general idea of weighting individual estimates based on each estimate's effectiveness has been studied in a slightly different context of statistical learning theory, called **boosting**. If we view each estimator as a person, boosting shows how they can collaborate, through sequential decision making, to make the resulting estimate much better than any individual one can be.

5.6.1 Adaptive boosting

Ada Boost, short for adaptive boosting, captures the idea that by *sequentially* training estimators, we can make the average estimator more accurate. It is like an experience many students have while reviewing class material before an exam. We tend to review those points that we already know well (since that makes us feel better), while the right approach is exactly the opposite: to focus on those points that we do not know very well yet.

As in Figure 5.4, consider N estimators $y_i(\mathbf{x})$ that each map an input $\mathbf{x}$ into an estimate, *e.g.*, the number of stars in an aggregate rating. The final aggregate rating is a weighted sum: $\sum_i \alpha_i y_i$, where $\{\alpha_i\}$ are scalar weights and $\{y_i\}$ are functions that map vector $\mathbf{x}$ to a scalar. The question is how to select the right weights α_i. Of course, those y_i that are more accurate deserve a larger weight α_i. But how much larger?

Let us divide the training data into M sets indexed by j: $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_M$. For each training set, there is a right answer $t_j,\ j = 1, 2, \ldots, M$, known to us since we are training the estimators. So now we have N estimators and M data sets. As each estimator y_i gets trained by the data sets, some data sets are well handled while others are less so. We should *adapt* accordingly, and give challenging data

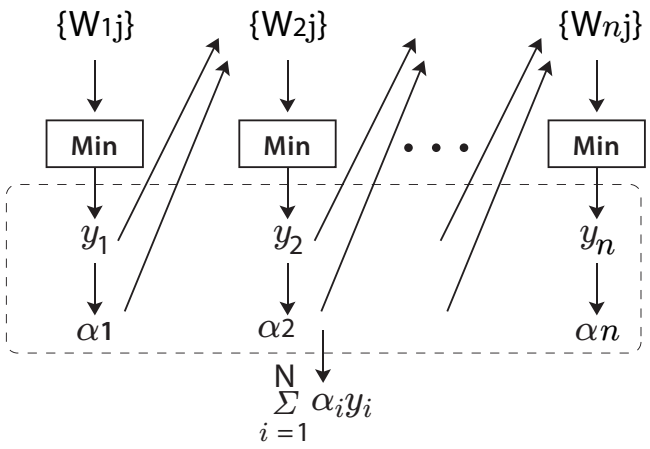


Figure 5.4 The schematic of Ada Boosting. There are N estimators, indexed by i: $\{y_i\}$, and M training data sets, indexed by j. Training is done to minimize weighted errors, where the weights w_{ij} are sequentially chosen from i to $i+1$ according to how well each training data set is learned so far. The final estimator is a weighted sum of individual estimators, where the weights $\{\alpha_i\}$ are also set according to the error of each estimator y_i.

sets, those leading to poor performance thus far, more weight w in the next estimator's parameter training.

So both the training weights $\{w_{ij}\}$ and the estimator combining weights $\{\alpha_i\}$ are determined by the performance of the estimators on the training sets.

We start by initializing the training weights w_{1j} for estimator 1 to be even across the data sets:

$$w_{1j} = \frac{1}{M}, \;\; j = 1, 2, \ldots, M.$$

Then *sequentially* for each i, we train estimator y_i by minimizing:

$$\sum_{j=1}^{M} w_{ij} 1_{y_i(\mathbf{x}_j) \neq t_j},$$

where 1 is an indicator function, which returns 1 if the subscript is true (the estimator is wrong) and 0 otherwise (the estimator is correct).

After this minimization, we get the resulting estimator leading to an error indicator function abbreviated as 1_{ij}: the optimized error indicator of estimator i being wrong on data set j.

The (normalized and weighted) sum of these error terms becomes:

$$\epsilon_i = \frac{\sum_j w_{ij} 1_{ij}}{\sum_j w_{ij}}.$$

Let the estimator combining weight for estimator i be the (natural) log scaled

version of this error term:

$$\alpha_i = \log(1/\epsilon_i - 1).$$

Now we update the training weights for the next estimator $i + 1$:

$$w_{i+1,j} = w_{ij} e^{\alpha_i 1_{ij}},$$

which turns into w_{ij} if estimator i gets data set j right, and $w_{ij}(1/\epsilon_i - 1)$ otherwise. If the jth training data set is poorly estimated, then $w_{i+1,j} > w_{i,j}$ in the next estimator, just as we desired.

Now we repeat the above loop for all the estimators i to compute all the α_i. Then the aggregate estimator is

$$y(\mathbf{x}) = \sum_{i=1}^{N} \alpha_i y_i(\mathbf{x}).$$

Clearly, we put more weight on estimators that are more accurate; the smaller ϵ_i, the bigger α_i. This Ada Boost method is illustrated in Figure 5.5.

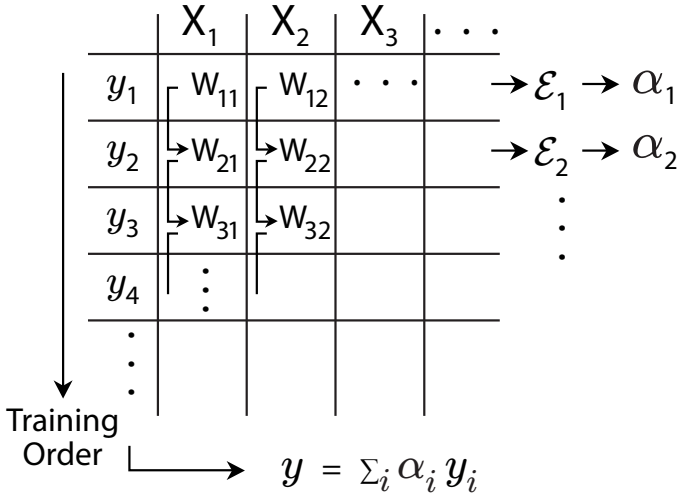


Figure 5.5 A detailed view of the sequence of the steps in the iterations of adaptive boosting. Each column represents a training data set. Each row represents an estimator. Sequential training happens down each column and across the rows. Performance per row is summarized by α_i, which defines the linear combination y as the final estimator derived from the wisdom of crowds.

Further reading

There is a gap between the rich theory of signal processing, statistical learning, and Bayesian analysis on the one hand, and the simple but tricky question of when can I trust a product rating average on Amazon.

1. The following blog provides a concise summary of Bayesian ranking:
`http://www.thebroth.com/blog/118/bayesian-rating`
The next blog adds further details:
`http://andrewgelman.com/2007/03/bayesian-sortin/`
And this one too, especially on using a Dirichlet prior for a multinomial model:
`http://masanjin.net/blog/how-to-rank-products-based-on-user-input`

2. The following popular science book has a basic mathematical treatment of averaging effect (in the Notes of the book):
[Fis09] L. Fisher, *The Perfect Swarm*, Basic Books, 2009.

3. A standard textbook on machine learning, including different ways to combine estimators like Ada Boost, is
[Bis06] C. M. Bishop, *Pattern Recognition and Machine Learning*, Springer, 2006.

4. For a graduate level comprehensive treatment on the subject of Bayesian approach to data analysis, the following is a standard choice:
[GCSR04] A. Gelman, J. B. Carlin, H. S. Stern, and D. B. Rubin, *Bayesian Data Analysis*, 2nd Ed., Chapman and Hall/CRC, 2004.

5. The following is an excellent book organizing the methodologies of analyzing patterned structures in signals around 6 prominent applications:
[MD10] D. Mumford and A. Desolneux, *Pattern Theory: The Stochastic Analysis of Real-World Signals*, A. K. Peters, 2010.

Problems

5.1 *Bayesian ranking* ⋆

Suppose there are 50 total ratings for printers with an average score of 4. Given rankings 5,5,5 for printer brand Canot, and ratings 4.5, 5, 4, 5, 4, 5, 3, 5, 4.5, 3.5 for Hewlett-Backward, calculate ranking by mean rating and the Bayesian ranking.

5.2 *Averaging a crowd* ⋆

Suppose the estimation errors ϵ_i are independent and identically distributed

random variables that takes values 1 and -1 with equal probability. X is a uniform random variable that takes on 100 possible values ($P(X = x) = \frac{1}{100}$, $x = 1, 2, \ldots, 100$). Calculate E_{AE} and E_{EA} for $N = 100$. Does the relationship between E_{AE} and E_{EA} hold?

Let $N = 1000$ and let X take on 1000 possible values uniformly. Plot the histogram of $\frac{1}{N}\sum_{i=1}^{N}\epsilon_i(x)$ over all trials of X. What kind of distribution is the histogram? What is the variance? How does this relate to E_{AE}?

5.3 *Averaging a dependent crowd* ⋆⋆

Consider 3 people making dependent estimates of a number, with the following expectations of errors and correlations of errors:

$$\begin{aligned}\mathbf{E}[\epsilon_1^2] &= 1773\\ \mathbf{E}[\epsilon_2^2] &= 645\\ \mathbf{E}[\epsilon_3^2] &= 1796\\ \mathbf{E}[\epsilon_1\epsilon_2] &= 1057\\ \mathbf{E}[\epsilon_1\epsilon_3] &= 970\\ \mathbf{E}[\epsilon_2\epsilon_3] &= 708\end{aligned}$$

Compute the average of errors and the error of the average in this case.

5.4 *Independence of random variables* ⋆⋆

Many types of logical relationships can be visualized using a **directed acyclic graph**. It is a graphical model with directed links and no cycles (a path that ends at the same node where it starts). Here is an important special case. A **belief network** visualizes a distribution of the following form:

$$p(x_1, x_2, \ldots, x_n) = \prod_{i=1}^{n} p(x_i|Parent(x_i)),$$

where $Parent(x)$ is the set of parental (conditoining set of) random variables of random variable x. For example, we can write a generic joint distribution among three random variables as

$$p(x_1, x_2, x_3) = p(x_3|x_1, x_2)p(x_2|x_1)p(x_1).$$

We can now represent dependence among random variables in a directed acyclic graph. Each node i in this graph corresponds to a factor $p(x_i|Parent(x_i))$. A (directed) link in this graph represent a parental relationship. Look at the graph in Figure 5.6. Is (a, c) independent of e?

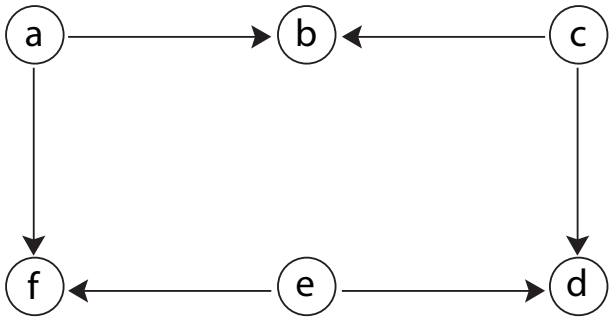


Figure 5.6 A belief network with six random variables. Their dependence relationships are represented by directed links.

5.5 *Analyzing Amazon data* ⋆⋆⋆

Given the rating data in worksheet "raw" of `amazon_data.xls` for the iPod touch, address the following:

(a) Compute the mean and adjusted mean. Here, we define the adjusted mean as $\sum$ (rating $\times$ number of helpful of reviews) / total number of helpful reviews.
(b) Plot the monthly mean.
(c) How does the pattern of the monthly mean inform your perception of the product quality?
(d) Which metric (raw mean, adjusted mean, or monthly mean) is the most accurate in your opinion? Why? Is a certain combination of the metrics more useful?

6 Why does Wikipedia even work?

Now we from the recommendation problem to three chapters on influence in social networks. We start with forming consensus from conflicting opinions in this chapter before moving onto a collection of influence models in the next two.

Let us compare the four different consensus models we have covered in Chapters 3-6, as visualized in Figure 6.1:

- Google pagerank turns a graph of webpage connections into a single ranked order list according to their pagerank scores.
- Netflix recommendation turns a user-movie rating matrix into many ranked order lists, one list per user based on the predicted movie ratings for that user.
- Amazon rating aggregation turns a vector of rating scores into a single scalar for each product.
- Voting systems turn a set of ranked order lists into a single ranked order list, as we will see in this chapter.

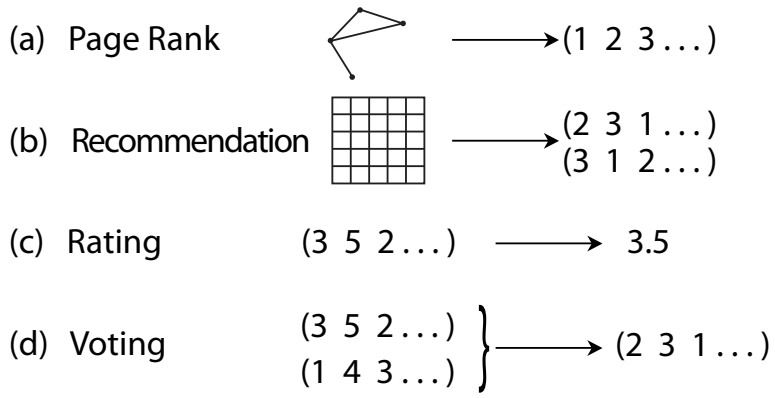


Figure 6.1 Comparison of the four types of consensus formation mechanisms with their inputs and outputs. The first three mechanisms have been covered in the last three chapters, and the last one will be part of this chapter.

6.1 A Short Answer

Crowdsourcing knowledge representation with unpaid and possibly anonymous contributors is very tricky. It faces many challenges for this idea to "work," including two major ones: how to create incentives for people to keep contributing, and how to handle disagreements among the contributors.

Launched in 2001, Wikipedia represented a convergence of three forces that had been gathering momentum: wikis for online collaboration among people, the free and open software movement, and the appearance of online encyclopedias. Within a decade, Wikipedia has generated 3.7 million articles in the US and 19 million articles worldwide. It has become one of the most popular sources of information online. For certain fields, like medicine, the quality of Wikipedia articles are consistently high. And for many fields, if you google a term, a Wikipedia entry will likely come up in the top few search results. It is quite amazing that Wikipedia actually "worked" as much as it did. As we will see in Chapter 11, when people interact with each other, there is often the risk of *tragedy of the commons*. How does Wikipedia turn that into effective collaboration?

Of course, there are also *limitations* to Wikipedia in its capacity as an encyclopedia:

- *Misinformation*: sometimes information on Wikipedia is plainly wrong, especially in articles with a small audience. But Wikipedia provides an effective self-correcting mechanism: it is open to edits from anyone.
- *Mistakes*: there are also honest mistakes, but again, anyone can edit an article, and the edit will stay there as long as no other contributor can present a stronger case otherwise.
- *Missing information*: no encyclopedia can be truly *complete* to everyone's liking, not even the largest encyclopedia in history.

There have been some high profile cases of limitation and abuse. Still, Wikipedia stands as a major success of online collaboration.

There had been other efforts aimed at creating free, online, open encyclopedia before, and the success of Wikipedia is often attributed to a "good faith collaboration" environment within the Wikipedia contributor community. If we count the number of pairwise links in a fully connected graph with n nodes, we have on the order of n^2 such links. But if we examine the number of opinion configurations, we have 2^n possibilities if each person has 2 choices. This n^2 vs. 2^n tension examplifies the positive and the negative sides of the **network effect**. Converging on one of these configurations is difficult, and Wikipedia mostly follows the principle of "rough consensus". The process of reaching a rough consensus can be understood from voting theory (even though it does not explicitly involve voting by providing a ranked order list) and from bargaining theory.

Wikipedia is free, open, dynamic, interactive, and extensively linked. There are natural pros and cons associated with such a model of encyclopedia that

complements other forms of encyclopedia. Let us consider three distinct features of Wikipedia:

- It is free. How can people be motivated to contribute? Incentives do not have to be financial; the ability to influence others is a reward in its own right to most people. This requires the Wikipedia audience to be very large.
- Anyone can write or add to an article, including non-experts, anonymous writers, and people with conflicts of interest. The key is *check and balance*. Precisely because anyone can contribute, Wikipedia has a large body of writers who check others' writing frequently through a mechanism for debates and updates. Sometimes, however, a contributor or an IP address may be blocked if it is detected as a regular source of deliberate misinformation.
- Any subject may be contributed, including controversial ones. Sometimes, however, certain articles can be "protected" from too frequent edits to give time for the community of contributors to debate. How does Wikipedia avoid unbalanced treatment or trivial subjects? It turns out that there are Policies and Guidelines, and there are conflict resolution by editors.

The first and second features above provide Wikipedia with a strong, positive networking effect: a larger audience leads to more contributors, which in turn leads to more audience, provided that the quality of contributions are kept high. This brings us to the third feature above.

In general, how does Wikipedia enforce quality and resolve conflicting contributions? In addressing this question, we also bear in mind the obvious fact that Wikipedia is not a sovereign state with the power of a government. So issues such as voting, decisioning, and free speech do not have the same context.

To start with, there are three core Policies on Wikipedia to help ensure reliability and neutrality as much as possible:

- *Verifiability (V)*: each key point or data in an article must be externally verifiable, with link to the primary source for verification by readers.
- *No Original Research (NOR)*: this is to prevent people from using Wikipedia as a publication venue of their new results.
- *Neutral Point of View (NPOV)*: the basic rule is that a reader must not be able to tell the bias of the author in reading through a Wikipedia article. It is particularly important for controversial topics, but also the most difficult to use exactly in those cases, *e.g.*, contentious political, social, and religious topics. Unlike the above two policies, it is harder to enforce this one since "neutrality" is subjective.

Wikipedia also installs several mechanisms for debates and updates. One is the use of the history page and the talk page, available for public view through tags on top of each article's page. All previous versions and all the changes made are recorded too.

There is also a reputation system for contributors, similar to the reviewer rating system in Amazon. Each article can be rated on a 1-6 grade scale. For

those who do not reveal their names, it is a reputation system of IP addresses of the devices from which contributions are sent. In addition, links across article pages are analyzed similar to Google pagerank's reputation analysis for ranking pages in each search.

But perhaps the ultimate mechanism still boils down to people negotiating. Depending on the stage of the article, expert and non-expert contributors may join the discussion. There is a hierarchy of Wikipedia communities, and debates among contributors who cannot come to an agreement will be put forward to a group of the most experienced editors. This committee of editors acts like a jury, listening to the various sides of the debate, and then tries to decide by "rough consensus."

How do we model the process of reaching a "rough consensus" through "goodwill collaboration?" Not easy. We will see in this chapter two underlying theories: bargaining theory and voting theory. But much translation is needed to connect either of these theories to the actual practice of Wikipedia.

- In a bargaining model, the the contributors need to reach a compromise, otherwise there would be no agreement. Each contributor's utility function, and the default position in the case of no agreement, need to reflect the goodwill typically observed in Wikipedia collaboration.
- In a voting model, each contributor has some partially ordered list of preferences, and a *threshold* on how far the group decision can be away from her own preferences before she vetoes the group decision and thereby preventing the consensus from being reached. (Sometimes a decision is actually carried out by explicit votes in the arbitration committee. And the committee members are also elected through a voting system.) Dealing with partial ordering, quantifying the distance between two ordered lists, and modeling the veto decision threshold are still much under-explored in the study of group interaction dynamics.

In contrast to coordination through well-defined pricing feedback signals that we will see in several later chapters, coordination though bargaining or voting is much harder to model. In the next section, we will present the basics of the rich theory of voting and social choice. Then in Advanced Material, we will briefly discuss the mathematical language for bargaining and cooperative games.

6.2 A Long Answer

Wikipedia consensus formation illustrates important issues in the general case of reaching consensus among a group of individuals that is binding for everyone, *i.e.*, how to sign a social contract with a binding force. This is different from presenting the rank ordered list for each individual to evaluate individually (like Netflix recommendation). It is different from coordinating individual actions through

pricing (like web auction and Internet congestion control). Ranking preferences is also different from ranking objects (like webpages or products).

Voting is obviously essential for elections of those who will make and execute binding laws to be imposed on those casting the votes. It is also useful for many other contexts, from talent competitions to jury decisions. It is a key component of the **social choice theory**, and how individual preferences are collected and summarized.

Voting theory studies how to aggregate vectors, where the vectors are collections of individual preferences. We will see an axiomatic treatment of voting methods in this section. Later in Chapter 20, we will see another axiomatic treatment of scalarizing a vector of resource allocations. In both cases, turning a not-well-ordered input into an ordered output must satisfy certain intuitive properties, and it is often tricky to accomplish that.

6.2.1 Major types of voting

A **voting system** is a function that maps a set of voters' preferences, called a **profile**, to an **outcome**. There are N voters and M candidates. A profile in this chapter is a collection of ranked order lists, one list per voter that lists all the candidates. An outcome is a single ranked order list.

The requirement of complete ranked order lists as inputs can be too stringent. When there are many candidates, often a coarser granularity is used. For example, Wikipedia's arbitration committee members are elected by a binary voting system. Each voter divide the list of candidates into just two parts: those she votes "for" and those she votes "against". Then the percentage of "for" votes, out of all the votes received by each candidate, is calculated to rank order the candidates. The tradeoff between user-interface simplicity and voting result's consistency and completeness is something interesting to explore but does not have as many results as voting systems with complete ranked order input lists.

A voting system *aggregates* N lists into 1 list, like squeezing a 3 dimensional ball into a 2 dimensional "paste". Naturally, some information in the original set of N lists will be lost after this mapping, and that sometimes leads to results not conforming to our (flawed) intuition. We will focus on three commonly used voting systems to illustrate key points.

Perhaps the most popular voting system is **plurality voting**. We simply count the number of voters who put a candidate j in the first position in their lists. Call these numbers $\{V_j\}$, and the candidate j with the largest V_j wins and is put on the first position of the list in the outcome. Put the candidate with the second largest V_j in the second position, and so on. To simplify the discussion, we will assume no ties. There is also a variant called the **Kemeny rule**, where we count how many "least-liked" votes a candidate receives, and rank in reverse order of those numbers. There are other voting systems that try to determine the least objectionable candidate to help reach consensus. Plurality voting sounds reasonable

and is often practiced, but there are many profiles that lead to counter-intuitive results.

A generalization of plurality voting is **positional voting**. Looking at a voter's list, we assign some numbers to each candidate based on its position in the list. The most famous, and the only "right" positional voting that avoids fundamental dilemmas in voting, is the **Borda count**. It is named after the French mathematician in the 18th century who initiated the scientific study of voting systems. By Borda count, the first position candidate in each list gets $M-1$ points, the second position one gets $M-2$ points, and so on, and the last position one gets 0 point, since being the last one in a list that must be complete carries no information about a voter's preference at all. Then, across all the lists, the candidate are ranked based on their total points across all the voters.

Yet another method is **Condorcet voting**, named after another French mathematician who founded the research field of voting study shortly after Borda. It is an aggregation of binary results from pairwise comparisons. All voting paradoxes must have at least three candidates. When $M = 2$, the result is always clear-cut, since each voter's preference can be characterized by one number, and aggregating scalars is unambiguous. So how about we look at each possible pair of candidates (A, B), and see how many voters think one is better than the other? This unambiguously decides the winner out of that pair: if more voters think A is better than B, denoted as A>B, then A is placed higher than B in the aggregated ranked order list. Now, if the pairwise decisions generate a consistent ranked order list, we are done. But it may not be able to, for example, when the three individual input lists for three candidates are: A>B>C, B>C>A, and C>A>B. We can see that pairwise comparison between A and B is A>B, similarly B>C and C>A. But that is *cyclic*, and thus logically inconsistent. There is no Condorcet voting output that is self consistent in this case.

6.2.2 A counter-intuitive example

Suppose the editors of Wikipedia need to vote on a contentious line in an article on "ice cream" about which flavor is the best: chocolate (C), vanilla (V), or strawberry (S); with $M = 3$ candidates and $N = 9$ voters. There are 6 positional possibilities for 3 candidates, and it turns out that half of these receive zero votes, while the other three possibilities of ranked-order receive the following votes:

- C V S: 4 votes
- S V C: 3 votes
- V S C: 2 votes

What should the aggregated ranked order list look like?

By plurality vote, the aggregation is clear: [C S V]. But something does not sound right. Here, those who favor strawberry over chocolate outnumbers those who favor chocolate over strawberry. So how could [C S V] be right?

Well, let us look at Condorcet voting then.

- C or V? Pairwise comparison shows V wins.
- S or V? Pairwise comparison shows V wins again.
- C or S? Pairwise comparison shows S wins.

The above 3 pairwise comparisons are all we need in this case, and aggregation is clear: V wins over both C and S, so it should come out first. Then S wins over C. So the outcome is clear: [V S C]. But wait, this is *exactly* the opposite of the plurality vote's outcome.

How about the Borda count? V gets 11 points, C 8 points, and S 8 points too. So V winns and C and S tie.

Three voting methods gave three different results. Weird. You may object: "But this profile input is synthesized artificially. Real world ones will not be like this."

Well, first of all, this synthesized input is indeed designed to highlight that our intuition of what constitutes an accurate aggregation of individual preferences is incomplete at best.

Second, there are many more paradoxes like this. In fact, we will go through a method that can generate as many paradoxes as we like. This is not an isolated incident.

Third, how do you define "real world cases"? Maybe through some intuitively correct statements that we will take as true to start the logical reasoning process. We call those **axioms**. But some seemingly innocent axioms are simply not compatible with each other. This is the fundamental, negative result of **Arrow's Impossibility Theorem** in social choice theory.

6.2.3 Arrow's impossibility results

Let us look at the following five statements that sound very reasonable about any voting system, *i.e.*, axioms that any voting system must satisfy. Two of them concern a basic property called **transitivity**, a logical self-consistency requirement: if there are three candidates (A, B, C), and in a list A is more preferred than B, and B is more preferred than C, then A is more preferred than C. Symbolically, we can write this as A>B>C $\Rightarrow$ A>C. Had this not been true, we would have a cyclic, thus inconsistent preference: A>B>C>A.

Now the five axioms proposed by Arrow:

1. Each input list (in the profile) is complete and transitive.
2. The output list (in the outcome) is complete and transitive.
3. The output list cannot just be the same as one input list no matter what the other input lists are.
4. (Pareto Principle) If all input lists prefer candidate A over candidate B, the output list must do so too.
5. (IIA Principle) If between a given pair of candidates (A,B), each input list's preference does not change, then even if their preferences involving other

candidates change, the output list's preference based on A and B does not change.

The last statement above is called **Independence of Irrelevant Alternatives** (IIA), or pairwise independence. As we will see, these alternatives are actually not irrelevant after all.

You might think there should be a lot of voting systems that satisfy all the five axioms above. Actually, as soon as we have $M = 3$ candidates or more, there are *none*. If the surprise factor is a measure of a fundamental result's elegance, this impossibility theorem by Arrow in his Ph.D. thesis in 1950 is among the most elegant ones we will see.

How could that be? Something is wrong with the axioms. Some of them are not as innocent as they might seem to be at first glance. The first two axioms are about logical consistency, so we have to keep them. The third one is the underlying assumption of social choice, without which aggregation becomes trivial. So it must be either the Pareto axiom or the IIA axiom.

Usually in an axiomatic system, the axiom that takes the longest to describe is the first suspect for undesirable outcomes of the axiomatic system. And IIA looks suspicious. Actually, how A and B compare with each other in the *aggregate output list* should depend on other options, like candidate C's ranking in the *individual input lists*. To assume otherwise actually opens the possibility that transitive inputs can lead to cyclic output. When a voter compares A and B, there may be a C in between or not, and that in turn determines if the outcome is transitive or not. IIA prohibits the voting system from differentiating between those input lists that lead to only transitive outputs and those that may lead to cyclic outputs.

This will be clearly demonstrated in the next section, where we will see that a group of input rank lists, each transitive, can be just the same as a group of input rank lists where some of them are cyclic. Clearly, when an input is cyclic, the output may not be transitive. In the language of axiomatic construction, if axiom 5 can block axiom 2, no wonder this set of axioms is not self-consistent. The negative result is really a positive highlight on the importance of maintaining logical consistency and the flawed intuition in IIA.

In hindsight, Arrow's impossibility theorem states that when it comes to ranking three or more candidates, pairwise comparisons are inadequate. Then the next question naturally is: what additional information do we need? It turns out that scale, rather than just relative order, will lead to a "possibility result".

6.2.4 Possibility results

What is the *true intention* of the voters? Well, the answer is actually obvious: the entire profile *itself* is the true intention. Voting can be no more universally reflective of the "true intent" of voters than two points, say (1,4) and (3,2), on a 2D plane be compared to decide which is bigger.

Voting in political systems, despite the occurrence of counter-intuitive results stemming from our flawed intuition, is a universal right that provides the basic bridge between individuals and the aggregate, an effective means to provide check and balance against absolute power, and the foundation of consent from the governed to the government. No voting system is perfect, but it is better than the alternative of no voting. Moreover, some voting systems *can* achieve a possibility result.

For example, by replacing IIA with the **Intensity form of IIA** (IIIA), there are voting systems that can satisfy all the axioms. What is IIIA? When we write A>B, we now also have to write down the number of other candidates that sit in between A and B, this is the *intensity*. If none, intensity is zero. IIIA then states that, in the outcome list, the ranking of a pair of candidates depends only on the pairwise comparison *and* the intensity.

While the original 5 axioms by Arrow are not compatible, it turns out that the modified set of axioms, with IIA replaced by IIIA, is: the Borda count is a voting system that satisfies all five axioms now. This stems from a key feature of Borda count: the point spread between two adjacent positions in a rank list is the same no matter which two positions we are looking at. In essence, we need to *count* (the gaps between candidates) rather than just *order* (the candidates).

We have not, however, touched on the subject of manipulation, collusion, or strategic thinking, based on information or estimates about others' votes. For example, Borda count can be easily manipulated if people do not vote according to their true rank ordered list. In Chapter 7, we will look at information cascade as a particular example of influencing people's decision.

6.3 Examples

6.3.1 Sen's result

Another fundamental impossibility theorem was developed by Sen in the 1970s. This time, it turns out the following four axioms are incompatible:

1. Each input list is complete and transitive.
2. The output list is complete and transitive.
3. If all input lists prefer candidate A over candidate B, the output list must too.
4. There are at least two decisive voters.

The first three are similar to what are in Arrow's axioms, and the last one concerns a **decisive voter**: a voter who can decide (at least) one pair of candidates' relative ranking for the whole group of voters, *i.e.*, other voters' preferences do not matter for this pairwise comparison.

Just like Arrow's impossibility theorem, the problematic axiom, in this case axiom 4, precludes the insistency of transitivity of the output list. What axiom 4

implies is actually the following: one voter can impose strong negative externality on all other voters. This is illustrated next.

6.3.2 Constructing any counter examples you want

All examples of Sen's result can be constructed following a procedure illustrated in a small example.

Suppose there are $N = 5$ candidates and $M = 3$ voters. Voter 1 is the decisive voter for (A,B) pairwise comparison, voter 2 for (C,D) pair, and voter 3 for (E,A) pair. These will be marked in bold in tables that follow. We will show how decisive voters can preclude transitivity in the output list.

Let us start with a cyclic ranking for every voter: A>B>C>D>E>A, as shown in Table 6.1.

Table 6.1 Step 1 of constructing examples showing inconsistency of Sen's axioms. Each row represents the draft preferences of a voter. The columns contain a subset of the pairwise comparisons between 5 candidates. Pairwise preferences in bold indicate that they come from decisive voters.

	A,B	B,C	C,D	D,E	E,A
1	**A>B**	B>C	C>D	D>E	E>A
2	A>B	B>C	**C>D**	D>E	E>A
3	A>B	B>C	C>D	D>E	**E>A**

We do not want input rankings to be cyclic, so we need to flip the pairwise comparison order at least at one spot for each voter. But we are guaranteed to find, for each voter, two spots where flipping the order above will not change the outcome. Those are exactly the two pairwise comparisons where some other voter is the decisive voter. So flip the order in those two spots and you are guaranteed a transitive ranking by each voter. The resulting output, however, remains cyclic, as shown in Table 6.2.

This example not only demonstrates how easy it is to generate examples illustrating Sen's negative result, but also that each decisive voter is actually destroying the knowledge of the voting system on whether transitivity is still maintained. If a decisive voter ranks A>B, and another voter ranks not just B>A, but also B> k other candidates>A, then we say the decisive voter imposes a k *strong negative externality* to that voter. In cyclic ranking in Sen's system, each voter suffers strong negative externality from some decisive voter.

This example highlights again the importance of keeping track of the *position* of candidates in the *overall* ranking list by each voter, again motivating the Borda count. We simply cannot consolidate ranking lists by extracting some portion of each list in isolation. "Sensible voting" is still possible if we avoid that compression of voters' intentions.

Table 6.2 Step 2 of constructing examples showing the inconsistency of Sen's axioms. Pairwise preferences in quotation marks are those are flipped from the draft version in Table 6.1 to turn input rankings transitive without destroying the cyclic nature of the outcome ranking.

	A,B	B,C	C,D	D,E	E,A
1	**A>B**	B>C	"D>C"	D>E	"A>E"
2	"B>A"	B>C	**C>D**	D>E	"A>E"
3	"B>A"	B>C	"D>C"	D>E	**E>A**
Outcome	A>B	B>C	C>D	D>E	E>A

6.3.3 Connection to prisoner's dilemma

In fact, the prisoner's dilemma we saw back in Chapter 1 is a special case of Sen's negative result, and similar dilemmas can be readily constructed now. Recall that there are four possibilities: both prisoners 1 and 2 do not confess (A), 1 confesses and 2 does not (B), 1 does not confess and 2 does (C), and both confess (D).

Table 6.3 Prisoner's dilemma as a special case of Sen's impossibility result. Think of the two prisoners as two voters, and the four possible outcomes as four candidates. Four pairs of candidates are compared since they are the four individual actions afforded to the two prisoners. Those pairwise preferences that do not matter are marked in "-" since the other prisoner is the decisive voter for that pair. An additional pairwise comparison is obvious: both do not confess (candidate A) is better than both confess (candidate D).

	A,B	B,D	A,D	C,D	A,C
Prisoner 1	**B>A**	-	A>D	**D>C**	-
Prisoner 2	-	**D>B**	A>D	-	**C>A**
Outcome	B>A	D>B	A>D	D>C	C>A

Prisoner 1 is the decisive voter on (A,B) pair and (C,D) pair. Prisoner 2 is the decisive voter on (A,C) pair and (B,D) pair. Together with the obvious consensus A>D, we have an outcome that contains two cyclic rankings: D>B>A>D and A>D>C>A, as shown in Figure 6.2. This presence of cyclic output rankings in the outcome is another angle to understand the rise of the socially-suboptimal equilibrium D as the Nash equilibrium.

6.4 Advanced Material

The second conflict resolution mechanism, in a Wikipedia editorial decision as well as in many other contexts, is **bargaining**. Each bargaining party has a selfish

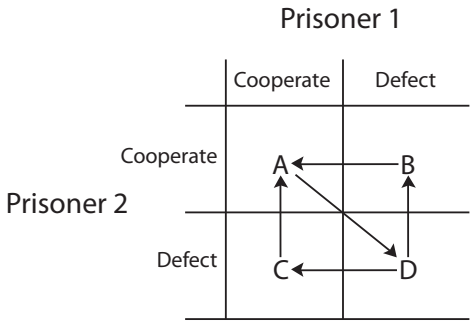


Figure 6.2 Prisoner's dilemma produces two cycles of ranking orders: D>B>A>D and A>D>C>A. This leads to a special case of Sen's impossibility result.

motivation and yet all the parties want to achieve some kind of agreement. If no agreement is achieved, then each party goes back to its own **disagreement point**. This interaction is studied in **cooperative game theory**.

We will first follow the more intuitive approach by Rubenstein in the 1980s before turning to Nash's axiomatic approach in his Ph.D. thesis in 1950, almost at the same time as Arrow's thesis. We will see an IIA style axiom too. But this time it is a possibility theorem that followed, in fact, a unique function modeling bargaining that satisfies all the axioms.

6.4.1 Bargaining: Interactive offers

Suppose there are two people, A and B, bargaining over how to divide a cake of size 1. This cake-cutting problem will be picked up again in Chapter 20 in our study of fairness, with extensions like "one cuts, the other chooses."

For now, consider the following procedure. At the end of each of the discrete timeslots with duration T, each person takes a turn to offer the other person how to share the cake. It is essentially a number $x_1 \in [0, 1]$ for A and $x_2 = 1 - x_1$ for B. This iterative procedure starts with the initial offer at time 0 from A to B. If the offer is accepted, an agreement is reached. If it is rejected, the other person makes another offer at the next timeslot.

But wait a minute. This bargaining process can go on forever. Why would either person be motivated to accept an offer (other than one that gives her the whole cake)? There must be a price to pay for disagreeing. In Rubenstein's model, the price to pay is *time*. If an agreement is reached at the kth iteration, a person's payoff is

$$u_i = x_i e^{-r_i k T}, \quad i = 1, 2,$$

where r_i is a positive number capturing "the tolerance to wait and keep bargaining." So the payoff depends on both the deal itself (x_i) and when you seal the deal (k), with the second dependence sensitive to each person's bargaining power: the one with a larger r_i has more to lose by hanging on to keep bargaining for and rejecting offers.

We will not go into the details of the equilibrium properties of this procedure. But it is intuitively clear that if waiting for the next round of negotiation gives me the same payoff as accepting this round's offer, I might as well accept the offer. Indeed, it can be shown that the equilibrium offers (x_1^*, x_2^*) from each side satisfy the following two equations simultaneously:

$$1 - x_1^* = x_2^* e^{-r_2 T}$$
$$1 - x_2^* = x_1^* e^{-r_1 T}.$$

There is a unique solution to the above pair of equations:

$$x_1^* = \frac{1 - e^{-r_2 T}}{1 - e^{-(r_1 + r_2)T}}$$
$$x_2^* = \frac{1 - e^{-r_1 T}}{1 - e^{-(r_1 + r_2)T}}.$$

As the bargaining rounds get more efficient, *i.e.*, $T \to 0$, the exponential penalty of disagreement becomes linear: $\exp(-r_i T) \to 1 - r_i T$, and the solution simplifies to the following approximation for small T:

$$x_1^* = \frac{r_2}{r_1 + r_2}$$
$$x_2^* = \frac{r_1}{r_1 + r_2}.$$

This proportional allocation makes sense: a bigger r_2 means a weaker hand of B, thus a bigger share to A at equilibrium.

6.4.2 Bargaining: Nash bargaining solution

Iterative bargaining is just one mechanism of bargaining. The model can be made agnostic to the mechanism chosen.

Let the payoff function U_i map from the space of allocation $[0, 1]$ to some real number. Assume these are "nice functions:" strictly increasing and concave. If no agreement is reached, a disagreement point will be in effect: (d_1, d_2). Assume disagreement is at least as attractive as accepting the worst agreement: $d_i \geq U_i(0)$, $\forall i$.

The set of possible agreement points is

$$\mathcal{X} = \{(x_1, x_2) : x_1 \in [0, 1], x_2 = 1 - x_1\}.$$

The set of possible utility pairs is

$$\mathcal{U} = \{(u_1, u_2) : \text{there is some } (x_1, x_2) \in \mathcal{X} \text{ such that } U_1(x_1) = u_1, U_2(x_2) = u_2\}.$$

Nash showed that the following four reasonable statements about a payoff point $\mathbf{u}^* = (u_1^*, u_2^*)$ can be taken as axioms that lead to a unique and useful solution:

1. (Symmetry) If two players, A and B, are identical ($d_1 = d_2$ and U_1 is the same as U_2), the payoffs received are the same too: $u_1^* = u_2^*$.
2. (Affine Invariance) If utility functions or disagreement points are scaled and shifted, the resulting $\mathbf{u}^*$ is scaled and shifted in the same way.
3. (Pareto Efficiency) There cannot be a strictly better payoff pair than $\mathbf{u}^*$.
4. (IIA) Suppose A and B agree on a point $\mathbf{x}^*$ that lead to $\mathbf{u}^*$ in $\mathcal{U}$. Then if in a new bargaining problem, the set of possible utility pairs is a strict subset of $\mathcal{U}$, and $\mathbf{u}^*$ is still in this subset, then the new bargaining's payoffs remain the same.

The first axiom on symmetry is most intuitive. The second one on affine invariance says changing your unit of payoff accounting should not change the bargaining result. The third one on Pareto effiency prevents clearly inferior allocation. The fourth one on IIA is again the most controversial one. But at least it does not preclude other axioms here, and indeed the above set of axioms are consistent.

Nash proved that there is one and only one solution that satisfies the above axioms, and it is the solution to the following maximization problem, which maximizes the product of the gains (over the disagreement point) by both A and B, over the set of payoffs that is feasible and no worse than the disagreement point itself:

$$\begin{array}{ll} \text{maximize} & (u_1 - d_1)(u_2 - d_2) \\ \text{subject to} & (u_1, u_2) \in \mathcal{U} \\ & u_1 \geq d_1 \\ & u_2 \geq d_2 \\ \text{variables} & u_1, u_2 \end{array} \tag{6.1}$$

This solution (u_1^*, u_2^*) is called the **Nash Bargaining Solution** (NBS).

Obviously there is a tradeoff between possible u_1 and possible u_2. If A gets payoff u_1, what is the payoff for B? Using the above definitions, it is:

$$u_2 = g(u_1) = U_2(1 - U_1^{-1}(u_1)).$$

We just defined a mapping, g, from player 1's payoff value to player 2's. This allows us to draw a g as a curve in the payoff plane. It is a similar visualization to the SIR feasibility region in Chapter 1.

If this function g is differentiable, we can differentiate the objective

$$(u_1 - d_1)(g(u_1) - d_2)$$

with respect to u_1, set it to zero, and obtain:

$$(u_1 - d_1)g'(u_1) + (g(u_1), d_2) = 0.$$

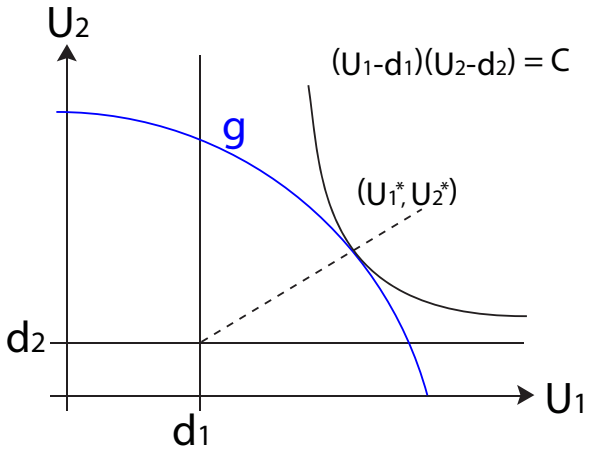


Figure 6.3 Nash bargaining solution illustrated on the two-player payoff plane. It is the intersection of the g curve, which captures the feasible tradeoff between the two players' payoffs, and the straight line normal to g and originating from the disagreement point (d_1, d_2).

Since $g(u_1) = u_2$, we have

$$g'(u_1) = -\frac{u_2 - d_2}{u_1 - d_1}.$$

This has a geometric interpretation, as illustrated in Figure 6.3. NBS (u_1^*, u_2^*) is the unique point on the graph of g where the line from the disagreement point (d_1, d_2) intersects g perpendicular to the slope of g. This clearly illustrates that the bigger the ratio d_1/d_2, *i.e.*, the bigger A's bargaining power relative to B's, the more favorable will u_1 be relative to u_2 at NBS.

For example, if both payoff functions are linear (which actually violates the strictly increasing property we assumed of U_i), then the cake cutting NBS has a simple solution: each person first takes the disagreement point allocation away, and then evenly splits the rest of the cake.

$$x_1^* = d_1 + 0.5(1 - d_1 - d_2)$$
$$x_2^* = d_2 + 0.5(1 - d_1 - d_2).$$

There is another way to model bargaining power: turn the objective function to

$$(u_1 - d_1)^\theta (u_2 - d_2)^{1-\theta},$$

where θ and $1 - \theta$, for $\theta \in [0, 1]$, are the normalized bargaining power exponents for A and B, respectively. There is an interesting connection between the iterative bargaining solution and this axiomatically developed NBS. In the limit of $T \to 0$, the iterative bargaining solution is the same as the NBS solution with asymmetric

bargaining power ($\theta = r_2/r_1 + r_2, 1 - \theta = r_1/r_1 + r_2$) and disagreement point $(0, 0)$.

Further Reading

There is very little mature work on mathematically modeling Wikipedia, but a rich research literature on both voting theory and bargaining theory.

1. A comprehensive survey of the features in Wikipedia, including the policies, guidelines, and editorial procedure, can be found in the following book.

[AMY08] P. Ayers, C. Matthews, and B. Yates, *How Wikipedia Works*, No Starch Press, 2008.

2. Arrow's impossibility theorem was part of Arrow's Ph.D. dissertation, and originally published in 1950 and then in his book in 1951:

[Arr51] K. Arrow, *Social Choice and Individual Values*, Yale University Press, 1951.

3. One of the foremost researchers in voting theory today is Saari, who published several books interpreting and overcoming the negative results of Arrow and of Sen. Our treatment of IIIA, construction of examples of Sen's results, and connection to Prisoner's dilemma, all follow Saari's books. The following one is an accessible and rigorous survey:

[Saa06] D. Saari, *Dethroning Dictators, Demystifying Voting Paradoxes*, 2006.

4. Nash bargaining solution was axiomatically constructed as part of Nash's Ph.D. dissertation, and originally published in the following paper in 1950:

[Nas50] J. Nash, "The bargaining problem," *Econometrica*, vo. 18, pp. 155-162, 1950.

5. Among the many books devoted to bargaining theory since then is a concise and rigorous survey in the following book:

[Mut99] A. Muthoo, *Bargaining Theory with Applications*, Cambridge University Press, 1999.

Problems

6.1 *Differences between Borda count, Condorcet voting and plurality voting* ⋆

Consider an election which consists of 31 voters and 3 candidates A, B, C with the profile summarized as follows:

list	voters
$C > A > B$	9
$A > B > C$	8
$B > C > A$	7
$B > A > C$	5
$C > B > A$	2
$A > C > B$	0

What is the voting result by (a) Plurality voting (b) Condorcet voting (c) Borda count?

6.2 *List's list* ⋆

A three-member faculty committee must determine whether a student should be advanced to Ph.D candidacy or not by the student's performance on both the oral and written exams. The following table summarizes the evaluation result of each faculty member:

Professor	Written	Oral
A	Pass	Pass
B	Fail	Pass
C	Pass	Fail

(a) Suppose the student's advancement is determined by a majority vote of all the faculty members, and a professor will agree on the advancement if and only if the student passes both the oral and written exams. Will the committee agree on advancement?

(b) Suppose the student's advancement is determined by whether she passes both the oral and written exams. Whether the student passes an exam or not is determined by a majority vote of the faculty members. Will the committee agree on advancement?

6.3 *Anscombe's paradox* ⋆⋆

Suppose there are three issues where a "yes" or "no" vote indicates a voter's support or disapproval. There are two coalitions of voters, the majority coalitions $\mathcal{A} = \{A_1, A_2, A_3\}$ and the minority coalitions $\mathcal{B} = \{B_1, B_2\}$. The profile is summarized as follows:

Voter	Issue 1	Issue 2	Issue 3
A_1	Yes	Yes	No
A_2	No	Yes	Yes
A_3	Yes	No	Yes
B_1	No	No	No
B_2	No	No	No

(a) What is the majority voting result of each issue?

(b) For each member in the majority coalition A, how many issues out of three does she agree with the voting result?

(c) Repeat (b) for the minority coalition B.

(d) Suppose the leader in A enforces "party discipline" on all members, namely, members in coalition A first vote internally to achieve agreement. Then on the final vote where B is present, all members in A will vote based on their internal agreement. What happens then to the final voting result?

6.4 *Nash Bargaining Solution* ⋆⋆

Alice (Bargainer A) has an alarm clock (good A_1) and an apple (good A_2); Bob (Bargainer B) has a bat (good B_1), a ball (good B_2), and a box (good B_3). Their utilities for these goods are summarized as follows:

Owner	Goods	Utility to Alice	Utility to Bob
Alice	Alarm Clock (A_1)	2	4
Alice	Apple (A_2)	2	2
Bob	Bat (B_1)	6	3
Bob	Ball (B_2)	2	1
Bob	Box (B_3)	4	2

What is the Nash bargaining result between Alice and Bob?

6.5 *Wikipedia (Open-ended question)*

Take a look at the History pages and Discussion pages of two Wikipedia articles: "Abortion", and "Pythagorean Theorem". Summarize 3 key (qualitative) differences you can see between them.

7 How do I viralize a YouTube video and tip a Groupon deal?

A quick recap of where we have been so far in the space of online services and web 2.0. In Chapter 3, we discussed recommendation of webpages with objective metrics computed by Google from the graph of hyperlinked webpages. In Chapter 4, we discussed recommendation of movies with subjective opinions estimated by Netflix from movie-user bipartite graphs.

Then we investigated the wisdom of crowds. In Chapter 5, we discussed *aggregation* of opinion in (more or less) independent ratings on Amazon. In Chapter 6, we discussed centralized *resolution* of opinion conflicts in Wikipedia.

In this chapter, we will talk about *dependence* of opinions, taking a macroscopic, topology-agnostic approach, and focusing on the viral effect in YouTube and tipping Groupon. Then in the next chapter, we will talk about the effect of network topology on the dependence of opinion.

As will be further illustrated in this and the next chapters, networking effects can be positive or negative. They can also be studied as externalities (coupling in the objective function or the constraint functions), or as information dependence.

7.1 A Short Answer

7.1.1 Viralization

YouTube is a "viral" phenomenon itself. In the space of user-generated video content, it has become the dominant market leader, exhibiting the "winner takes all" phenomenon. More recently it has also featured movies for purchase or rental, and commissioned professional content, to compete against Apple's iTunes and the studios.

YouTube started in February 2005 and was acquired by Google in 2006. Within several years people watched videos on YouTube so much that it became the second largest search engine with 2.6 billion searches in August 2008, even though we normally would not think of YouTube as a search engine. Its short video clip format, coupled with its recommendation page, is particularly addictive. In summer 2011, more than 40% of Internet videos were watched on YouTube, with over 100 million unique viewers each month just in the U.S. Each day over a billion video plays are played, and each minute more than 24 hours of new video clips are uploaded.

There are interesting analytic engines like "YouTube Insight" that highlight the aggregate behavior of YouTube watching. Some videos have gone viral, the most extreme example being "Charlie bit my finger - again," a less-than-one-minute clip that has generated 370 million views as of August 2011. If you just look at the viewer percentage across all the videos on YouTube, it exhibits the heavy tail distribution that we will discuss in Chapter 10.

There has been a lot of social media and web 2.0 marketing research on how to make your YouTube video go viral, including practical advice on the four main paths that lead a viewer to a YouTube clip:

- Web search
- Subscription to a YouTube channel
- Referral through email or twitter
- Browsing through YouTube recommendation page.

It is also interesting to see that YouTube does not use a pagerank-style algorithm for ranking the video clips, since linking the videos by tags is too noisy. Nor does it use the sophisticated recommendation engine such as Netflix Prize solutions, since viewing data for short clips is too noisy and YouTube videos often have short lifecycles.

Instead, YouTube recommendation simply uses video association through **co-visitation count**: how often each pair of videos is watched together by a viewer over, say, 24 hours. This gives rise to a set of related videos for each video, a link relationship among the videos, and thus a graph with the videos as nodes. From the set of videos in k hops from a given video, together with explicit ratings by users and matching of keywords in the video title, tags, and summary, YouTube then generates a top n recommendation page. It has also been observed that often only those videos with a watch count number similar to, or slightly higher than, that of the current video are shown in the recommendation page. This makes it easier for widely watched videos to become even more widely watched, possibly becoming viral.

Now, how do you even define "viral"? There is no commonly accepted definition, but probably the notion of "viral" means that the rate-of-adoption curve should exhibit three features, like curve (c) shown in Figure 7.1:

- High peak
- Large volume, *i.e.*, the adoption lasts long enough in addition to having a high peak
- A short time to rise to the peak.

7.1.2 Tipping

Another web 2.0 sensation that has gone viral is the daily deal service, such as Groupon and LivingSocial. Groupon was formed in 2008, and after two years was generating over $300 million annual revenue from more than 500 cities.

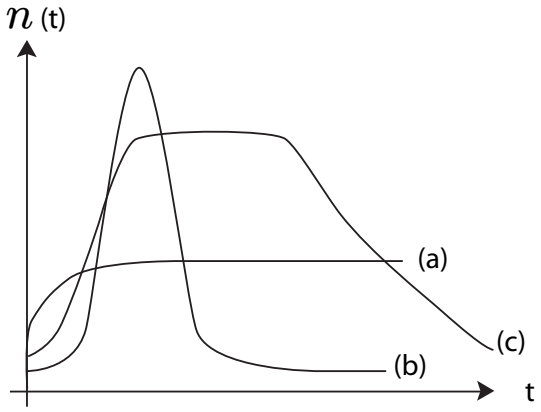


Figure 7.1 A few typical shapes of adoption trajectory $n(t)$ over time t. (a) stays at a low level. (b) rises very quickly but then dies out very quickly too. (c) has a reasonably sharp rise to a high level and a large area under the curve. We can also have combinations of these curves, *e.g.*, a curve with a sharp rise to a high level and stays there.

The effectiveness of Groupon (and the like) for the suppliers and the consumers is still somewhat under-studied. In a detailed study by Dhulakia in 2011, 324 businesses in 23 US markets were surveyed. Some interesting results emerged: close to 80% of deal users were new customers, but only 36% of them spent beyond the deal's face value, and only 20% returned to buy at full price later. On the supplier side, 55% made money from the deals, but 27% lost money, and less than half of them expressed interest to participate in the future, restaurants and salons being particularly negative. Across hundreds of daily deals websites, there are few differentiation factors at this point. It is projected that the cut into the deals' revenues by these websites will have to be lowered in the future as the industry consolidates.

In a daily deal operation, a supplier of some goods or services announces a special discount, which must have a large enough number of users signed up within a 24-hour period. If the number of users exceeds the target threshold, the deal is tipped, and each user has, say, 3 months to redeem the coupon. The supplier's hope is that the discount is in part compensated for by the high volume, and the possibility of repeat customers who will return in the future and pay the full price. This is **the power of crowds** in action. More specifically, a daily deal tipping needs a sufficiently large number of people to make the same decision within a sufficiently small window of time.

This chapter presents models that can be used to characterize and understand phenomena such as viralization, tipping, and synchronization.

7.2 A Long Answer

There are two distinct reasons for popularity of a product:

1. *Intrinsic value*: you may enjoy certain music or buy a particular product just because you like it, whether the rest of the world agrees or not.
2. *Network effect*: your decision depends on what others do, either because (a) the fact that others like a product gives you information, possibly leading to what we call information cascades, an example of the fallacy of crowds, or (b) because the value of the service or product depends on the number of people who use it, like the fax machine or Wikipedia, an example of positive externality.

We will first discuss models of 2(a), such as **information cascade** studied in the political economy literature and **tipping** from an unstable equilibrium towards a stable equilibrium. Then in Advanced Material, we will discuss the combination of 1 and 2(b), called the intrinsic factor and the imitation factor in **diffusion models** studied in the marketing literature, as well as a **synchronization** model.

All the models in this chapter are population-based and agnostic of the actual topologies, although information cascade implicitly assumes a linear topology. In the next chapter, we will focus on topology-based models in the study of influence.

The models in both chapters are summarized in Table 7.1. Except for the synchronization and random walk models, all assume the nodes (the people) have discrete, often binary, states of mind. We will also see that some of these models become similar when generalized a little.

Which model to use really depends on what we are trying to model: many people acting at the same time? Or a few early adopters changing others' minds? Or does one more person carry over the threshold and trigger a change in others' behavior? Each of these models is motivated by a different type of influence, and has its own use and abuse.

Viralizing a YouTube video, tipping a Groupon deal, and influencing via Facebook and Twitter posts are three particular examples in these two chapters. But for these emerging tasks involving human psychology factors, we still do not know much about which of these models and their extensions, if any, fit our tasks sufficiently well to render predictive power.

Across these models, the following issues are raised and some of them addressed:

- Distribution at equilibrium
- Time it takes to reach an equilibrium
- More generally, transient behavior before an equilibrium is reached

We first must observe how individual decisions and local interactions lead to a global property. Then, the general workflow of our modeling process may run as

Table 7.1 Influence Models in Chapters 7 and 8. Tipping and contagion models are highly related, as are information cascade and random walk, and diffusion and infection. Those models we discuss in Long Answers are in italics, while the other models are in Advanced Material. The precise forms of deterministic or random interaction models will be presented as we go through these two chapters.

	Deterministic Interaction Model	Random Interaction Model
Population based	*Information cascade* Synchronization	*Tipping* Diffusion
Topology dependent	*Contagion* Random walk	*Infection*

follows: pick a curve of adoption evolving over time, like one of those in Figure 7.1, reverse engineer it through some mathematical languages (differential equation, dynamic systems, sequential decisioning, selfish optimization, *etc.*), and finally, hope it will also shed light on forward engineering. The last step is where the gap between theory and practice lies. Having said that, let us see what kinds of explanatory models have been created.

7.2.1 Information Cascade

One of the large scale, controlled experiments that resemble the YouTube experience was run by Salganik, Dodds, and Watts in 2005. Each of the 14341 participants rated each of the $S = 48$ (unknown) songs (from unknown bands) from 1 to 5 stars. There were four different conditions tested, depending on

- Whether the songs were presented at random, or in descending order of the current download counts.
- Whether the current download numbers were hidden, or shown next to each song.

When the current download numbers were shown, social influence was present. And when the order of the songs followed the download numbers, this influence became even stronger.

How do we measure the *spread* of the download numbers of the songs? Let m_i be the percentage of downloads, out of all the downloads, received by song i, for $i = 1, 2, \ldots, S$. We can examine the (normalized) sum of the differences: $\sum_{i,j} |m_i - m_j|$, which the experimental data showed to be always larger under social influence than under the independent condition,

In addition, a heavier social influence also increased the *unpredictability* of a song's popularity. Really good songs (as defined by download popularity under no social influence) rarely do badly even under social influence, and really bad ones almost never do very well, but for the majority of the songs in between, stronger social influence significantly increases their range of popularity.

This experiment, together with many other controlled experiments or empirical social data analyses, demonstrate what we often feel intuitively: we are influenced by others' opinions even when we do not know the underlying reasons driving their opinions.

On YouTube, if many others before us watched a video, we are more likely to watch it too. We might abort the viewing if we realize we actually do not like it, but this still counts to the viewing number shown next to the video and partially drives its place on the recommendation page.

On a street corner, if one person looks up to the sky, you may think she has a nose bleed or just turned philosophical, and you would pass by. But if ten people look up to the sky together, you may think there is something wrong up there, and stop and look up to the sky too. Now that makes the crowd even bigger. Until someone shouts: "Hey, these two guys have nose bleeds, that is why they stopped and tilted their heads!" Then you think everyone else, just like you, was misled, and decide to leave the crowd and keep on walking. Suddenly, the whole crowd disperses.

These examples, and many others in our lives, from stock market bubbles and fashion fads to pop stars emerging and totalitarian regimes collapsing, illustrate several key observations about information cascade as a model for influence in **sequential decision making**:

- Each person gets a *private signal* (my nose starts bleeding) and releases a *public action* (let me stop walking and tilt my head to the sky). Subsequent users can only observe the public action but *not* the private signal.
- When there are enough public actions of the same type, at some threshold point, all later users start ignoring their own private signals and simply follow what others are doing. A cascade, or viral trend, starts, and the *independence assumption* behind the wisdom of crowds in Chapter 6 breaks down.
- A cascade can start *easily* if people are ready to rely on others' public actions in their reasoning, it can accumulate to a *large size* through positive feedback, and it can be *wrong*.
- But a cascade is also *fragile*. Even if a few private signals are released to the public, the cascade can quickly disappear or even reverse direction, precisely because people have little faith in what they are doing even when there are many of them doing the same thing.

There are many dimensions to understanding the above observations. We focus in this subsection on a particular model from 1992 for sequential decision making that leads to information cascades. The simplest version of this model is a binary number guessing process.

Consider a set of people lined up in sequential order. Each person takes a turn observing a private signal $X_i = \{0, 1\}$ and trying to guess if the correct number is 0 or 1. To simplify the notation a little, we will assume that the correct number

is equally likely to be 0 or 1, although that does not need to be the case for what we will see.

The chance that the private signal is the correct number is $p_i > 0.5$, and with probability $1 - p_i$ it is not. It is important that $p_i > 0.5$. We will assume for simplicity that all $p_i = p$. In addition, the private signal is conditionally independent of other people's signals, conditioned on the underlying correct number.

Upon receiving her private signal, each user writes down her guess $Y_i = \{0, 1\}$ on a blackboard and walks away. That is her public action. Since it is a binary guess, she assesses that one number is more likely than the other, and writes down her guess accordingly. Every user can see the guesses made by people before her, but not their actual private signals.

If you are the first person, what should you do? If you see 1, you will guess 1, since the chance of that being the correct guess is $p_1 > 0.5$. Similarly, if you see 0, you will guess 0.

If you are the second person, what should you do? You have one private signal X_2 and one public action recorded from the first person Y_1. You know how person 1 *reasoned*: X_1 must be equal to Y_1, even though you cannot see her X_1. So, as the second user, you actually know the first user's private signal: $X_1 = Y_1$. Now equipped with two private signals $\{X_1, X_2\}$, you will decide as follows: if both X_1 and X_2 are 0, then obviously the correct number is more likely to be 0, and you guess 0. If both are 1, then guess 1. If one is 0 and the other is 1, then flip a coin and randomly choose 0 or 1.

Now comes the first chance of an information cascade starting. If you are the third person, what should you do? You have one private signal X_3, and two public actions Y_1, Y_2.

- If $Y_1 \neq Y_2$, by reasoning through the first two persons' reasoning, you know the public actions by prior users collectively do not tell you anything, and you should just rely on your own private signal. In this case, you are in exactly the same shoe as the first user. And the fourth user will be in the same shoe as the second user.
- However, if $Y_1 = Y_2$, and they are both the same as X_3, you will obviously pick that number. But even if your private signal X_3 differs, you still will pick what two public signals suggest. This follows the same *Bayesian reasoning* as in Chapter 5. Say $Y_1 = Y_2 = 1$ and $X_3 = 0$. What is the probability that the correct number is 1 given this sequence of $(1, 1, 0)$? By Bayes' rule, this probability, denoted as $P[1|(1, 1, 0)]$, equals

$$\frac{P[1]P[(1,1,0)|1]}{P[(1,1,0)]} = \frac{P[1]P[(1,1,0)|1]}{P[1]P[(1,1,0)|1] + P[0]P[(1,1,0)|0]}, \tag{7.1}$$

  where $P[1]$ and $P[0]$ denote the probabilities that the correct number is 1 and 0, respectively, $P[(1, 1, 0)|1]$ and $P[(1, 1, 0)|0]$ denote the conditional probabilities that $Y_1 = 1, Y_2 = 1, X_3 = 0$ given that the correct number is 1 and 0, respectively.

Plugging in the numbers into (7.1), we have

$$\frac{0.5(p^2(1-p)+0.5p(1-p)^2)}{0.5(p^2(1-p)+0.5p(1-p)^2)+0.5((1-p)^2p+0.5p^2(1-p))}$$

since the sequence of $Y_1 = 1, Y_2 = 1, X_3 = 0$ could have either come from $X_1 = 1, X_2 = 1, X_3 = 0$, or $X_1 = 1, X_2 = 0, X_3 = 0$ but the second person flips the coin and so happens to choose $Y_2 = 1$. Now, is this expression bigger than half? Certainly. After cancelling $p(1-p)$, we are dividing $2p + (1-p)$ by $2p + (1-p) + 2(1-p) + p$, which is $(1+p)/3$. Since $p > 0.5$, that ratio is indeed bigger than $1/2$. So person 3 guesses that the correct number is 1 even when her private signal is 0.

In summary, once an odd-numbered user and then an even-numbered user show the same public action in a row, the next user will just follow, no matter what her private signal is. An information cascade thus starts.

The probability of no cascade after two people have received their private signals and made their public actions is equal to the probability that the first two private signals are different. If the correct number is 1, the probability that $X_1 = Y_1 = 1, X_2 = 0$ is $p(1-p)$, and with probability $1/2$, user 2 will choose $Y_2 = 0$. So the probability of $Y_1 = 1, Y_2 = 0$ is $p(1-p)/2$. The probability that $Y_1 = 0$ and $X_2 = 1, Y_2 = 1$ is $p(1-p)/2$ too, so the probability of $Y_1 \neq Y_2$, *i.e.*, no cascade, is $p(1-p)$. By symmetry, the answer is the same when the correct number is 0. In conclusion, the probability of no cascade is

$$\text{Prob}_{no} = p(1-p).$$

And by symmetry, the probability of a cascade of 1's (call that an "up" cascade) and that of a cascade of 0's (call that a "down" cascade) are the same, each taking half of $1 - \text{Prob}_{no}$:

$$\text{Prob}_{up} = \text{Prob}_{down} = \frac{1-p(1-p)}{2}.$$

Following a similar argument, we see that after an even number, $2n$, of people have gone through the process, we have:

$$\text{Prob}_{no} = (p(1-p))^n, \tag{7.2}$$

and

$$\text{Prob}_{up} = \text{Prob}_{down} = \frac{1-(p(1-p))^n}{2}, \tag{7.3}$$

since no cascades happen if each pair of people (1,2), (3,4), (5,6)... take different public actions.

Therefore, intuitively and mathematically, we see that cascades eventually will happen as more people participate. And this conclusion has been proven to hold for general cases (under some technical conditions), even with multiple levels of private signals and multiple choices to adopt.

It is more instructive to look at the probability of correct vs. incorrect cascades, rather than the symmetric behavior of a up and down cascades. Of course that depends on the value p: how noisy the private signal is. If the signal is very noisy, $p = 0.5$, by symmetry, correct cascades are as likely to happen as incorrect cascades. But as p approaches 1, the chance of a correct cascade goes up pretty fast and saturates towards 1. This computation and visualization is shown in an example later.

How long will a cascade last? Well, forever, unless there is some kind of disturbance, *e.g.*, a release of private signals. Even a little bit of that often suffices, because despite the number of people in the cascade, they all know they are just following a very small sample. This is the counter part of an easy (and possibly wrong) cascade: the **Emperor's New Clothes effect**. Sometimes it only took one kid's shouting out a private signal to stop the cascade.

How do we break a cascade? Suppose a cascade of 1's has started. Now when one person gets a private signal of 0, she shouts *that* out, instead of her public action, which is still to guess that the correct number is 1. If the next person gets a private signal of 0, she would think that, on the one hand, now there are two private signals of 0s. On the other hand, there may have been only two private signals of 1s since that is enough to kick off a cascade of 1s (or even just one private signal of 1, since the following user receiving a private signal of 0 may break the tie by randomly choosing another 1 as the public action). So she will decide to guess 0, and break the cascade. Cascades only represent what happened with few people right around the time it started. If everyone knows that, another block of a few people may be able to break it.

Sometimes it takes the apparent conformity of more users, *e.g.*, 10 people standing on a street corner looking up at the sky, since for something with a low probability (something is wrong in the sky, wrong enough that I want to stop walking and take a look), you need more public actions to override your private signal. Back to our nosebleeding-created crowd on a street corner, a passer-by often needs more convincing than just guessing the right number with higher than 50% probability. That is why she needs to see a bigger crowd all tilting their heads on the street corner before she is willing to stop and do the same. A more curious and less busy person will need a lower threshold of crowd size for her to stop and do the same.

We have assumed that everyone has the same precision p, but each person i may have a different p_i. Suppose all people know the values of $\{p_i\}$ for everyone. Where should the high precision person be placed in the sequence of people (if you could control that)? If we put the highest precision person first, a cascade may start right after her. An authoritative person starts leading the pack. So if you want to start the cascade, that is the right strategy, and it partially explains why some cascades are difficult to break as private signals keep getting overwhelmed by public actions. But if you want a higher chance of reversing a cascade, you want to put higher precision persons later in the sequence.

More importantly, we have assumed that everyone acts rationally. What each person *should* do can be quite different from what she *actually* does in real life.

There are many implications of this information cascade model, since many social phenomena exhibit the features of (a) a crowd making a decision that ignores each individual's own private information, and yet (b) the chain reaction is fragile. Information cascade is part of the reason why US presidential primary elections use Super Tuesday to avoid sequential voting, why teenagers tend to obtain information from the experiences of peers, and why, once people suspect the underlying true signal has changed (whether it has actually changed or not), the ongoing cascade can quickly reverse.

So far we have focused on a model with essentially a *linear topology* of nodes (each node is a person) and each node's local decision is based on *strategic thinking*. In Advanced Material, we move on to a different model, with a macroscopic view on the effect of peers on each person without the detailed strategic thinking. In the next chapter, we will discuss models where an individual's decision depends on her local topology.

7.2.2 Tipping

But first, a different model that illustrates, in a simple way, the notion of tipping, and of stability of equilibrium. It has to do with the fundamental idea of **positive feedback**, a phenomenon that cuts across many domains, from a microphone generating excessively high volume during tune-up to high unemployment rate lasting longer as unemployment benefits extend. In contrast, in Chapter 14 we will look at also *negative* feedback in the Internet to help provide stability.

Suppose there are two possible states in each person's mind. Each person decides to flip from one state (*e.g.*, not buying an iPad) to the other (*e.g.*, buying an iPad) based on whether her utility function is sufficiently high. That utility function depends on p, the product **adoption percentage** in the rest of the population (or just among her neighbors). In the next chapter, we will go into the details of the best response strategy of a person, who learns that $p\%$ of her neighbors have adopted the product. Right now it suffices to say that we have a probability of a person adopting the product as a function of p. Then taking the average across all users, we have an **influence function** f that maps p at time slot t to p at the next time slot $t+1$.

We can readily visualize the dynamics of p by tracing two curves on the $(p, f(p))$ plane, as shown in Figure 7.2: the $f(p)$ curve in a solid line, and the straight line of 45 degrees in a dotted line from the origin. Given a p value at time t, the future evolution is visualized as the zig zag trajectory bouncing vertically from the 45 degree line to the $f(p)$ curve (carrying out the mapping step $f(p(t))$), and then horizontally back to the 45 degree line as the step after that (carrying out the equality step $p(t+1) = f(p)$). This is graphically showing one iteration of $p(t+1) = f(p(t))$.

When the two curves intersect at a point $(p^*, f(p^*))$, we have an *equilibrium*,

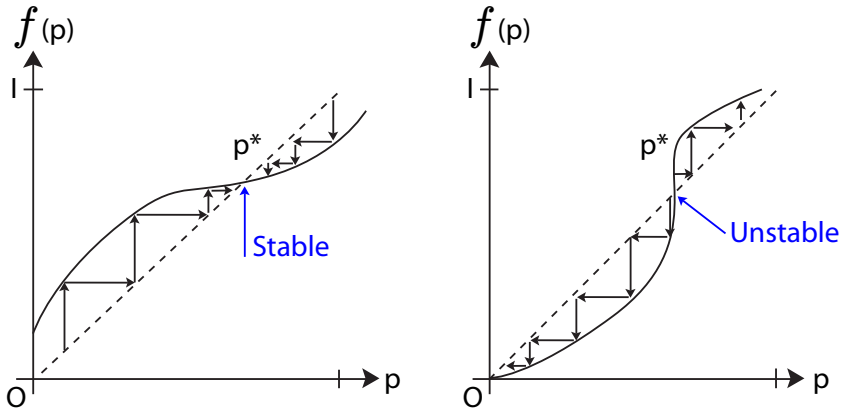


Figure 7.2 Influence function $f(p)$ is plotted on the graph. Starting from a given p value, we can move vertically to hit $f(p)$, and then move horizontally to hit the 45 degree line. That represents one iteration of $p(t+1) = f(p(t))$. Graph (a) illustrates a stable equilibrium, where $p = 0$ eventually climbs to the equilibrium $p = p^*$ despite perturbation around it. Graph (b) illustrates an unstable equilibrium, where perturbation around p^* moves away from it. For $f(p)$ that cuts across the 45 degree line once, if it is concave before the cross-over point and convex afterwards, that cross-over point is stable. If it is convex before and concave afterwards, the cross-over point is unstable.

since if the current $p = p^*$, the next p remains the same. There can be zero, one or many equilibria. The concept of equilibrium here is not the same as strategic equilibrium in a game, or convergence of an iterative algorithm, but reaching a fixed point of an update equation.

Some equilibria are **stable** while others are **unstable**. A stable equilibrium here is one where a perturbation around it will converge back to it. And an equilibrium that is not stable is called unstable. For example, in Figure 7.2, the equilibrium in (a) is stable, as the trajectory around it converges to it, whereas the equilibrium in (b) is unstable. The shape of influence function f matters.

Now we can view tipping as when the adoption percentage p reaches past an unstable equilibrium $(p_1, f(p_1))$, and falls into the **attraction region** of the next stable equilibrium $(p_2, f(p_2))$: as more people adopt it, even more people will adopt it, and this process continues until the adoption percentage is p_2. This is similar to the tipping behavior of popular videos on YouTube. As more people watch a video, its viewer counter increases, and more people think it is worthy of watching. If you would like to make that happen to your video clip, estimating $f(p)$ and then getting into the attraction region of an equilibrium with a high p are the difficult steps of tipping design.

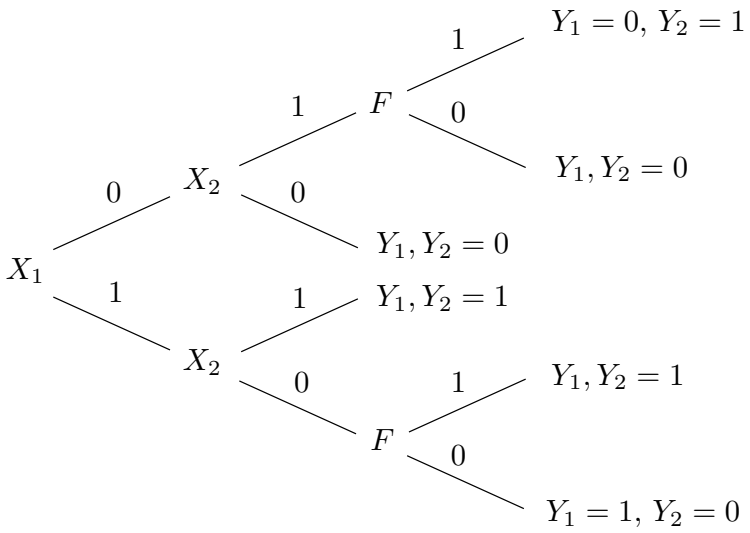


Figure 7.3 An example of information cascade, visualized as a tree. Each branching level is a new person. F denotes flipping a coin to randomly choose what number to guess.

7.3 Examples

7.3.1 Information Cascade

First recall what we discussed in Section 7.2.1 about information cascades through a simple binary number guessing process, and the symmetric behavior of up and down cascades. The example now further derives the probability of correct vs. incorrect cascades. Here we assume the correct number is 1. So an up cascade is a correct one. In contrast, in the last section we did not pre-set the correct number.

Consider the first two people. The different cases of X_1 and X_2 are shown in

Figure 7.3, where F denotes a coin flip by the second user when in doubt. Then

$$\begin{aligned}
\text{Prob}_{no} &= P(X_1 = 0, X_2 = 1, F = 1) + P(X_1 = 1, X_2 = 0, F = 0) \\
&= (1-p) \cdot p \cdot \frac{1}{2} + p \cdot (1-p) \cdot \frac{1}{2} \\
&= p(1-p), \\
\text{Prob}_{up} &= P(X_1 = 1, X_2 = 1) + P(X_1 = 1, X_2 = 0, F = 1) \\
&= p^2 + \frac{p(1-p)}{2} \\
&= \frac{p(1+p)}{2}, \\
\text{Prob}_{down} &= P(X_1 = 0, X_2 = 0) + P(X_1 = 0, X_2 = 1, F = 0) \\
&= (1-p)^2 + \frac{(1-p)p}{2} \\
&= \frac{(1-p)(2-p)}{2}.
\end{aligned}$$

Since the correct number is 1, $\text{Prob}_{correct} = \text{Prob}_{up}$ and $\text{Prob}_{incorrect} = \text{Prob}_{down}$.

Extending to the general case of $2n$ people,

$$\begin{aligned}
\text{Prob}_{no} &= (p(1-p))^n, \\
\text{Prob}_{correct} &= \sum_{i=1}^{n} P(\text{no cascade before } i\text{-th pair}, Y_{2i-1} = 1, Y_{2i} = 1) \\
&= \sum_{i=1}^{n} (p(1-p))^{i-1} \frac{p(p+1)}{2} \\
&= \frac{p(p+1)}{2} \sum_{i=0}^{n-1} (p(1-p))^i \\
&= \frac{p(p+1)}{2} \frac{1-(p(1-p))^n}{1-p(1-p)} \\
&= \frac{p(p+1)[1-(p-p^2)^n]}{2(1-p+p^2)},
\end{aligned}$$

and

$$\begin{aligned}
\text{Prob}_{incorrect} &= \sum_{i=1}^{n} P(\text{no cascade before } i\text{-th pair}, Y_{2i-1} = 0, Y_{2i} = 0) \\
&= \sum_{i=1}^{n} (p(1-p))^{i-1} \frac{(1-p)(2-p)}{2} \\
&= \frac{(1-p)(2-p)[1-(p-p^2)^n]}{2(1-p+p^2)}.
\end{aligned}$$

Figure 7.4 shows the plots of the probabilities as functions of p, for different n. The observations are as follows.

1. Obviously, a larger p (the probability of a private signal being correct) increases $\text{Prob}_{correct}$.
2. Increasing n reduces Prob_{no}. When $n = 1$, Prob_{no} is significant. When n is small, increasing it helps increase $\text{Prob}_{correct}$. But the effect of increasing n quickly saturates. The plots of $n = 5$ and $n = 100$ are almost indistinguishable.
3. Even for large n, $\text{Prob}_{correct}$ is significantly less than 1 for a large range of p. This is somewhat counter-intuitive because when n is large, we expect a large amount of information to be available and that a correct cascade should happen with high probability even for small p. But what happens is that cascades block the aggregation of independent information.

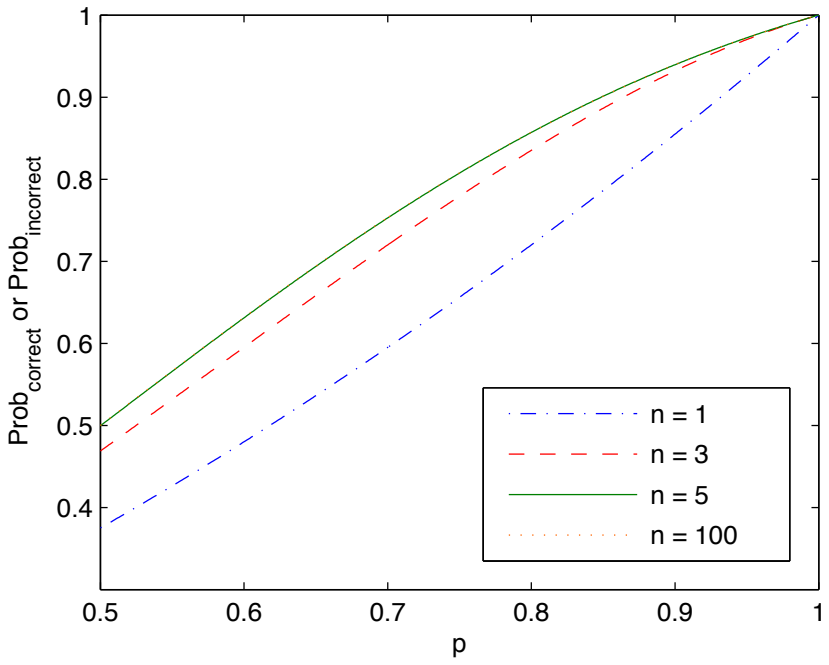


Figure 7.4 Probabilities of information cascade. As the probability p (of each person getting the correct private signal) increases, the probability of forming a correct cascade increases roughly linearly. Once the number of people n reaches 5, the curves are almost identical even as n increases further.

7.3.2 Tipping

Now recall what the influence function f that evolves the product adoption percentage p in Section 7.2.2. The following example illustrates the notion of

tipping. Consider two influence functions:

$$f_1(p) = \frac{1}{1 + e^{-12(p-0.5)}} \tag{7.4}$$

$$f_2(p) = 0.5 + \frac{1}{12}\log\Big(\frac{p}{1-p}\Big). \tag{7.5}$$

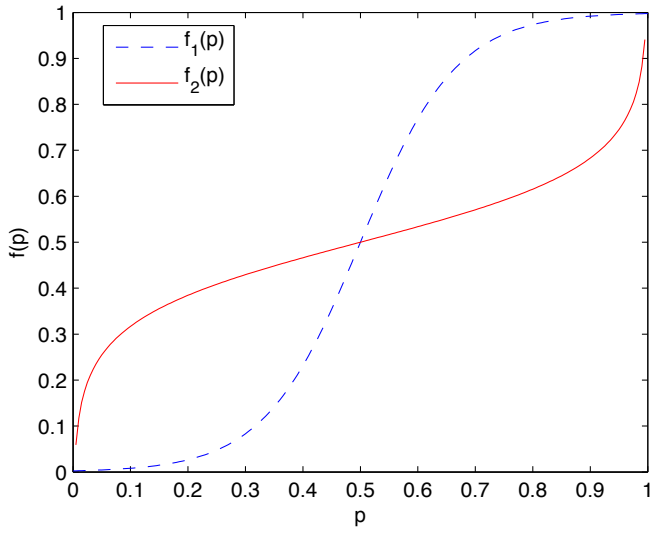


Figure 7.5 A tipping example, with two different reaction functions $f(p)$. One leads to a stable equilibrium at $(0.5, 0.5)$, and the other to an unstable equilibrium at the same point.

Figure 7.5 shows the plots of both functions, each with three equilibrium points $p^* = 0.00255, 0.5, 0.997$, where $f(p^*) = p^*$. For $f_1(p)$ the equilibria 0.00255 and 0.997 are stable, while for $f_2(p)$ the equilibrium 0.5 is stable.

Now consider what happens when we try to tip from an unstable equilibrium. For $f_1(p)$, we start at $p_0 = 0.5 - 0.001 = 0.499$ and iteratively update p_i (where $i = 0, 1, 2, \ldots$) as

$$p_{i+1} = f(p_i),$$

then p_i updates as 0.499, 0.497, 0.491, 0.473, 0.419, 0.276, 0.0639, 0.00531, 0.00264, 0.00255, 0.00255... and p_i eventually stops changing when it reaches the stable equilibrium $p^* = 0.00255$.

For $f_2(p)$, we iterate from $p_0 = 0.00255 + 0.00001 = 0.00256$, and p_i updates as 0.00256, 0.0134, 0.141, 0.350, 0.448, 0.483, 0.494, 0.498, 0.499, 0.500, 0.500... Again p_i stops changing when it reaches the stable equilibrium $p^* = 0.5$.

7.4 Advanced Material

7.4.1 Synchronization

Sometimes herding or tipping manifests on the time dimension too, when many people's behaviors are synchronized. This is one step beyond just converging on the same public action (in information cascade) or on adoption percentage (in tipping). The study of **synchronization** started with clocks. Two clocks hang on the wall next to each other. Vibration propagating through the wall produces a coupling between the two clocks. Soon their ticking will be synchronized. Synchronization happens a lot in nature: from fireflies glowing to crickets chirping, and from people walking to pacemakers adjusting.

The standard model to study synchronization is **coupled oscillators**. We will look at a weak form of coupling called **pulse-coupling**. Later we will also study *decoupling* over a network in Chapter 14, and how to use very little and implicit communication, plus a little randomization, to avoid synchronization in Chapter 18.

We will focus on the case of 2 weakly pulse-coupled oscillators: A and B, and visualize the dynamics in a state-phase diagram in the $[0,1] \times [0,1]$ grid. Each oscillator's state x evolves over time, with magnitude normalized to $[0,1]$. We parameterize this movement not directly as a function of time t, but instead by a phase parameter that keeps track of the asynchronism between A and B. The state x depends on the phase ϕ:

$$x(t) = f(\phi(t)),$$

where $\phi \in [0,1]$ is the phase variable, itself moving over time steadily as $d\phi/dt = 1/T$ for cycle period T, thus driving x over time as well. We can think of $\phi(t)$ as the degree marked on the clock by an arm, and $x(t)$ the state of the spring behind the arm. The boundary conditions are $f(0) = 0$ and $f(1) = 1$, as shown in Figure 7.6.

A simple and important model of $f(\phi)$ is the "pacemaker trajectory:"

$$f_\gamma(\phi) = (1 - e^{-\gamma})(1 - e^{-\gamma\phi}),$$

where γ is a constant in the pacemaker model. Strictly speaking, the upper bound on the state x is now $(1 - e^{-\gamma})^2$ (close to 1 for large γ but not exactly 1), achieved when phase ϕ is 1.

When $x = 1$ for, say, oscillator A, three things happen:

- Oscillator A "fires."
- x_A goes back to 0.
- The other oscillator B gets a "kick:" its own x_B goes up to $x_B(t) + \epsilon$, unless that value exceeds 1, in which case B fires too, and the two oscillators are synchronized from that point onward.

So if we can show that, no matter what the initial condition, the two oscillators A and B will eventually fire together, then we will have shown that synchronization must happen.

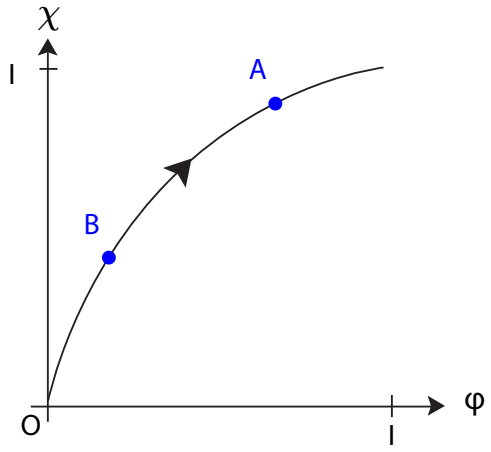


Figure 7.6 Trajectory of two pulse-coupled oscillators: A and B, shown on the state-phase plane. As time passes by, both A and B follow the trajectory f. When either reaches $(1, 1)$, it fires, returns to the origin, and kicks the other vertically by ϵ. Once they fire at the same time, they are synchronized as they will travel together on this plane from that point onwards.

As shown in Figure 7.6, A and B move along curve f with a constant horizontal distance between them until one of them reaches the firing threshold, fires, drops back to the origin, and causes the other one to jump vertically by ϵ.

Before walking through a proof that synchronization will happen, let us first look at a numerical example. Oscillators A and B have initial phases 0.4 and 0.6, respectively. Consider the pacemaker trajectory with $\gamma = 2$, thus the trigger threshold is $(1 - \exp -\gamma)^2 = 0.7476$. The coupling strength is $\epsilon = 0.01$. The two oscillators start out at different trajectories, and as they oscillate, whenever one oscillator "fires" and drops down, the other oscillator's trajectory is "kicked" upwards, increasing by a small quantity $\epsilon = 1/100$. The time of synchronization, $t = 3181$, is the first instance both oscillators achieve threshold in the same time interval, as illustrated by the dotted blue line in Figure 7.7.

Now our reasoning through the general case. Suppose A just fired, and B has phase ϕ. This is an abuse of notation, as we now use ϕ as a specific phase rather than a generic notation of the phase axis. What will be the phase of B after the *next* firing of A? We hope A and B will be closer to being synchronized. Call the answer to this question the image of a **return function** $R(\phi)$.

First, it is clear that after A's firing, B is leading A now. After a phase shift of $1 - \phi$, B fires, when A is at phase $1 - \phi$, and with state $x_A = f(1 - \phi)$. Right

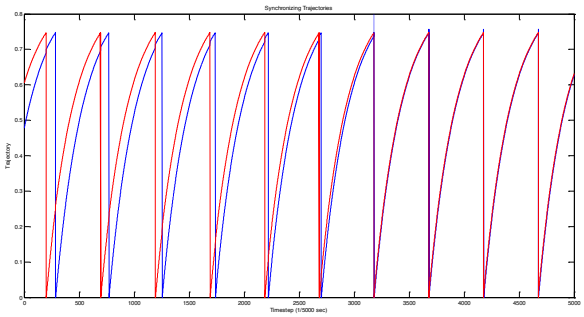


Figure 7.7 An illustration of the synchronization of two weakly coupled oscillators. The phases get closer and closer over time and become synchronized at $t = 3181$.

after B fires, A's state jumps to

$$x_A = f(1-\phi) + \epsilon, \tag{7.6}$$

if it is less than 1. Of course, if it is not less than 1, it must be 1, thus achieving synchronization between A and B's firing. So let us say it is less than 1. Obviously, for that to happen, ϵ needs to be less than 1 (thus the assumption of *weak* coupling here), and ϕ needs to be sufficiently away from 0: since $f(1-\phi)+\epsilon < 1$, we know

$$\phi > \phi_{min} = 1 - f^{-1}(1-\epsilon).$$

Now back to (7.6), the corresponding phase is

$$\phi_A = f^{-1}(f(1-\phi)+\epsilon) = h(\phi), \tag{7.7}$$

where h is a shorthand notation, standing for "half" of a complete round.

In summary, after one firing, *i.e.*, half of a return round, we go on the state-phase plane from

$$(\phi_A, \phi_B) = (0, \phi)$$

to

$$(\phi_A, \phi_B) = (h(\phi), 0).$$

Applying h one more time, we will get to

$$(\phi_A, \phi_B) = (0, h(h(\phi))).$$

So the return function is simply

$$R(\phi) = h(h(\phi)),$$

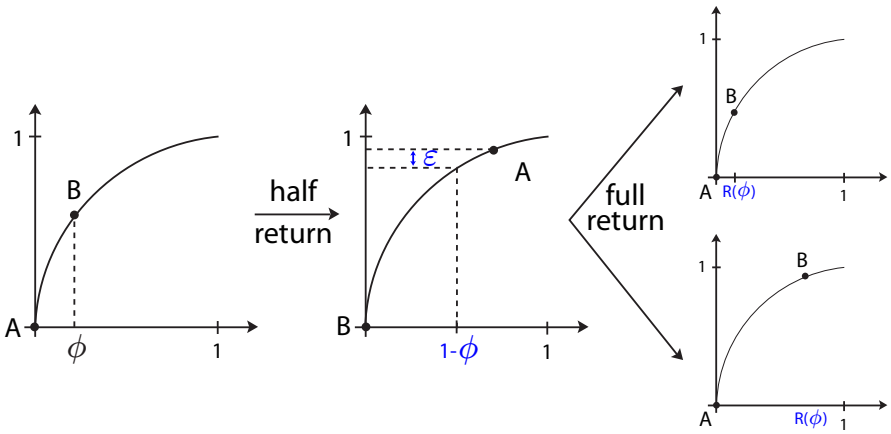


Figure 7.8 Pictorial illustration of the proof of synchronization. h indicates half of a return (from A firing to B firing). R indicates a full return (from A firing to A firing again). We want to show that if ϕ, the phase of B at A firing, is smaller than a threshold, B will be closer to phase 0 after one return. And if ϕ is larger than that threshold, B will be closer to phase 1 after one return. Either way, A and B are driven closer to synchronization after each return.

where h is defined as in (7.7). This is illustrated in Figure 7.8.

Now we want to show that as A and B each gets to fire, they will be closer on the (ϕ, x) plane, either towards $\phi = 0$ or $\phi = 1$, so eventually they must meet. This is a **contraction mapping** argument.

So, we want to show that there is a unique ϕ^*, such that $R(\phi) > \phi$ if $\phi > \phi^*$ and $R(\phi) < \phi$ if $\phi < \phi^*$. This will prove that synchronization must eventually happen.

So, we want to first show that there is a unique fixed point ϕ^* of R. If h has a unique fixed point, obviously R has too.

Does h has a unique fixed point? Or, equivalently, is there one, and only one point ϕ^*, such that

$$F(\phi^*) = 0,$$

where $F(\phi) = \phi - h(\phi)$? Well, we can easily check that $\phi_{min} < h(\phi_{min})$, and consequently, flipping the arrow of time, $h^{-1}(\phi_{min}) > h(h^{-1}(\phi_{min})) = \phi_{min}$. That means

$$F(\phi_{min}) < 0 \text{ and } F(h^{-1}(\phi_{min})) > 0.$$

There must be some $\phi^* \in [\phi_{min}, h^{-1}(\phi_{min})]$ such that $F(\phi^*) = 0$, *i.e.*, a fixed point. We have proved the existence of fixed points.

But there might be *many* fixed points in the above interval. Let us check F's

first derivative $1 - h'(\phi)$. By the chain rule, we know that

$$h'(\phi) = -\frac{f^{-1'}(\epsilon + f(1-\phi))}{f^{-1'}(f(1-\phi))}.$$

Since f is increasing and concave, we know $f^{-1'}$ is increasing and convex, the above ratio must be less than -1. That means F' must be greater than 2, thus F is monotonically increasing. This proves that not only are there fixed points, but also there is a *unique* fixed point of h.

Therefore, R also has a unique fixed point ϕ^*. By the chain rule, we know $h' < -1$ implies $R' > 1$. So if $\phi > \phi^*$, after two rounds of firing, one by A and another by B, the system is driven toward $\phi = 1$. If $\phi < \phi^*$, it is driven toward $\phi = 0$. Either way, it is driven closer to synchronization. More rounds will eventually lead to synchronization.

The above derivation does not use a differential equation approach as we will see next in the diffusion model. That would have been a non-scalable proof technique as we generalize from 2 oscillators to N of them. Instead, this *fixed point* approach readily generalizes itself to all N, and for all increasing and concave trajectories too, without having to write down the differential equation. It goes straight to the root cause that leads to synchronization.

Over the past two decades, there have also been other generalizations: to different types and representations of coupling, to network topology-dependent coupling, and to synchronization even with delay in coupling.

7.4.2 Diffusion

The tipping model is based on population dynamics. It can be further elaborated in various ways. Consider a fixed market of size M people that a new product (or a song, or an app) wants to diffuse into. Each user chooses between adopting or not, again in a binary state of mind, based on two factors:

- Something intrinsic about the product itself, independent of how many other people have adopted it. Some call this the innovation or external component. We will refer to it as the **intrinsic factor**.
- The networking effect: either because more adopters change the value of the product itself, or because of the information cascade effect. Some call this the internal component. We will refer to this as the **imitation factor**.

At time t, the number of adopters is $n(t)$, and the cumulative number of adopters is $N(t)$. The following equation is the starting point of the **Bass model** in 1969:

$$n(t) = \frac{dN(t)}{dt} = \left[p + q\frac{N(t)}{M}\right](M - N(t)), \tag{7.8}$$

where p is the **intrinsic factor coefficient**, and q the **imitation factor coefficient**, which multiplies the fraction of adopters in the market. The sum of

the two effects is the growth factor, which when multiplied with the number of non-adopters left in the population, gives the number of new adopters at time slot t. Let us assume at time 0, $N(0) = 0$: the product starts from zero market diffusion.

This population-based model, captured through a simple differential equation, has been extensively studied and generalized in the marketing literature. The basic version above has a closed-form solution. Solving the differential equation (7.8) with zero initial condition, we see that, at each time t, the number of new adopters is

$$n(t) = M\frac{p(p+q)^2\exp(-(p+q)t)}{(p+q\exp(-(p+q)t))^2}. \tag{7.9}$$

Further integrating the resulting $n(t)$ over time, the number of total adopters at to that point is

$$N(t) = M\frac{1-\exp(-(p+q)t)}{1+\frac{q}{p}\exp(-(p+q)t)}. \tag{7.10}$$

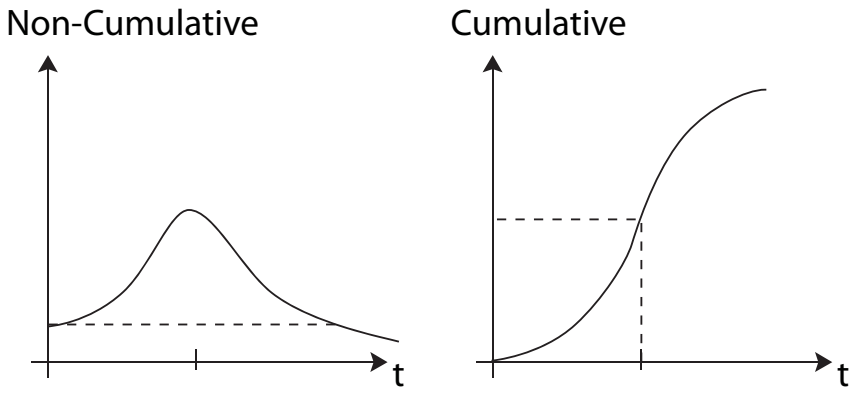


Figure 7.9 Two typical graphs illustrating the Bass diffusion model. The non-cumulative adoption rate is shown on the left, rising initially and then quiet down as the market saturates. The cumulative adoption population is shown on the right as a sigmoidal curve.

Figure 7.9 shows the typical evolution of diffusion over time: the cumulative adoption $N(t)$ climbs up to M through a sigmoidal function: convex first and then concave past the inflection point. The reason is clear: the adoption rate curve $n(t)$ rises initially because there are still many non-adopters, but eventually, the market saturates and there are not enough non-adopters left.

What is interesting is to see the breakdown of $N(t)$ into those adoptions due to the intrinsic factor, which monotonically decreases over time because of market saturation, and those due to the imitation factor, which rises for a while, as the

imitation effect overcomes the market saturation effect. The peak of this rise, and the rise of $n(t)$, happens at time

$$T = \frac{1}{p+q} \log(\frac{q}{p}).$$

This makes sense since it is the ratio of the two coefficients that determines the relative weight of the two curves, and the faster the overall adoption coefficient, the earlier the market reaches its peak rate of adoption.

As a numerical example, we take $p = 0.01$ and $q = 0.1$ or 0.2. The diffusion behavior is shown in Figures 7.10 and 7.11. From both graphs, we see that when q is larger, the adoption happens faster.

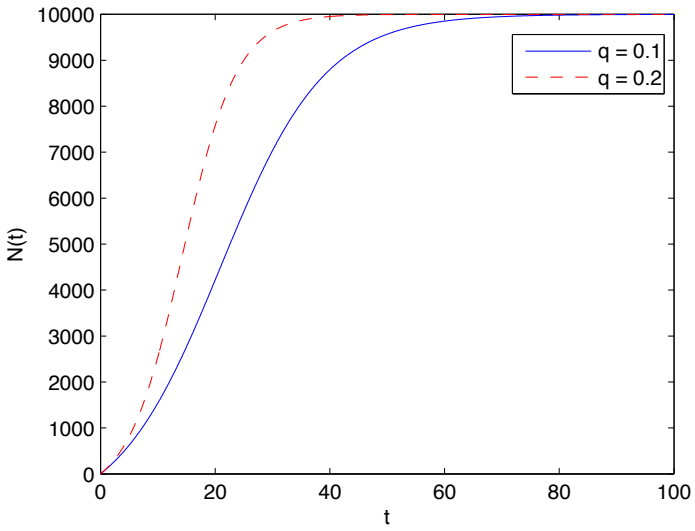


Figure 7.10 A numerical example of the trajectory of Bass model for diffusion. The cumulative adoption is shown over time, for two different values of the imitation factor coefficient q. Both curves are sigmoidal: convex first and then concave.

But how do we pick p and q values? That is often done through a parameter training phase like the baseline predictor in the Netflix Prize in Chapter 4, using known marketing data, or a new product trial. Once p and q are fixed, we can run the model for prediction and planning purposes.

There have been many extensions in the past four decades of the basic model above:

- Incorporate individual decision-making processes.
- Divide the population into groups with different behaviors.
- Allow the market size M to vary with the effectiveness of adoption rather than fixed a priori.

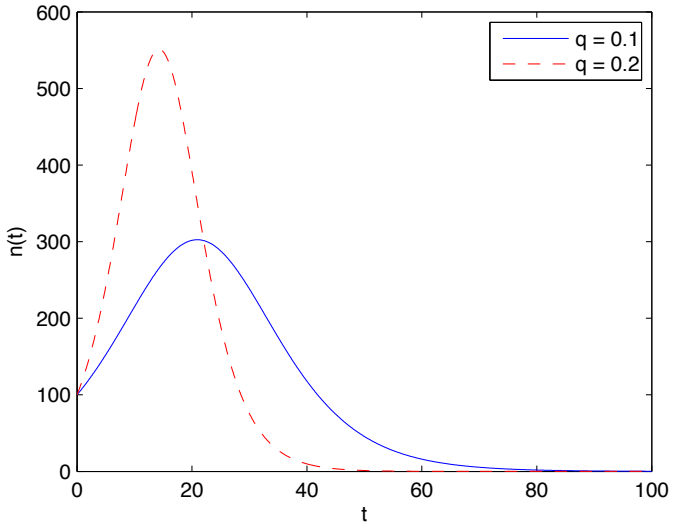


Figure 7.11 A numerical example of the trajectory of Bass model for diffusion. The rate of adoption is shown over time, for two different values of the imitation factor coefficiente q. Both curves rise up to a peak value and then decay to zero when everyone has adopted.

- Others like time-varying nature of innovation, limitation of supply, and non-binary choices by consumers.

There have also been many refinements of the coefficient models, especially the imitation factor coefficient q. We highlight a particularly interesting one that has the element of time explicitly modeled: **flexible logistic growth** model. Here we have

$$q(t) = q(1 + kt)^{\frac{\mu - k}{k}},$$

where k and μ are two parameters modeling the increasing, decreasing, or constant power of imitation over time, and their values, when properly chosen, can allow the model to put the inflection point anywhere between 0 and 100% market diffusion.

The resulting cumulative adoption population has a somewhat complicated form. In the simpler case when $p = 0$, it is:

$$N(t) = \frac{M}{1 + \exp(-(c + qt(\mu, k)))},$$

where c is some constant, and the function $t(\mu, k)$, when $\mu \neq 0$ and $k \neq 0$, is

$$\frac{(1 + kt)^{\mu/k} - 1}{\mu},$$

and, when $\mu = 0, k \neq 0$, is

$$\frac{1}{k}\log(1+kt).$$

One can also think of a model where the market size is essentially infinite, and there is no market saturation effect. For example, the change in $N(t)$ is the product of imitation effect $qN(t)$ and a penalty term $D(t)$ that decreases the attractiveness of the product as time goes on. This then leads to the following differential equation:

$$n(t) = \frac{dN(t)}{dt} = qN(t)D(t) - N(t).$$

So, when time t is such that $D(t) < 1/q$, the market adoption rate will decrease even without a fixed market size.

No matter which model we use, there are two essential ingredients: an imitation effect q and a shrinkage effect (either because of time passing $D(t)$, or because of a fixed market size M), as well as an optional ingredient of intrinsic effect p.

Further Reading

The material in this chapter comes from a diverse set of sources: political economy, marketing, physics, and sociology.

1. On information cascade model, we follow the classical work below:

[BHW92] S. Bikhchandani, D. Hirshleifer, and I. Welch, "A theory of fads, fashion, custom, and cultural change as information cascades," *Journal of Political Economy*, vol. 100, no. 5, pp. 992-1026 , October 1992.

2. On synchronization model, we follow the seminal paper using fixed point techniques to analyze synchronization:

[MS90] R. E. Mirollo and S. H. Strogatz, "Synchronization of pulse-coupled biological oscillators", *SIAM Journal of Applied Mathematics*, vol. 50, no. 6, pp. 1645-1662, December 1990.

3. On diffusion model, an extensive survey of the research area since 1968 can be found at:

[MMB90] V. Mahajan, E. Muller, and F. M. Bass, "New product diffusion models in marketing: A review and directions for research", *Journal of Marketing*, vol. 54, no. 1, pp. 1 - 26, January 1990.

4. A class work on threshold models in sociology is the following one:

[Gra78] M. Granovetter, "Threshold models of collective behavior," *Americal Journal of Sociology*, vol. 83, no. 6, 1978.

5. The music market's trends under social influence were studied in the following experiment, which we summarized in this chapter:

[SDW06] M. J. Salganik, P. S. Dodds, and D. J. Watts, "Experimental study of inequality and unpredictability in an artificial cultural market," *Science*, vol. 311, pp. 854 - 856, February 2006.

Problems

7.1 *A citation network* ⋆

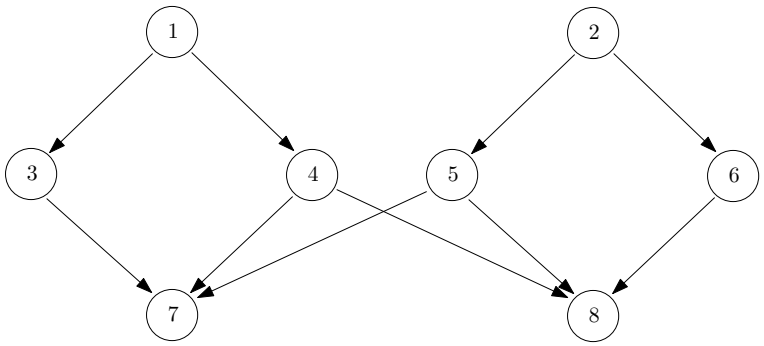


Figure 7.12 A citation network.

Consider a set of eight papers with their citation relationships represented by the graph in Figure 7.12. Each paper is a node, and a directed edge from node i to node j means paper i cites paper j.

(a) Write down the adjacency matrix $\mathbf{A}$, where the (i, j) entry is 1 if node i and node j are connected, and 0 otherwise.

(b) Compute the matrix $\mathbf{C}$ defined as

$$\mathbf{C} = \mathbf{A}^T\mathbf{A}$$

and compare the values C_{78} and C_{75}. In general, what is the physical interpretation of the entries C_{ij}?

7.2 *Physical interpretation of matrix multiplication* ⋆

(a) Refer to the graph in Figure 7.12. Compute

$$\mathbf{A}^2 = \mathbf{A}\mathbf{A},$$
$$\mathbf{A}^3 = \mathbf{A}^2\mathbf{A}.$$

(b) Is there anything special about $\mathbf{A}^3$? In general, what do the entries in $\mathbf{A}^m$ (where $m = 1, 2, \ldots$) represent?

7.3 *Hypergraphs and bipartite graphs* ⋆⋆

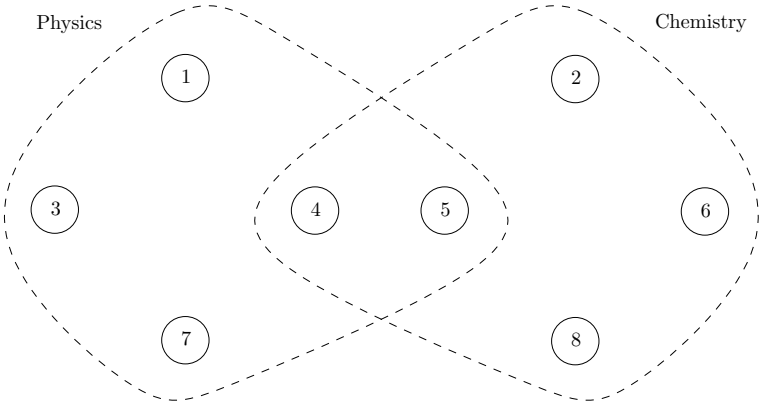


Figure 7.13 Hypergraphs allow each link to connect more than two nodes.

Suppose the papers are in the fields of physics or chemistry. Group membership, *i.e.*, to which field a papers belongs, is presented as a **hypergraph** in Figure 7.13: each dotted area is a group or a **hyperedge**, which is a generalization of an undirected edge to connect possibly more than two nodes. Note that papers 4 and 5 are "interdisciplinary" papers, so their nodes are contained in both hyperedges.

(a) We can transform the hypergraph in Figure 7.13 into an undirected bipartite graph by introducing two more nodes, each representing one of the hyperedges, and linking a "standard" node with a "hyperedge" node with an edge if the former is contained in the corresponding hyperedge. Draw this bipartite graph.

(b) Define an *incidence matrix* $\mathbf{B}$ of size 2×8 with

$$B_{ij} = \begin{cases} 1 & \text{node } j \text{ is contained in group } i, \\ 0 & \text{otherwise,} \end{cases}$$

where group 1 is "Physics" and group 2 is "Chemistry". Write down $\mathbf{B}$ for this graph.

(c) Compute the matrix $\mathbf{B}^T\mathbf{B}$. What is its interpretation?

(d) Compute the matrix $\mathbf{B}\mathbf{B}^T$. What is its interpretation?

7.4 *Perturbing flipping behaviors* ⋆⋆

(a) Consider the $f(p)$ function in Figure 7.14, which has four equilibria $p^* = 0,\ \frac{1}{3},\ \frac{2}{3},\ 1$. Suppose we start with $p_0 = 0.01$ (or some constant slightly greater than 0), find the equilibrium fraction p_∞.

(b) Suppose $f(p)$ is slightly modified as in Figure 7.15, such that the point

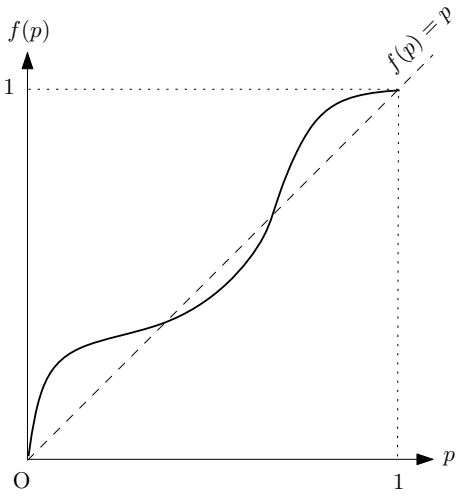


Figure 7.14 An original curve to model flipping.

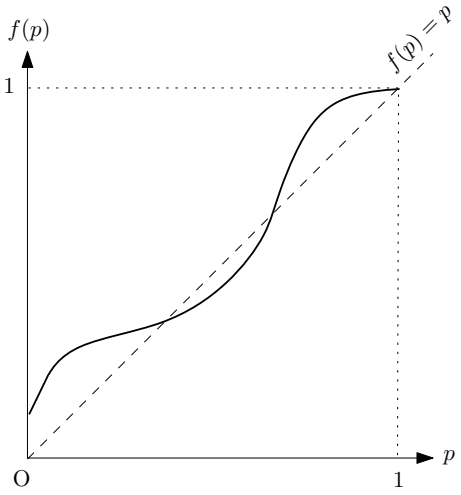


Figure 7.15 The curve is modified.

$p = 0$ is no longer an equilibrium, *i.e.*, $f(0) > 0$. Again use a graphical argument to find p_∞, starting at $p_0 = 0$.

(c) Suppose $f(p)$ is further slightly modified as in Figure 7.16, such that $f(p) > p$ for $0 \leq p < 1$. Find p_∞ starting at $p_0 = 0$.

7.5 *Flocking birds* ⋆⋆⋆

In our study of the dynamics of collective behavior, we are often intrigued by the process that goes from individual actions and local interactions to large-scale, global patterns. This happens in many ways in human crowds, bird flocks, fish schools, bacteria swarms, etc. A simple and powerful illustration is Conway's game of life.

(A comprehensive survey of animal behavior can be found in I. D. Couzin and

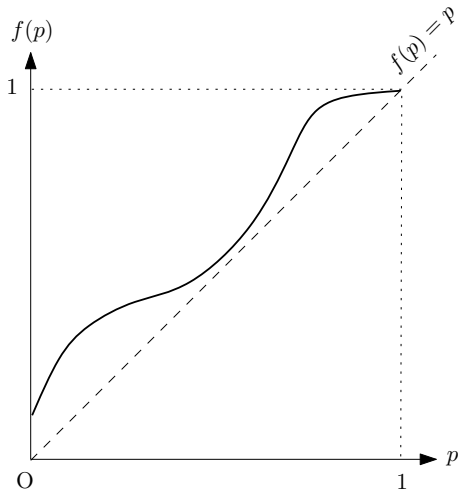


Figure 7.16 The curve is modified again.

J. Krause, "Organization and collective behavior in vertebrates," *Advances in the Study of Behavior*, 2003.)

Here is a very simple model for bird flocks that assumes away many features but sufficies to illustrate the point in a homework problem. Suppose there is a 2-dimensional plane and N points moving in it. Each point has neighbors, which are the points within a circle of radius r units. All the points move with the same constant speed, say, 1 unit, but along different directions. At each time slot, each point's direction is updated as the average of its current direction and all the directions of its neighbors. We can think of a graph where each node is a point, and each link is a neighbor relationship. But this graph evolves over time as the points' positions change.

In this homework problem, you are asked to random place 100 points in a 10 by 10 units square, and initialize their directions randomly. You should try different values of r. Simulate the above model over time, and see what happens to the directions of the points.

(For more details, see T. Vicsek, A. Czirok, E. Ben Jacob, I. Cohen, and O. Schochet, "Novel type of phase transitions in a system of self-driven particles," *Physics Review Letter*, vol. 75, pp. 1226-1229, 1995.)

8 How do I influence people on Facebook and Twitter?

To study a network, we have to study both its *topology* (the graph) and its *functionalities* (tasks carried out on top of the graph). This chapter on topology-dependent influence models indeed pursues both, as do the next two chapters.

8.1 A Short Answer

Started in October 2003 and formally founded in February 2004, Facebook has become the largest social network website, with 750 million users worldwide and 140 million unique monthly visitors in the U.S. as of summer 2011. Many links have been formed among these nodes, although it is not straightforward to define how many frequent mutual activities on each other's wall constitute a link.

Founded in July 2006, Twitter attracted more than 200 million users in 5 years. In summer 2011, over 200 million tweets were handled by Twitter each day. Twitter combines several functionalities into one platform: microblogging (with no more than 140 characters), group texting, and social networking (with one-way following relationship, *i.e.*, directional links).

Facebook and Twitter have become two of the most influential communication modes, especially among young people. In summer 2011's east coast earthquake in the US, tweets traveled faster than the earthquake itself from Virginia to New York. They have also become a major mechanism in social organization. In summer 2009, Twitter became a significant force in how the Iranians organized themselves against the totalitarian regime, and again in 2011 during the Arab Spring.

There have been all kinds of attempts at figuring out

- (a) how to measure the influential power of individuals on Facebook or Twitter;
- (b) how to leverage the knowledge of influential power's distribution to actually influence people online.

Question (a) is an analysis problem and question (b) a synthesis problem. Neither is easy and there is a significant gap between theory and practice. Later in this chapter we will visit some of the fundamental models that have yet to

make a significant impact on characterizing and optimizing influence over these networks.

But the difficulty did not prevent people from trying out heuristics. Regarding (a), for example, there are many companies charting the influential power of individuals on Twitter, and there are several ways to approximate that influential power: by the number of followers, by the number of retweets (with "RT" or "via" in the tweet), or by the number of reposting of URLs. There are also companies data mining the friendship network topology of Facebook.

As to (b), Facebook uses simple methods to recommend friends, often based on email contact lists or common backgrounds. Marketing firms also use Facebook and Twitter to stage marketing campaigns. Some "buy off" a few influential individuals on these networks, while others buy off a large number of randomly chosen, reasonably influential individuals.

It is important to figure out who are the influential people. An often-quoted historical anectode are the night rides by Paul Revere and by William Dawes on 18-19 April in 1775. Dawes left Boston earlier in the evening than Revere. They took different paths towards Lexington, and then rode together from Lexington to Concord. Revere alerted influential militia leaders along his route to Lexington, and was much more effective in spreading the word of the imminent British military action, leading to winning on the next day the first battle that started the American Revolutionary War.

How do we quantify which nodes are more important? The question dates back thousands of years, and one particularly interesting example occurred during the Renaissance in Italy. The Medici family was often viewed as the most influential among the 15 most prominent families in Florence in the 15th and 16th century. As shown in Figure 8.1, it sat in the "center" of the family social network through strategic marriages. We will see several ideas quantifying the notion of centrality.

How do we quantify which links (and paths) are more important? We will define strong vs. weak ties. Their effects can be somewhat unexpected. For example, Granovetter's 1973 survey in Amherst, Massachusetts showed the strength of weak ties in communicating. We will see another surprise of weak ties' roles in social networks, on six degree separation in Chapter 9.

Furthermore, how do we quantify which subset of nodes (and the associated links) are connected enough among themselves, and yet disconnected enough from the rest of the network, that we can call them a "group"? We save this question for Advanced Material.

Before proceeding further, we first formally introduce two commonly used matrices to describe graphs. We will construct a few other matrices later as concise and useful representations of graphs. We will see that properties about a graph can often be summarized by linear-algebraic quantities about the corresponding matrices.

First is the **adjacency matrix A**, of dimension N by N, of a given graph G with N nodes connected through links. For the graphs we deal with in this book, $\mathbf{A}_{ij}$ is 1 if there is a link from node i to j, and 0 otherwise. We mostly focus on

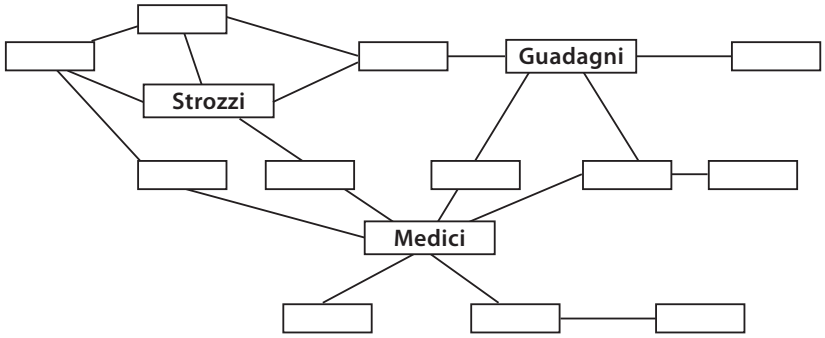


Figure 8.1 The central position of the Medici family in Renaissance Florence. Each node is a family, three of them shown with their names. Each link is a marriage or kinship relationship. The Medici family clearly had the largest degree, but its influential power relative to the other families was much more than the degree distribution would indicate. Other measures of centrality, especially betweenness centrality, reveal just how influential the Medici family was.

undirected graphs in this chapter, where each link is **bidirectional**. Given an undirected graph, $\mathbf{A}$ is symmetric: $A_{ij} = A_{ji},\ \forall i, j$. If a link can be **unidirectional**, we have a **directed** graph, like the Twitter following relationship graph.

Second and less used in this book is the **incidence matrix** $\hat{\mathbf{A}}$, of dimension N by L, where N is again the number of nodes and L the number of links. For an undirected graph, $\hat{A}_{ij} = 1$ if node i is on link j, and 0 otherwise. For a directed graph, $\hat{A}_{ij} = 1$ if node i starts link j, $\hat{A}_{ij} = -1$ if node i ends link j, and $\hat{A}_{ij} = 0$ otherwise.

Straightforward as the above definitions may be, it is often tricky to define what exactly constitutes a link between two persons: known by first-name as in Milgram's small world experiment? Or friends on Facebook who have never met or communicated directly? Or only those with whom you text at least one message a day? Some links are also directional: I may have commented on your wall postings on Facebook but you never bothered reading my wall at all. Or I may be following your Tweets, but you do not follow mine.

Even more tricky is to go beyond the simple static graph metrics and into the functionalities and dynamics on a graph. But that is much tougher a subject, and we start with some simple static graph metrics first.

8.2 A Long Answer

8.2.1 Measuring node importance

You must be in many social networks, online as well as offline. How important are you in each of those networks? Well, that depends on how you define the "importance" of a node. Eventually it depends on the specific functionalities we are looking at and it evolves over time, but we shall restrict ourselves to just static graph metrics for now. And it is not easy to discover the actual topology of the network. But let us say for now that we are given a network of nodes and links.

There are at least four different approaches to measuring the importance, or **centrality** of a node, say node 1.

The first obvious choice is **degree**: the number of nodes connected to node 1. If it is a directed graph, we can count two degrees: the **in-degree**: the number of nodes pointing towards node 1, and the **out-degree**: the number of nodes that node 1 points to. **Dunbar's number**, usually around 150, is often viewed as the number of friends a typical person may have, but the exact number of course depends on the definition of "friends". The communication modes of texting, tweeting, and blogging may have created new shades of definition of "friends". In Google+, you can also create your own customized notions of friends by creating new circles.

But we will see there are many issues with using the degree of a node as its centrality measure. One issue is that if you are connected to more important nodes, you will be more important than you would be if you were connected to less important nodes. This reminds you of the pagerank definition in Chapter 3. Indeed, we can take pagerank as a centrality measure.

A slightly simpler but still useful view of centrality is to just look at the successive multiplication of the centrality vector $\mathbf{x}$ by the adjacency matrix $\mathbf{A}$ that describes the network topology, starting with an initialization vector $\mathbf{x}(0)$:

$$\mathbf{x}(t) = \mathbf{A}^t \mathbf{x}(0).$$

In a homework problem, you will discover a motivation for this successive multiplication.

We can always write a vector as a linear combination of the eigenvectors $\{\mathbf{v}_i\}$ of $\mathbf{A}$, arranged in descending order and indexed by i, for some weight constants $\{c_i\}$. For example, we can write the vector $\mathbf{x}(0)$ as follows:

$$\mathbf{x}(0) = \sum_i c_i \mathbf{v}_i.$$

Now we can write $\mathbf{x}(t)$ at any iteration t as a weighted sum of $\{\mathbf{v}_i\}$:

$$\mathbf{x}(t) = \mathbf{A}^t \sum_i c_i \mathbf{v}_i = \sum_i c_i \mathbf{A}^t \mathbf{v}_i = \sum_i c_i \lambda_i^t \mathbf{v}_i, \tag{8.1}$$

where $\{\lambda_i\}$ are the eigenvalues of $\mathbf{A}$.

As $t \to \infty$, the effect of the largest eigenvalue λ_1 will dominate, so we approximate by looking only at the effect of λ_1. Now the **eigenvector centrality** measures $\{x_i\}$ constitute a vector that solves:

$$\mathbf{A}\mathbf{x} = \lambda_1 \mathbf{x},$$

which means

$$x_i = \frac{1}{\lambda_1} \sum_j A_{ij} x_j, \quad \forall i. \tag{8.2}$$

We can also normalize the eigenvector centrality $\mathbf{x}$.

The third notion, **closeness centrality**, takes a "distance" point of view. Take a pair of nodes (i, j) and find the shortest path between them. It is not always easy to compute shortest path, as we will see in Chapters 9 and 13. But for now, say we have found these shortest paths, and denote their length as d_{ij}. The largest d_{ij} across all (i, j) pairs is called the **diameter** of the network. The average of d_{ij} for a given node i across all other $n-1$ nodes is an average distance $\sum_j d_{ij}/(n-1)$. Closeness centrality is the reciprocal of this average:

$$C_i = \frac{n-1}{\sum_j d_{ij}}. \tag{8.3}$$

We have used the arithmetic mean, but could also have used other "averages", such as the harmonic mean.

Closeness centrality is quite intuitive: the more nodes you know or closer you are to other nodes, the more central you are in the network. But there is another notion that is just as useful, especially when modeling influence and information exchange: **betweenness centrality**. If you are on the (shortest) paths of many *other* pairs of nodes, then you are important. We can also extend this definition to incorporate more than just the shortest paths, as in the context of Internet routing in Chapter 13.

Let g_{st} be the total number of shortest paths between two diffferent nodes, source s and destination t (neither of which is node i itself), and n_{st}^i be the number of such paths that node i sits on. Then betweenness centrality of node i is defined as

$$B_i = \sum_{st} \frac{n_{st}^i}{g_{st}}. \tag{8.4}$$

A node with a large closeness centrality may not have a large betweenness centrality, and vice versa. In fact, many social networks exhibit the small world phenomenon as explained in Chapter 9, and the closeness centrality values of different nodes tend to be close to each other. Betweenness centrality values tend to have a larger dynamic range.

We can also think of hybrid metrics. For example, first weight each node by eigenvector centrality, then weight each node pair by the product of their eigenvector centrality values, and then calculate betweenness centrality by weighting each (st) term in (8.4) accordingly.

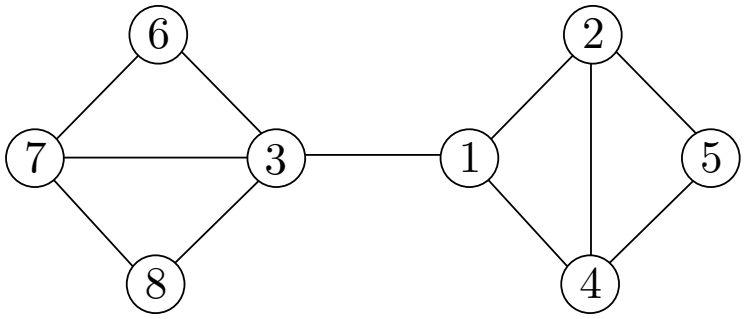


Figure 8.2 An example network to illustrate node importance metrics, link importance metrics, and group connectedness metrics. For example, how much more important node 1 is relative to node 2 depends on which centrality metric we use.

In the Renaissance Florence family relationship graph, it is obvious that the Medici family has the largest degree, 6, but the Strozzi and Guadagni families degrees are not too far behind: 4 for both. But if we look at the betweenness centrality values, Medici family has a value of 50.83, which is 5 times as central as Strozzi and twice as central as Guadagni, not just a mere factor of 1.5.

Now let us take a look at how different node importance metrics turn out to be for two of the nodes, 1 and 2, in the small network in Figure 8.2.

For degree centrality, obviously $d_1 = d_2 = 3$. But nodes 1 and 2 cannot be the same in their importance according to most people's intuition that says nodes gluing the graph together are more important.

Let us compute eigenvector centrality next. From the adjacency matrix

$$\mathbf{A} = \begin{bmatrix} 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \end{bmatrix},$$

we can solve for

$$\mathbf{A}\mathbf{x} = \lambda_1 \mathbf{x}$$

to obtain

$$\lambda_1 = 2.8723,$$
$$\mathbf{x} = [0.4063, 0.3455, 0.4760, 0.3455, 0.2406, 0.2949, 0.3711, 0.2949]^T.$$

So node 1 is slightly more important than node 2: $0.4063 > 0.3415$. But is it really just *slightly* more important?

To compute closeness centrality, we first write out the pairwise (shortest) distances from node 1 to the other nodes, and from node 2 to the other nodes:

$$d_{12} = 1, d_{13} = 1, d_{14} = 1, d_{15} = 2, d_{16} = 2, d_{17} = 2, d_{18} = 2,$$
$$d_{21} = 1, d_{23} = 2, d_{24} = 1, d_{25} = 1, d_{26} = 3, d_{27} = 3, d_{28} = 3.$$

Then, we can see that node 1 is again only slightly more important:

$$C_1 = \frac{7}{1+1+1+2+2+2+2} = 0.6364$$
$$C_2 = \frac{7}{1+2+1+1+3+3+3} = 0.5.$$

Finally, to compute betweenness centrality, it helps to first write out the quantities of interest. Let $\mathbf{G}$ be the matrix with $G_{ij} = g_{ij}$, $\mathbf{N}^1$ be the matrix with $\mathbf{N}^1_{ij} = n^1_{ij}$ for node 1, and $\mathbf{N}^2$ be the matrix with $\mathbf{N}^2_{ij} = n^2_{ij}$ for node 2. Also let X denote a matrix entry that is not involved in the calculations (a "don't care"). Then, we have

$$\mathbf{G} = \begin{bmatrix} X & 1 & 1 & 1 & 2 & 1 & 1 & 1 \\ 1 & X & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & X & 1 & 2 & 1 & 1 & 1 \\ 1 & 1 & 1 & X & 1 & 1 & 1 & 1 \\ 2 & 1 & 2 & 1 & X & 2 & 2 & 2 \\ 1 & 1 & 1 & 1 & 2 & X & 1 & 2 \\ 1 & 1 & 1 & 1 & 2 & 1 & X & 1 \\ 1 & 1 & 1 & 1 & 2 & 2 & 1 & X \end{bmatrix},$$

$$\mathbf{N}^1 = \begin{bmatrix} X & X & X & X & X & X & X & X \\ X & X & 1 & 0 & 0 & 1 & 1 & 1 \\ X & 1 & X & 1 & 2 & 0 & 0 & 0 \\ X & 0 & 1 & X & 0 & 1 & 1 & 1 \\ X & 0 & 2 & 0 & X & 2 & 2 & 2 \\ X & 1 & 0 & 1 & 2 & X & 0 & 0 \\ X & 1 & 1 & 0 & 2 & 0 & X & 0 \\ X & 1 & 0 & 1 & 2 & 0 & 0 & X \end{bmatrix},$$

$$\mathbf{N}^2 = \begin{bmatrix} X & X & 0 & 0 & 1 & 0 & 0 & 0 \\ X & X & X & X & X & X & X & X \\ 0 & X & X & 0 & 1 & 0 & 0 & 0 \\ 0 & X & 0 & X & 0 & 0 & 0 & 0 \\ 1 & X & 1 & 0 & X & 1 & 1 & 1 \\ 0 & X & 0 & 0 & 1 & X & 0 & 0 \\ 0 & X & 0 & 0 & 1 & 0 & X & 0 \\ 0 & X & 0 & 0 & 1 & 0 & 0 & X \end{bmatrix}.$$

Applying (8.4), we clearly see an intuitive result this time: node 1 is much more

important than node 2:

$$B_1 = 12,$$
$$B_2 = 2.5.$$

8.2.2 Measuring link importance

Not all links are equally important. Sometimes there is a natural and operationally meaningful way to assign weights, whether integers or real numbers, to links. For example, the frequency of Alice retweeting Bob's tweets, or of reposting Bob's URL tweets, can be the weight of the link from Bob to Alice. Sometimes there are several categories of link strength. For example, you may have 500 friends on Facebook, but those with whom you have had either one way or mutual communication might be only 100, and those with whom you have had mutual communication might be only 20. The links to these different types of Facebook friends belong to different strength classes. Weak links might be weak in action, but strong in information exchange.

A link can also be important because it "glues" the network together. For example, if a link's two end points A and B have no common neighbors, or more generally, do not have any other paths of connection with k hops or less, this link is *locally important* in connecting A and B. Links that are important connecting many node pairs, especially when these nodes are important nodes, are *globally important* in the entire network. These links can be considered as *weak*, since they connect nodes that otherwise have no, or very little, overlap. (The opposite is "triad closure" that we will see in the next chapter.)

But these weak links are *strong* precisely for the reason that they open up communication channels across groups that normally do not communicate with each other, as seen in Granovetter's 1973 experiment. One way to quantify this notion is **link betweenness**: similar to the betweenness metric defined for nodes, but now we count how many shortest paths a *link* lies on.

Back to Figure 8.2, we can compute $B_{(i,j)}$, the betweenness of link (i, j) by

$$B_{(i,j)} = \sum_{st} \frac{n_{st}^{(i,j)}}{g_{st}}, \tag{8.5}$$

where $n_{st}^{(i,j)}$ is the number of shortest paths between two nodes s and t that traverse link (i, j).

Let $\mathbf{N}^{(i,j)}$ be a matrix with the (s, t) entry as $n_{st}^{(i,j)}$. We can compare, *e.g.*, the

betweenness values of the links $(1,3)$ and $(1,2)$.

$$\mathbf{N}^{(1,3)} = \begin{bmatrix} X & 0 & 1 & 0 & 0 & 1 & 1 & 1 \\ 0 & X & 1 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & X & 1 & 2 & 0 & 0 & 0 \\ 0 & 0 & 1 & X & 0 & 1 & 1 & 1 \\ 0 & 0 & 2 & 0 & X & 2 & 2 & 2 \\ 1 & 1 & 0 & 1 & 2 & X & 0 & 0 \\ 1 & 1 & 0 & 1 & 2 & 0 & X & 0 \\ 1 & 1 & 0 & 1 & 2 & 0 & 0 & X \end{bmatrix},$$

$$\mathbf{N}^{(1,2)} = \begin{bmatrix} X & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & X & 1 & 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & X & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & X & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & X & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 & X & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & X & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & X \end{bmatrix}.$$

Now we can compute link importance by (8.5). We have the intuition quantified: link (1,3) is much more important than link (1,2) as it glues together the two parts of the graph:

$$B_{(1,3)} = 16,$$
$$B_{(1,2)} = 7.5.$$

8.2.3 Contagion

Now that we have discussed the basic (static) metrics of a graph, we continue with our influence model discussions with the help of network topology.

Remember the last chapter's section on tipping behavior under best response strategy? In this 2-state model, the initialization has a subset of the nodes adopting one state, for example, the state of 1, while the rest of the nodes adopt the other state, the state of 0. We can consider the state-1 nodes as the early adopters of a new product, service, or trend, so it is likely they would be a small minority in the network and not necessarily aggregated together.

Now the question is: when, if at all, will all the nodes flip to the new trend, *i.e.*, flip from state-0 to state-1?

Unlike the diffusion model in the last chapter, now it is the *local* population, the set of neighbors of each node, rather than the global population, that matters. One possible local flipping rule is a memoryless, threshold model: if $p\%$ or more of your neighbors flipped to state-1, you will flip too. For now, let us say all the nodes have the same flipping threshold: the same p for all nodes.

An example is shown in Figure 8.3 with a flipping threshold of $p = 0.49$, and the highlighted node being initialized at state-1. At time 1, the leftmost

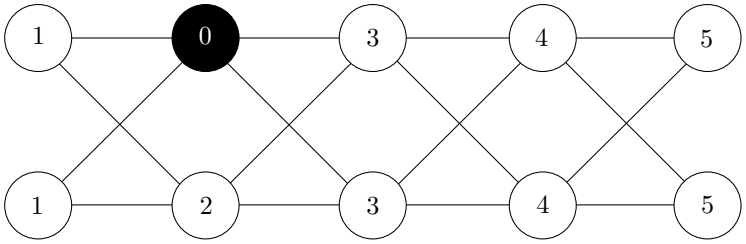


Figure 8.3 Example for contagion model with flipping threshold $p = 0.49$. Numbers in nodes indicate the times at which they change from state-0 to state-1: at time 1 the leftmost two nodes flip because one of their two neighbors is in state-1, then this triggers a cascade of flips from left to right.

two nodes flip, and that triggers a cascade of flipping from left to right of the network.

In general, the first question we would ask is: will the entire network flip? It turns out there is a clear-cut answer: if and only if there is no cluster of density $1 - p$ or higher, in the set of nodes with state-0 at initialization. As will be elaborated in Advanced Material, a **cluster** of density p is a set of nodes such that each of these nodes has at least $p\%$ of its neighbors also in this set. For example, in Figure 8.4, the set of nodes forming the lower left triangle is a cluster with density 0.5, whereas the set of nodes forming the square is a cluster with density 2/3.

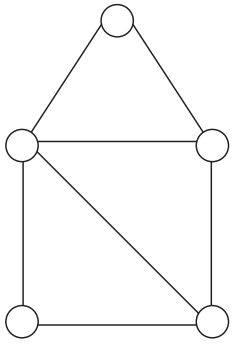


Figure 8.4 A small graph illustrating clusters with various densities.

Without going through the proof of this answer, the "if" direction of the proof is intuitively clear: a cluster of density p, all with state-0, will never see any of its node flip since the inertia from within the cluster suffices to avoid flipping. *Homophily* creates a blocking cluster in the network.

The second question is: if the entire network eventually flips, how many iterations will that take? And the third question is: if only part of the network flips

in the end, how big of a portion will flip (and where is it in the network)? These two questions are much harder to answer, and depend a lot on the exact network topology.

But perhaps the most useful question to answer for viral marketing is one of design: suppose each node in a given, known graph can be influenced at the initialization stage: if you pay node i $\$x_i$, it will flip. Presumably those nodes that perceive themselves as more influential will charge a higher price. Under a total budget constraint, which nodes would you buy (pay it to change its state and advertise that to neighbors) in order to maximize the extent of flipping at equilibrium, and furthermore, minimize the time it takes to reach that equilibrium?

While this question is hard to answer in general, some intuitions are clear. If you can only buy one node , it should be the most important one (by some centrality measure). But once you can buy two nodes, it is the *combined* influential power of the pair that matters. For example, you want the two nodes to be close enough to ensure some nodes will flip, but also want them to be far apart enough from each other so that more nodes can be covered. This tradeoff is further influenced by the heterogeneity of flipping thresholds: for the same cost, it is more effective to influence those eaiser to flip. Network topology is also important: you want to flip them in order to create a cascade so some nodes can help flip others.

8.2.4 Infection: Population based model

We have already seen five influence models between the last and this chapter. There is another model frequently used in modeling the spread of infectious disease. Compared to the other models, this one has a state transition, between two, three, or even more states that each node may find itself in. We will first describe the interaction using differential equations and assuming that each node can interact with any other node (which could have been introduced in the last chapter since this model does not depend on the topology). And then bring in network topology so that a node can only directly interact with nodes around it.

These variants of the infection model differ from each other based on the kind of state transitions are allowed. We will only cover 2-state and 3-state models. As shown in Figure 8.5, S stands for susceptible, I stands for infected, and R stands for recovered, and the symbols above the state transition arrows represent the rates of those transitions: how many switch per unit of time. We will use $S(t), I(t), R(t)$ to represent the *proportion* of population in that state at time t. The initial condition at time 0 is denoted as $S(0), I(0), R(0)$.

The first model is called **SI model**, and is very similar to the Bass model for diffusion in the last chapter, described by the following pair of differential equations, where the transition rates are proportional to the product $S(t)I(t)$ at

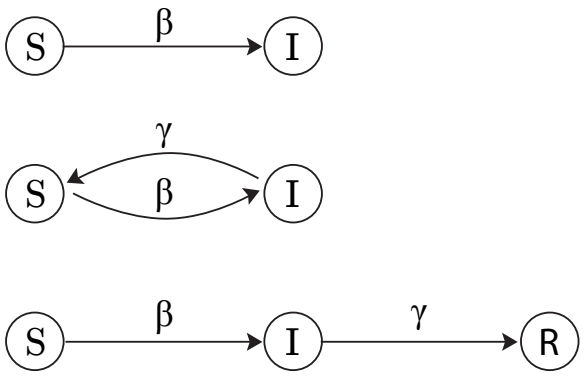


Figure 8.5 Three of the simplest state transition models for infection: SI model, SIS model, and SIR model. S stands for susceptible population, I infected, and R recovered. Each model can be mathematically described through a system of differential equations, some of which have analytic solutions while others need to be numerically solved. One caveat in this graphic representation of the differential equations: the β arrows indicate the transition rate that multiplies the product of the S and I populations, while the γ arrows indicate the transition rate that multiplies just the I population.

each time t:

$$\frac{dS(t)}{dt} = -\beta S(t)I(t) \tag{8.6}$$

$$\frac{dI(t)}{dt} = \beta S(t)I(t). \tag{8.7}$$

We could have also used just one of the two equations above and the normalization equation: all population proportions need to add up to 1: $S(t) + I(t) = 1$ at all times t. Substituting $S(t) = 1 - I(t)$ into (8.6), we have

$$\frac{dI(t)}{dt} = \beta(1 - I(t))I(t) = \beta(I(t) - I^2(t)).$$

This is a simple second-order differential equation just like what we used in the last chapter's Bass model. The closed-form solution is indeed a special case of the Bass model, a sigmoidal curve for the infected population over time, parameterized as a logistic growth equation:

$$I(t) = \frac{I(0)e^{\beta t}}{S(0) + I(0)e^{\beta t}}. \tag{8.8}$$

And of course, $S(t) = 1 - I(t)$.

We do not go through differential equation solvers here, but it is easy to *verify* through differentiation, that the above equation indeed matches the differential equations of SI model.

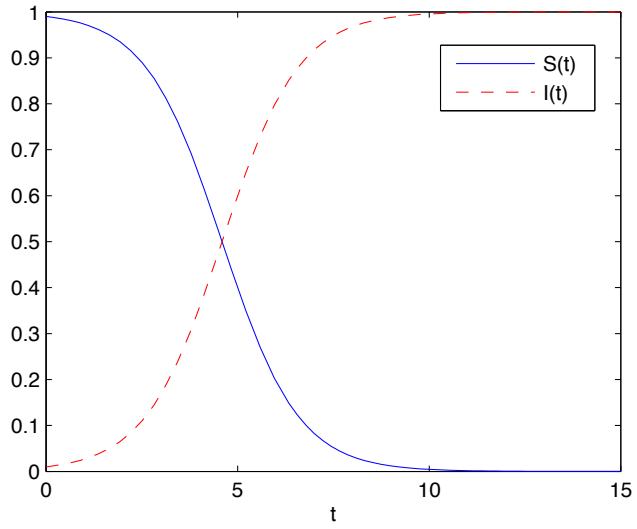


Figure 8.6 Population evolution over time for SI model. Eventually everyone is infected.

When t is small, the numerator dominates and $I(t)$'s growth is similar to exponential growth. When t becomes large, the ratio in (8.8) approaches 1. The whole curve is shown in Figure 8.6.

The SI model assumes that once infected, a person stays infected forever. In some diseases, a person can become non-infected but still remain susceptible to further infections. As in Figure 8.5, this **SIS model** is described by the following equations:

$$\frac{dS(t)}{dt} = \gamma I(t) - \beta S(t)I(t) \tag{8.9}$$

$$\frac{dI(t)}{dt} = \beta S(t)I(t) - \gamma I(t). \tag{8.10}$$

Without even solving the equations, we can guess that, if $\beta < \gamma$, the infected proportion dies out exponentially. If $\beta > \gamma$, we will see a sigmoidal curve of $I(t)$ going up, but not to 100% since some of the infected will be going back to the susceptible state. The exact saturation percentage of $I(t)$ depends on γ/β.

These intuitions are indeed confirmed in the closed-form solution. Again using $S(t) = 1 - I(t)$ and solving the resulting differential equation in $I(t)$, we have

$$I(t) = (1 - \gamma/\beta)\frac{ce^{(\beta-\gamma)t}}{1 + ce^{(\beta-\gamma)t}}, \tag{8.11}$$

for some constant c that depends on the initial condition. A sample trajectory, where $\beta > \gamma$, is shown in Figure 8.7.

Indeed, the growth pattern depends on whether $\beta > \gamma$ or not, and the $I(t)$ saturation level (as $t \to \infty$) depends on γ/β too. There is a name for this important constant

$$\sigma = \beta/\gamma,$$

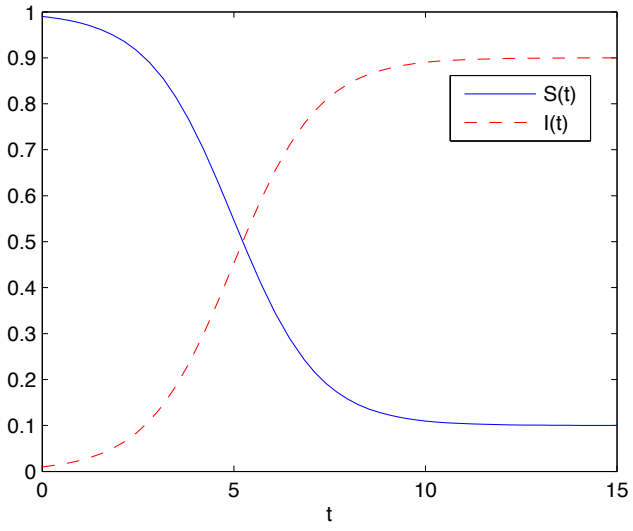


Figure 8.7 Population evolution over time for SIS model. At equilibrium, some people are not infected.

the **basic reproduction number**.

Both the SI and SIS models miss a common feature in many diseases: once infected and then recovered, a person becomes immunized. This is the R state. In the **SIR** model (not to be confused with the Signal to Interference Ratio in wireless networks in Chapter 1), one of the most commonly used, simple models for infection, the infected population eventually goes down to 0. As shown in Figure 8.5, the dynamics are described by the following equations:

$$\frac{dS(t)}{dt} = -\beta S(t)I(t) \tag{8.12}$$

$$\frac{dI(t)}{dt} = \beta S(t)I(t) - \gamma I(t) \tag{8.13}$$

$$\frac{dR(t)}{dt} = \gamma I(t). \tag{8.14}$$

Here, $\sigma = \beta/\gamma$ is the contact rate β (per unit time) times the average infection period $1/\gamma$. We can run substitution twice to eliminate two of the three equations above, but there is no closed-form solution to the resulting differential equation. Still, we can show that $\sigma S(0)$ plays the role of threshold level that determines whether $I(t)$ will go up first before coming down. The trajectory of this SIR model has the following properties over a period of time $[0, T]$:

- If $\sigma S(0) \leq 1$, then $I(t)$ decreases to 0 as $t \to \infty$. There is not enough $S(0)$ initially to create an epidemic.
- If $\sigma S(0) > 1$, then $I(t)$ increases to a maximum of

$$I_{max} = I(0) + S(0) - 1/\sigma - \log(\sigma S(0))/\sigma,$$

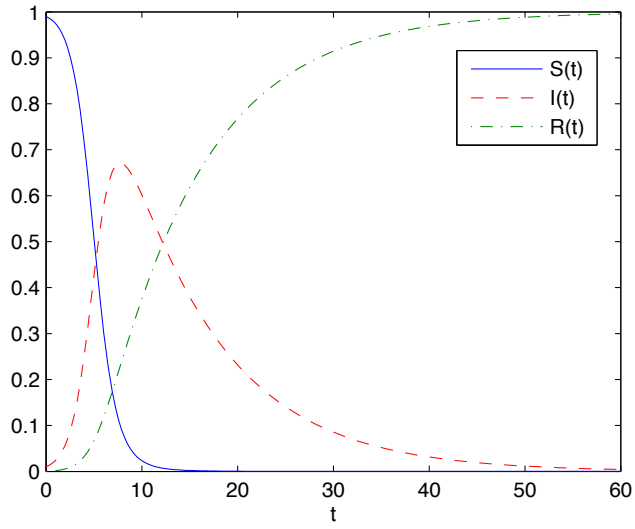


Figure 8.8 Population evolution over time in SIR model, for $\sigma S(0) > 1$. Eventually everyone is recovered.

then decreases to 0 as $t \to \infty$. This is the typical curve of an **epidemic** outbreak.

- $S(t)$ is always a decreasing function, the limit $S(\infty)$ as $t \to \infty$ is the unique root in $(0, 1/\sigma)$ of the following equation:

$$I(0) + S(0) - S(\infty) + \frac{1}{\sigma} \log\left(\frac{S(\infty)}{S(0)}\right) = 0.$$

A typical picture of the evolution is shown in Figure 8.8. Eventually everyone is recovered.

There are many other variants beyond the simplest three above, including the SIRS model where the recovered may become susceptible again, models where new states are introduced, and models where new births and deaths (due to infection) are introduced.

8.2.5 Infection: Topology based model

Up to this point, we have assumed that only the global population matters: each node feels the averaged influence from the entire network. This is sometimes called the *meanfield approximation* approach to enable mathematical analysis' tractability.

But in reality, of course infectious diseases spread only between two neighbors in a graph (however you may define the "link" concept for each disease). The difference between the contagion model and the infection model, as well as with random walk model, now boils down to how we model the local processing.

As illustrated in Figure 8.9: in contagion each node makes a deterministic decision of flipping or not (based on whether local influence is strong enough),

whereas in infection each node is imposed with a probabilistic "decision", the likelihood of catching the disease, with the rate of change of that likelihood dependent on the amount of local influence. So the discrete state actually turns into a continuous state representing the probability of finding a node in that state.

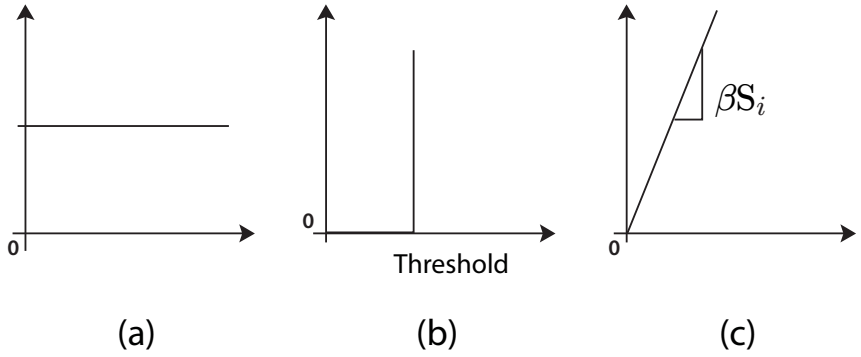


Figure 8.9 Local node processing models, the rate of change of node i's state (y-axis) vs. its neighbors' influence (x-axis), for (a) random walk, (b) contagion, and (c) infection. Contagion exhibits a flipping threshold, while infection has a linear dependence of rate of change on neighbor influence.

Intuitively, if the topology is such that there is a bottleneck subset of nodes, it will be harder to spread the disease to the entire network. More precisely, we need to include the adjacency matrix in the update equation.

For example, take the simplest case of SI model, now we have for each node i the following differential equation:

$$\frac{dS_i(t)}{dt} = -\beta \sum_j A_{ij}[S_i(t)I_j(t)] \tag{8.15}$$

$$\frac{dI_i(t)}{dt} = \beta \sum_j A_{ij}[S_i(t)I_j(t)]. \tag{8.16}$$

There are two tricky points in this seemingly simple translation from the original population-based model to this topology-based model. First, quantities S_i and I_i should be read as the *probability* of node i in state S or I, respectively. Second, it is tempting to pull S_i out of the summation over j, but actually we need to read S_iI_j as one quantity: the *joint* probability that node i is state S and its neighbor node j in state I. So the above notation is actually wrong. But to estimate this joint probability, we need to know the probability that some neighbor of node j (other than node i) was in state I while node j itself was in state S, for that is the only way we can get to the current state of i in S and j in

I. Following this line of reasoning, we have to enumerate all the possible paths of evolution of global states across the *whole network* over time. That is too much computation, and we have to stop at some level and approximate.

The first order of approximation is actually to break the joint probability exactly as in (8.15): the *joint* probability of node i in state S and node j in state I is approximated as the *product* of the individual probabilities of node i in state S and of node j in state I.

For many other network computation problems, from decoding over wireless channels to identifying people by voice recognition, it is a common technique to reduce computational load by breaking down *global* interactions to *local* interactions. For certain topologies like trees, a low order approximation can even be exact, or at least accurate enough for practical purposes.

With this first order approximation, we have the following differential equation for the SI model (which can also be readily extended to Bass model) with topology taken into account:

$$\frac{dI_i(t)}{dt} = \beta S_i(t) \sum_j A_{ij} I_j(t) \tag{8.17}$$

$$= \beta(1 - I_i(t)) \sum_j A_{ij} I_j(t). \tag{8.18}$$

The presence of quadratic term and of the adjacency matrix makes it difficult to solve the above equation in closed-form. But during the early times of the infection, $I_i(t)$ is very small, and we can approximate the equation as a linear one by ignoring $I_i(t)$. In vector form, it becomes:

$$\frac{d\mathbf{I}(t)}{dt} = \beta \mathbf{A}\mathbf{I}. \tag{8.19}$$

Here, $\mathbf{I}$ is not an identity matrix, but a vector of the probabilities of the nodes being in state I. We can, as in (8.1), decompose $\mathbf{I}$ as a weighted sum of eigenvectors $\{\mathbf{v}_k\}$ of $\mathbf{A}$, and the weights $w_k(t)$ solve the following linear, scalar, differential equation for each eigenvector k:

$$\frac{dw_k(t)}{dt} = \beta \lambda_k w_k(t),$$

giving rise to the solution

$$w_k(t) = w_k(0) e^{\beta \lambda_k t}.$$

Since $\mathbf{I}(t) = \sum_k w_k \mathbf{v}_k$, the solution to (8.19) is

$$\mathbf{I}(t) = \sum_k w_k(0) e^{\beta \lambda_k t} \mathbf{v}_k.$$

So the growth is still exponential at the beginning, but the growth exponent is weighted by the eigenvalues of the adjacency matrix $\mathbf{A}$ now.

Another approximation is to assume that all nodes of the same degree at the same time have the same S or I value. Like the order-based approximation above,

it is clearly incorrect, but useful to generate another tractable way of solving the problem.

So far we have assumed a detailed topology with an adjacency matrix given. An alternative is to take a generative model of topology that gives rise to features like small-world connectivity, and run infection models on those topologies. We will be studying generative models in the next two chapters.

We can also randomly pick a link in the given topology to be in an "open" state with probability p that a disease will be transmitted from one node to another, and in a "closed" state with $1 - p$. Then from any given initial condition, say, an infected node, there is a set of nodes connected to the original infected node through this maze of open links, and another set of nodes not reachable since they do not lie on the paths consisting of open links. This turns the infection model into the study of **percolation**.

8.3 Examples

8.3.1 Seeding a contagion

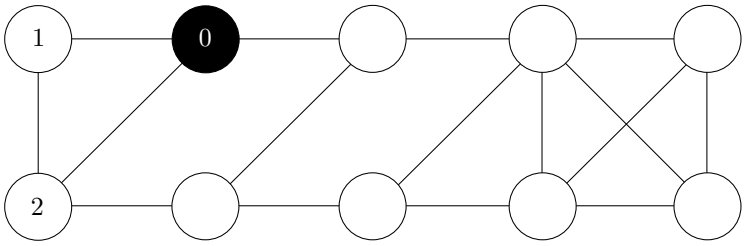


Figure 8.10 Example for contagion model with $p = 0.49$. One node is initialized to be at state-0, and the eventual number of flips is 3. The dark node with 0 written in it represents the initial seed at iteration 0. Nodes with numbers written in it are flipped through the contagion, and the number indicates the iteration at which each is flipped.

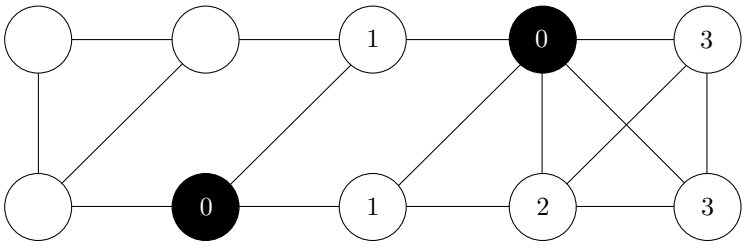


Figure 8.11 Example for contagion model with $p = 0.49$. Two nodes are initialized to be at state-0, and the eventual number of flips becomes 7.

Contagion depends on the topology. The graph in Figure 8.10 is obtained by repositioning 3 (out of 16) edges in the graph in Figure **??**. Suppose the same node is initialized as state-1, the number of eventual flips decreases sharply from 10 to 3. This shows how sensitive contagion outcome is with respect to network topology. We can also check that the density of the set of state-0 nodes is 2/3 after the left-most two nodes flip, which is higher than 1 minus the flipping threshold $p = 0.49$, thus preventing a complete flipping of the entire network.

Suppose now you want to stage a social media compaign by hiring twitter, facebook, and blog writers to spread their influence. Next consider the problem of buying, or seeding, nodes, assuming each node can be bought at the same price and the aim is to maximize the number of eventual flips. If we can buy off only one node, choosing the node highlighted in Figure 8.10 is actually the optimal strategy. If we can buy off two nodes, Figure 8.11 shows the optimal strategy and the eventual number of flips is 7, a significant improvement.

We also see that the nodes to be initialized as state-1 (seeded) in the two-node example do not include the one seeded in the one-node example. Optimal seeding strategies cannot be successively refined.

8.3.2 Infection: A case study

Just like the Netflix recommendation algorithm in Chapter 4 and the Bass diffusion model in Chapter 7, we actually do not know the model parameter values until we have some trial or historical data to train the model first.

In SIR model, we can observe through historical data for each disease the initial population $S(0)$ and final population $S(\infty)$ of susceptible people, and assume the initial infected population is negligible. This means we can approximate the key parameter σ as

$$\sigma = \frac{\log(S(0)/S(\infty))}{S(0) - S(\infty)}. \tag{8.20}$$

In public health decision making, a crucial one is the target vaccination rate for **herd immunity**: we want the immunized population to be large enough that infection does not go up at all, *i.e.*, $S(0) < 1/\sigma$, or,

$$R(0) > 1 - 1/\sigma. \tag{8.21}$$

That means the fraction of population with immunity, either through getting and recovering from the disease, or through vaccination (the more likely case), must be large enough. Using (8.20) to estimate σ, we can then estimate the vaccination rate $R(0)$ needed. These estimates are not exact, but at least they provide a sense of relative difficulty in controlling different infectious diseases.

Let us take a look at the case of measles. It causes about 1 million deaths worldwide each year, but in developed countries it is sufficiently vaccinated that it only affects very few. Its σ is quite large, estimated to be 16.67. By (8.21), this translates into a vaccination rate of 94% needed for herd immunity. But the

vaccine efficacy is not 100%, more like 95% for measles. So the vaccination rate needs to be 99% to achieve herd immunity. This is a very high target number, and can only be achieved through a two-dose program, more expensive than the standard single dose program.

When measles vaccination was first introduced in 1963 in the US, the measles infection population dropped but did not disappear: it stayed at around 50,000 a year. In 1978, the US government increased coverage of immunization in an attempt to eliminate measles, and infection population further dropped to 5,000, but still not near 0. In fact the number went back up to above 15,000 in 1989-1991. Just increasing the coverage of immunization did not make the immunization rate high enough. In 1989, the US government started using the two-dose program for measles: one vaccination at around 1 year old and another in around 5 years time. This time the immunization rate went up past the herd immunity threshold of 99% before children reach school age. Consequently, the number of reported cases of measles dropped to just 86 ten years later.

In a 2011 U.S. movie "Contagion" (we use "infection" for spreading of disease), the interactions among social networks, information networks, and disease spreading networks were depicted. Kate Winslet explained the basic reproduction number too (using the notation R_0, which is equivalent to σ for the cases we mentioned here). Some of the scenes in this drama actually occurred in real life during the last major epidemic, SARS, that started in China in 2003, including some governments suppressing news of the disease, some healthcare workers staying on their jobs despite very high basic reproduction number and mortality rate, and the speed of information transmission faster than that of disease transmission.

8.4 Advanced Material

8.4.1 Random walk

One influence model with network topology has already been introduced in Chapter 3: the pagerank algorithm. In that chapter, we wanted to see what set of numbers, one per node, is *self-consistent* on a directed graph: if each node spreads its number evenly across all its outgoing neighbors, will the resulting numbers be the same? It is a state of equilibrium in the influence model, where the influence is spread across the outgoing neighbors.

In sociology, the **DeGroot model** is similar, except that it starts with a *given* set of numbers $\mathbf{v}$, one per node (so the state of each node is a real number rather than a discrete one), and you want to determine what happens over time under the above influence mechanism.

The evolution of $\mathbf{x}(t)$, over time slots indexed by t, can be expressed as follows:

$$\mathbf{x}(t) = \mathbf{A}^t \mathbf{v}. \tag{8.22}$$

Here $\mathbf{A}$ is an influence relationship adjacency matrix. If $A_{ii} = 1$ and $A_{ij} = 0$ for all j, that means node i is an "opinion seed" that is not influenced by any other nodes.

We have seen this linear equation many times by now: power control, pagerank, centrality measures. It is also called **random walk on graph**. When does it converge? Will it converge on a subset of nodes (and the associated links)?

Following standard results in linear algebra, we can show that, for any subset of nodes that is **closed** (no link pointing from a node in the subset to a node outside), the necessary and sufficient conditions on matrix $\mathbf{A}$ for convergence are:

- *Irreducible*: an irreducible adjacency matrix means that the corresponding graph is connected: there is a directed path from any node to any other node.
- *Aperiodic*: an aperiodic adjacency matrix means that the lengths of all the cycles in the directed graph have the greatest common denominator of 1.

What about the *rate of convergence*? That is much harder to answer. But to the first order, an approximation is that the convergence rate is governed by the *ratio* of the second largest eigenvalue λ_2 and the largest eigenvalue λ_1. An easy way to see this is to continue the eigenvector centrality development (8.2). The solution to (8.22) can be expressed through the eigenvector decomposition $\mathbf{v} = \sum_i c_i \mathbf{v}_i$:

$$\mathbf{x}(t) = \mathbf{A}^t \sum_i c_i \mathbf{v}_i = \sum_i c_i \lambda_1^t \left(\frac{\lambda_i}{\lambda_1}\right)^t \mathbf{v}_i.$$

Dividing both sides by the leading term $c_1\lambda_1^t$, since we want to see how accurate is the first order approximation, and rearranging the terms, we have:

$$\frac{\mathbf{x}(t)}{c_1\lambda_1^t} = \mathbf{v}_1 + \frac{c_2}{c_1}\left(\frac{\lambda_2}{\lambda_1}\right)^t \mathbf{v}_2 + \ldots,$$

which means that the leading term of the *error* between $\mathbf{x}(t)$ and the first-order approximation $c_1\lambda_1^t\mathbf{v}_1$ is the second-order term in the eigenvector expansion, with a magnitude that evolves over time t proportional to:

$$\left(\frac{\lambda_2}{\lambda_1}\right)^t.$$

For certain matrices like the Google matrix $\mathbf{G}$, the largest eigenvalue is 1. Then it is the **second largest eigenvalue** that governs the rate of convergence.

As a small example, consider the network shown in Figure 8.12 consisting of two clusters with a link between them. Each node also has a self-loop. Suppose the state of each node is a score between 0 and 100, and the initial score vector is

$$\mathbf{v} = \begin{bmatrix}100 & 100 & 100 & 100 & 0 & 0 & 0\end{bmatrix}^T$$

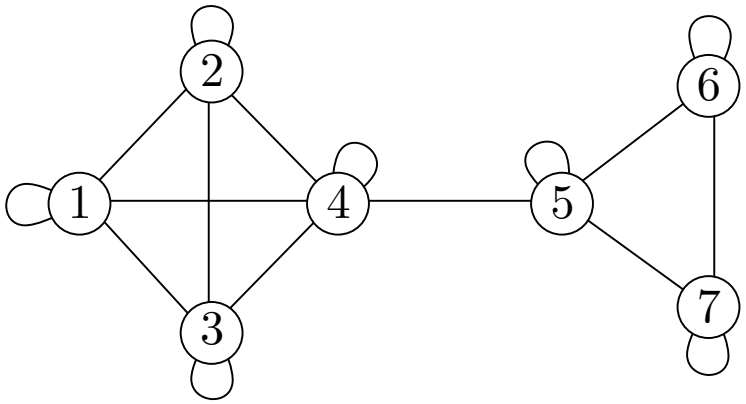


Figure 8.12 Example for DeGroot model and contagion model. The initial scores on nodes 1-4 eventually spread evenly to all nodes. The rate of convergence to this equilibrium is governed by the second largest eigenvalue of the adjacency matrix.

i.e., all nodes in the left cluster have an initial score of 100, and the right cluster has an initial score of 0.

From the network we can also write out $\mathbf{A}$:

$$\mathbf{A} = \begin{bmatrix} 1/4 & 1/4 & 1/4 & 1/4 & 0 & 0 & 0 \\ 1/4 & 1/4 & 1/4 & 1/4 & 0 & 0 & 0 \\ 1/4 & 1/4 & 1/4 & 1/4 & 0 & 0 & 0 \\ 1/5 & 1/5 & 1/5 & 1/5 & 1/5 & 0 & 0 \\ 0 & 0 & 0 & 1/4 & 1/4 & 1/4 & 1/4 \\ 0 & 0 & 0 & 0 & 1/3 & 1/3 & 1/3 \\ 0 & 0 & 0 & 0 & 1/3 & 1/3 & 1/3 \end{bmatrix}.$$

Then we iterate the equation

$$\mathbf{x}(t) = \mathbf{A}\mathbf{x}(t-1)$$

to obtain

$$\begin{aligned} \mathbf{x}(0) &= \begin{bmatrix} 100 & 100 & 100 & 100 & 0 & 0 & 0 \end{bmatrix}^T \\ \mathbf{x}(1) &= \begin{bmatrix} 100 & 100 & 100 & 80 & 25 & 0 & 0 \end{bmatrix}^T \\ \mathbf{x}(2) &= \begin{bmatrix} 95 & 95 & 95 & 81 & 26.25 & 8.333 & 8.333 \end{bmatrix}^T \\ \mathbf{x}(3) &= \begin{bmatrix} 91.5 & 91.5 & 91.5 & 78.45 & 30.98 & 14.31 & 14.31 \end{bmatrix}^T \\ &\vdots \\ \mathbf{x}(\infty) &= \begin{bmatrix} 62.96 & 62.96 & 62.96 & 62.96 & 62.96 & 62.96 & 62.96 \end{bmatrix}^T. \end{aligned}$$

We see that the network is strongly connected, *i.e.*, $\mathbf{A}$ is irreducible. The existence of self-loops ensures the network is also aperiodic. The scores at equilibrium

are biased towards the initial scores of the left cluster because it is the larger cluster.

8.4.2 Measuring subgraph connectedness

We have finished our tour of 7 influence models in two chapters. We conclude this cluster of chapters with a discussion of what constitutes a group in a network.

Intuitively, a set of nodes form a group if there is a lot of connections (counting links or paths) among them, but not that many between them and other sets of nodes. Suppose we divide a graph into two parts. Count the number of links between the two parts; call that A. Then for each part of the graph, count the total number of links with at least one end in that part, and take the smaller of this number from the two parts; call that B. A different way to divide the graph may lead to a different ratio A/B. The smallest possible A/B is called the **conductance** of this graph. It is also used to characterize convergence speed in random walk.

To dig deeper into what constitutes a group, we need some metrics that quantify the notion of connectedness in a **subgraph**: a subset of nodes, and the links that start or end with one of these nodes. First of all, let us focus on end-to-end connectedness, *i.e.*, the existence paths. We say a subset is a **connected component**, or just a **component**, if each node in the subset can reach any other node in the subset through some path. For directed graphs, the path needs to be directed too, and it is called a **strongly connected component**. We can also further strengthen the notion of component to **k-component**: a maximal subset of nodes in which each node can be connected to any other node through not just one path, but k node-disjoint paths.

As is typical in this subsection, to make the definitions useful, we refer to the *maximal* subset: you cannot add another node (and the associated links) to the subset and still satisfy the definition. Then a component is the largest subset in which every possible node pair is connected by some path. In almost all our networks, there is just one component: the entire network itself.

What if we shift our attention from end-to-end path connectedness to direct, one-hop connectedness? We call a maximal subset of nodes a **clique** if every node in the subset is every other node's neighbor, *i.e.*, there is a link between every pair of nodes in the subset. It is sometimes called a **full mesh**.

A clique is very dense. More likely we will encounter a cluster of density p as defined before in the analysis of the contagion model: a maximal subset of nodes in which each node has at least $p\%$ of its neighbors in this subset.

In between a component and a clique, there is a middle ground. A **k-clique** is a maximal subset of nodes in which any node can reach any other node through no more than k links. If these k links are also in between nodes belonging to this subset, we call the subset a **k-club**.

We have so far assumed that geographic proximity in a graph reflects social distance too. But that does not have to be the case. Sometimes, we have a system

of labeling nodes by some characteristics, and we want a metric quantifying the notion of **associative mixing** based on this labeling: that nodes which are alike tend to associate with each other.

Consider labeling each node in a given graph as belonging to one of M given types, *e.g.*, M social clubs, M undergraduate majors, or M dorms. We can easily count the number of links connecting nodes of the same type. From the given adjacency matrix $\mathbf{A}$, we have

$$\sum_{ij \in \text{same type}} A_{ij}. \tag{8.23}$$

But this expression is not quite right to use for our purpose. Some nodes have large degrees anyway. So we have to calibrate with respect to that. Consider an undirected graph, and pick node i with degree d_i. Each of its neighbors, indexed by j, has a degree d_j. Let us pick one of node i's links. What is the chance that on the other end of a link is node j? That would be d_j/L, where L is the number of links in the network. Now multiply by the number of links node i has, and sum over node pairs (ij) of the same type:

$$\sum_{ij \in \text{same type}} \frac{d_i d_j}{L}. \tag{8.24}$$

The difference between (8.23) and (8.24), normalized by the number of links L, is the **modularity** Q of a given graph (with respect to a particular labeling of the nodes):

$$Q = \frac{1}{L} \sum_{ij \in \text{same type}} \left(A_{ij} - \frac{d_i d_j}{L} \right). \tag{8.25}$$

Q can be positive, in which case we have *associative* mixing: people of the same type connect more with each other (relative to a random drawing). It can be negative, in which case we have *dissociative* mixing: people of different types connect more with each other. With the normalization by L in its definition, we know $Q \in [-1, 1]$.

Still using the same graph in Figure 8.2, we consider a labeling of the nodes into two types. In the following, nodes enclosed in dashed lines belong to type 1, and the remaining nodes belong to type 2.

Obviously the degrees are

$$\mathbf{d} = \begin{bmatrix} 3 & 3 & 4 & 3 & 2 & 2 & 3 & 2 \end{bmatrix}^T.$$

We also have

$$L = 11 \times 2 = 22,$$

since the links are undirected. In the computation of modularity, all pairs (ij) (such that $ij \in$ same type) are considered, which means every undirected edge is counted twice, thus the normalization constant L should be counted the same way.

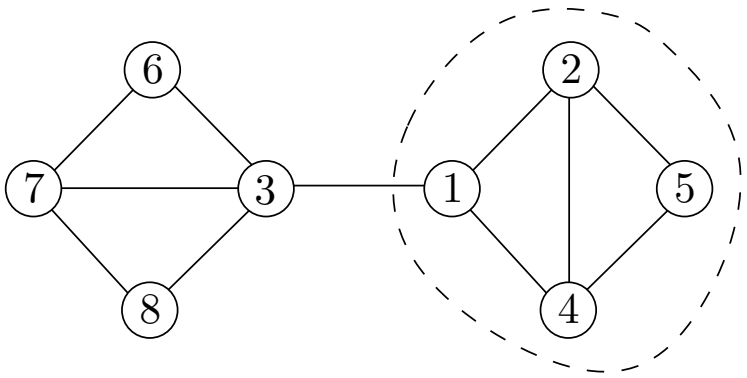


Figure 8.13 An associative labeling of the example network. Nodes close to each other are labeled into the same type, and modularity Q is positive.

Now consider associative mixing with the grouping in Figure 8.13. The modularity can be expressed as

$$\begin{aligned} Q =& \frac{1}{L} \sum_{ij \in \text{same type}} \left(A_{ij} - \frac{d_i d_j}{L} \right) \\ =& \frac{1}{L} \sum_{ij} S_{ij} \left(A_{ij} - \frac{d_i d_j}{L} \right), \end{aligned}$$

where index S_{ij} denotes whether i and j are of the same type as specified by the labeling. We can also collect these indices into a $(0, 1)$ matrix:

$$\mathbf{S} = \begin{bmatrix} 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 \end{bmatrix}.$$

Given $\mathbf{S}, \mathbf{A}$ and $\mathbf{d}$, we can compute Q. We see the modularity value is indeed positive for this labeling system, and reasonably high:

$$Q = 0.5413.$$

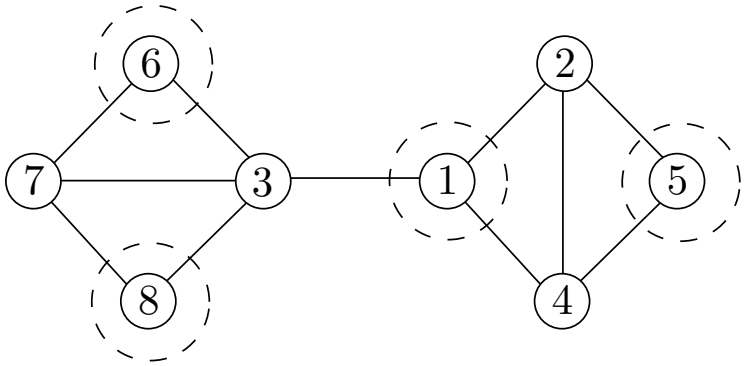


Figure 8.14 A disassociative labeling of the example network. Nodes close to each other are labeled into different types, and modularity Q is negative.

But suppose the labeling is changed to what is shown in Figure 8.14, *i.e.*,

$$S = \begin{bmatrix} 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \end{bmatrix}.$$

Then the modularity value is negative, as we would expect from dissociative mixing:

$$Q = -0.2025.$$

8.4.3 Graph partition and community detection

It is actually not easy to infer either the network topology or the traffic pattern from limited (local, partial, and noisy) measurement. In the last part of this chapter, we assume someone has built an accurate topology already, and our job is to detect the non-overlapping communities, or subgraphs, through some centralized, off-line computation.

The easiest version of the problem statement is that we are given the number of communities and the number of nodes in each community. This is when you have a pretty good prior knowledge about the communities in the first place. It is called the **graph partition** problem. We will focus on the special case where we partition a given graph into two subgraphs, the **graph bisection** problem, with a fixed target size (number of nodes) in each subgraph. The input is a graph $G = (V, E)$, and the output is two sets of nodes that add up to the original node set V.

How do we even define that one graph partition is better than another? One metric is the number of links between the two subgraphs, called the **cut size**. Later you will see the max flow min cut theorem in routing in Chapter 13. For now, we want to find a bisection that minimizes the cut size.

An algorithm that is simple to describe although heavy in computational load is the **Kernighan Lin algorithm**. There are two loops in the algorithm. In each step of the outer loop indexed by k, we pick any initialization of bisection: graphs $G_1[k]$ and $G_2[k]$. To initialize the first outer loop, we put some nodes in subgraph $G_1[1]$ and the rest in the other subgraph $G_2[1]$.

Now we go through an inner loop, where at each step we pick the pair of nodes (i, j), where $i \in G_1$ and $j \in G_2$, such that swapping them reduces the cut size most. If cut size can only be increased, then pick the pair such that the cut size increases by the smallest amount. After each step of the inner loop, the pair that has been swapped can no longer be considered in future swaps. When there are no more pairs to consider, we complete the inner loop and pick the configuration $(G_1^*[k], G_2^*[k])$ (*i.e.*, which nodes belong to which subgraph) with the smallest cut size $c^*[k]$.

Then we take that configuration $(G_1^*[k], G_2^*[k])$ as the initialization of the next step $k+1$ in the outer loop. This continues until cut size cannot be decreased further through the outer loops. The configuration with the smallest cut size:

$$\min_k \{c^*[k]\}$$

is the bisection returned by this algorithm.

More often, we do *not* know how many communities there are, or how many nodes are in each. That is part of the job of **community detection**. For example, you may wonder about the structure of communities in the graph of Facebook connections. And we may be more interested in the richness of connectivity within each subgraph than the sparsity of connectivity between them. Again, we focus on the simpler case of two subgraphs (G_1, G_2), but this time not imposing the number of nodes in each subgraph a priori.

Modularity is an obvious metric to quantify how much more connected a set of nodes is relative to the connectivity if links were randomly established among the nodes. Modularity is defined with respect to a labeling system. Now there are two labels, those belonging to G_1 and those to G_2.

So we can simply run the Kernighan-Lin algorithm again. But instead of picking the pair of nodes to *swap* across G_1 and G_2 in order to minimize the cut size, now we select one node to *move* from G_1 to G_2 (or the other way around), in order to maximize the modularity of the graph.

A different approach gets back to the cut size minimization idea, and tries to disconnect the graph into many pieces by deleting one link after another. This is useful for detecting not just two communities, but any number of communities. Which link to delete first? A greedy heuristic computes the betweenness metric of all the links (8.5), and then deletes the link with the highest betweenness value. If

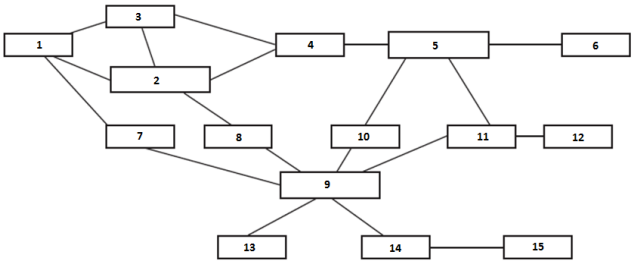


Figure 8.15 The original graph of the Florence families, viewed as a single community.

that does not break the graph into two subgraphs, then compute the betweenness values of all the remaining links, and delete the link with the highest value again. Eventually, this process will break the graph and give you 2 subgraphs (and $3, 4, \ldots, N$ graphs as you keep deleting links).

As an example, we consider the Renaissance Florence family graph again, shown in Figure 8.15.

Now we run the following community detection algorithm:

1. From graph $G = (V, E)$, find the adjacency matrix $\mathbf{A}$.
2. Compute the betweenness of all links $(i, j) \in E$ from $\mathbf{A}$.
3. Find the link $(i, j)^*$ that has the highest betweenness value. If more than one such link exists, select one of these randomly.
4. Remove link $(i, j)^*$ from G. Check to see if any communities have been detected, and return to step 1.

The betweenness values of all the links in the initial graph are shown in Table 8.4.3

The largest betweenness is that of link (9,14): 26. As a result, it will be eliminated from the graph. The result is shown in Figure 8.16 below. As one can see, removing (9,14) has created two distinct communities: the first is the node set $V_1 = \{1, ..., 13\}$, and the second is the set $V_2 = \{14, 15\}$.

Next, the adjacency matrix is modified according to the new graph, and the betweenness values are calculated again. The results are shown in Table **??**.

The largest betweenness is that of link (4,5): 17.5. As a result, it will be eliminated from the graph. However, removing this link does not extract additional communities, so the process is repeated. Running betweenness with (4,5) eliminated, the maximum is that of link (8,9): 25.5. Again, this link is eliminated. Finally, after running the procedure again, link (7,9) is found to have the highest

Link	Betweenness	Link	Betweenness
(1,2)	5.0000	(5,11)	17.0000
(1,3)	6.0000	(7,9)	19.0000
(1,7)	13.0000	(8,9)	18.1667
(2,3)	4.0000	(9,10)	15.5000
(2,4)	9.5000	(9,11)	23.0000
(2,8)	13.5000	(9,13)	14.0000
(3,4)	8.0000	(9,14)	**26.0000**
(4,5)	18.8333	(11,12)	14.0000
(5,6)	14.0000	(14,15)	14.0000
(5,10)	12.5000	–	–

Table 8.1 Betweenness values of the links from the initial graph. Link (9,14) has the largest betweenness and will be eliminated first.

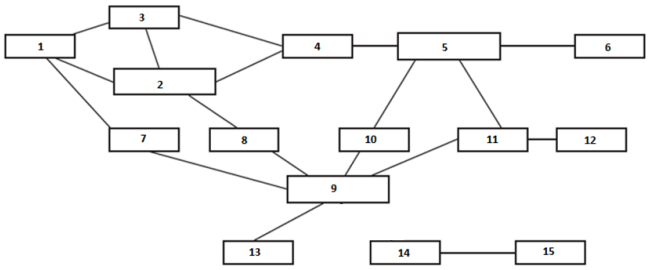


Figure 8.16 Eliminating the link with the highest betweenness, (9,14), detects two communities within the graph: Node sets $V_1 = \{1, ..., 13\}$ and $V_2 = \{14, 15\}$.

betweenness value of 42 and is eliminated. This separates the graph into three communities: $V_1 = \{1, 2, 3, 4, 7, 8\}$, $V_2 = \{5, 6, 9, 10, 11, 12\}$, and $V_3 = \{14, 15\}$, as shown in Figure 8.17.

In addition to modularity maximization and betweenness-based edge removal, there are several other algorithms for community detection, including graph Laplacian optimization, maximum likelihood detection, and latent space modeling. When it comes to a large scale community detection problem in practice, it remains unclear which of these will be most helpful in attaining the eventual goal of detecting communities and remain robust to measurement noise.

Instead of deleting links, we can also *add* links from a set of disconnected nodes. This way of constructing communities is called **hierarchical clustering**.

Now, finding one pair of similar nodes is easy, for example, by using node

Link	Betweenness	Link	Betweenness
(1,2)	5.0000	(5,11)	14.3333
(1,3)	5.0000	(7,9)	14.0000
(1,7)	10.0000	(8,9)	12.5000
(2,3)	3.0000	(9,10)	10.8333
(2,4)	8.8333	(9,11)	16.3333
(2,8)	9.8333	(9,13)	12.0000
(3,4)	8.0000	(9,14)	–
(4,5)	**17.5000**	(11,12)	12.0000
(5,6)	12.0000	(14,15)	1.0000
(5,10)	9.8333	–	–

??

Table 8.2 Betweenness values of the links from the two-component graph. Link (4,5) has the largest betweenness and will be eliminated.

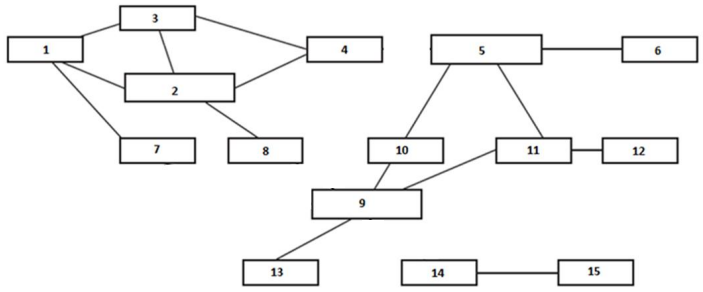


Figure 8.17 Eliminating links (4,5), (8,9), and (7,9) detects three communities: Node sets $V_1 = \{1, 2, 3, 4, 7, 8\}$, $V_2 = \{5, 6, 9, 10, 11, 12\}$, and $V_3 = \{14, 15\}$.

similarity metrics like cosine coefficient in Chapter 4. The difficulty is in defining a consistent and useful notion of similarity between two *sets* of nodes, based on a particular similaritiy metric between two nodes.

If there are N_1 nodes in G_1 and N_2 nodes in G_2, there are then N_1N_2 node pairs. Therefore, there are N_1N_2 similarity metric values. We needd to scalarize this long vector. We can take the largest, smallest, average, or any scalar representation of this vector of values as the similarity metric between the two sets of nodes.

Once a similarity metric is fixed and a scalarization method is picked, we can hierarchically run clustering by greedily adding nodes, starting from a pair of nodes until there are only two groups of nodes left.

Further Reading

Similar to the last chapter, there is a gap between the rich foundation of graph theory and algorithms and the actual operation of Facebook and Twitter and their third party service providers.

1. The standard reference on contagion models is the following one
[Mor00] S. Morris, "Contagion," *Review of Economic Studies*, vol. 67, pp. 57-78, 2000.

2. Our discussion of infection models follows the comprehensive survey article below:
[Het00] H. W. Hethcote, "The mathematics of infectious diseases," *SIAM Review*, vol. 42, no. 4, pp. 599-653, 2000.

3. A classical work on innovation diffusion, both quantitative models and qualitative discussions, can be found in the following book:
[Rog03] E. M. Rogers, *Diffusion of Innovation*, 5th Ed., Free Press, 2003.

4. There are many additional approaches developed and questions asked about social influence. A recent survey from the field of political science is the following paper:
[Sie09] D. Siegel, "Social networks and collective action," *American Journal of Political Science*, vol. 53, no. 1, pp. 122-138, 2009.

5. Many graph theoretic quantities on node importance, link importance, and group connectedness can be found in the following textbook:
[New10] M. E. J. Newman, *Networks: An Introduction*, Oxford University Press, 2010.

Problems

8.1 *Computing node centrality* ⋆

Compute the degree, closeness and eigenvector centralities of all nodes in the graph in Figure 8.18.

8.2 *Computing betweenness* ⋆

Refer to the graph in Figure 8.18.

(a) Compute the node betweenness centrality of nodes 2 and 3.

(b) Compute the link betweenness centrality of the edges/links $(3, 4)$ and $(2, 5)$.

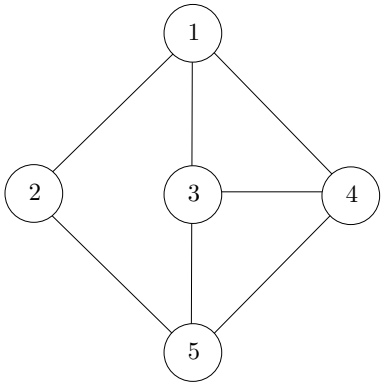


Figure 8.18 A simple network for computing centrality measures.

8.3 *SIRS infection model* ⋆⋆

We consider an extension to the SIR model that allows nodes in state R to go to state S. This model, known as the SIRS model, accounts for the possibility that a person loses the acquired immunity over time.

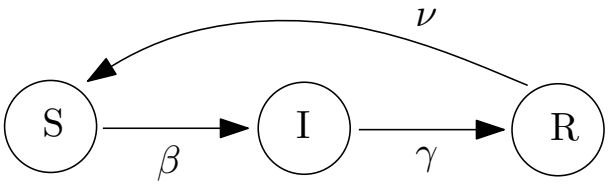


Figure 8.19 The state transition diagram for the SIRS infection model.

Consider the state diagram in Figure 8.19. We can write out the set of differential equations as

$$\begin{aligned}\frac{dS(t)}{dt} &= -\beta S(t)I(t) + \nu R(t)\\ \frac{dI(t)}{dt} &= \beta S(t)I(t) - \gamma I(t)\\ \frac{dR(t)}{dt} &= \gamma I(t) - \nu R(t).\end{aligned}$$

Modify the Matlab code for the numerical solution of the SIR model and solve for $t = 1, 2, \ldots, 200$ (set the `tspan` vector in code accordingly) with the following parameters and initial conditions: $\beta = 1$, $\gamma = 1/3$, $\nu = 1/50$, $I(0) = 0.1$, $S(0) = 0.9$, $R(0) = 0$. Briefly describe and explain your observations.

8.4 *Contagion* ⋆

Consider the contagion model being run in the graph in Figure 8.20 with $p = 0.3$.

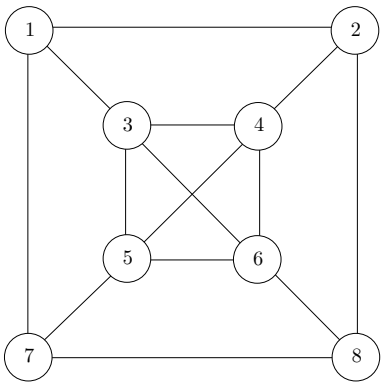


Figure 8.20 A simple network for studying contagion model.

(a) Run the contagion model with node 1 initialized at state-1 and the other nodes initialized at state-0.

(b) Run the contagion model with node 3 initialized at state-1 and the other nodes initialized at state-0.

(c) Contrast the results from (a) and (b) and explain in terms of the cluster densities of the sets of initially state-0 nodes.

8.5 *Networked sampling* ⋆⋆⋆

In sociology, estimating the percentage of a hidden population, *e.g.*, AIDS infected population, is difficult. One approach is to start with a few sampled "seeds", and then ask current sample members to recruit future sample members. The question is how to produce unbiased estimates.

Respondent driven sampling is a method to address this question, and it is used by many institutions including the US Center for Disease Control and UNAIDS. The basic methodology is random walk on graphs, similar to what we saw in this chapter and in Chapter 2.

(a) First let us characterize the sampling distribution at equilibrium. Let π_i be the stationary distribution of reaching person i.

Let K_{ij} be the probability of person i referring to person j. If each bidirectional link (i, j) has a weight $w_{(i,j)}$, and the probability of person i recruits person j is directly proportional to $w_{(i,j)}$, we have a recruiting mechanism similar to Google

pagerank's spread of importance score:

$$P[i \to j] = \frac{w_{(i,j)}}{\sum_k w_{(i,k)}}.$$

At equilibrium, what is the probability π_i that person i has been sampled?

(b) We follow a trajectory of sampling, starting with, say, one person, and runs through n people sampled. If a person i on the trail of sampling has AIDS, we add the counter of the infected population by 1. If we simply add these up and divide by n, it gives a biased estimate. The importance sampling method weight each counter by $1/(N\pi_i)$. But we often do not know the value of N. So in respondent-driven sampling, the estimate becomes:

$$(\text{Harmonic mean of } \pi_i) \sum_{\text{infected } i} \frac{1}{\pi_i}.$$

Suppose we have two social groups, A and B, of equal size forming a network, and that the infection rates are p_A and p_B, respectively. Between groups, links have weights c, where $c \in (0, 0.5)$. Within each group, links have weights $1 - c$.

If we follow respondent driven sampling, what do you think intuitively will happen? Confirm this with a simulation.

(For more details, see S. Goel and M. Salganik, "Respondent-driven sampling as Markov chain Monte Carlo", *Statistics in Medicine*, vol. 28, pp. 2202-2229, 2009.)

9 Can I really reach anyone in 6 steps?

In the last two chapters, we saw the importance of topology to functionality. In this and the next chapters, we will focus on **generative models** of network topology and reverse-engineering of network functionality. These are mathematical constructions that try to *explain* widespread empirical observations about social and technological networks: the small world property and the scale free property. We will also highlight common misunderstanding and misuse of generative models.

9.1 A Short Answer

Since Milgram's 1967 experiment, the **small world** phenomenon, or the **six degrees of separation**, has become one of the most widely known stories in popular science books. Milgram asked 296 people living in Omaha, Nebraska to participate in the experiment. He gave each of them a passport-looking letter, and the destination was in a suburb of Boston, Massachusetts, with the recipient's name, address, and occupation (stock broker) shown. Name and address sound obvious, and it turned out that it was very helpful to know the occupation. The goal was to send this letter to one of your friends, defined as someone you knew by first name. If you did not know the recipient by first name, you had to send the letter via others, starting with sending it to a friend (one hop), who then sent it to one of her friends (another hop), until the letter finally arrived at someone who knew the recipient by first name and sent it to the recipient.

Of these letters, 217 were actually sent out and 64 arrived at the destination, a seemingly small percentage of 29.5% but actually quite impressive, considering that a later replica of the experiment via email only had a 1.5% arrival rate. The other letters might have been lost along the way, and needed to be treated carefully in the statistical analysis of this experiment's data. But out of those 64 that arrived, the average number of hops was 5.2 and the median 6, as shown in Figure 9.2.

Researchers have long suspected that the **social distance**, the average number of hops of social relationships it takes (via a short path) to reach anyone in a population, grows very slowly as the population size grows, often logarithmically. Milgram's celebrated experiment codified the viewpoint. From the 1970s to the

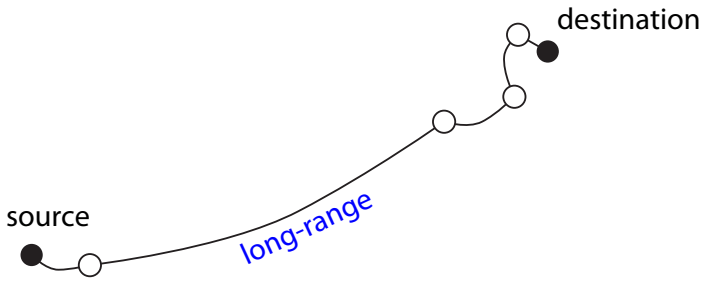


Figure 9.1 A picture illustrating the Milgram experiment in 1967. A key phenomenon is that there is one or two long range links in these short paths between Omaha and Boston. It turns out that they substantially reduced the shortest and the searchable path lengths without reducing the clustering coefficient significantly in the social network.

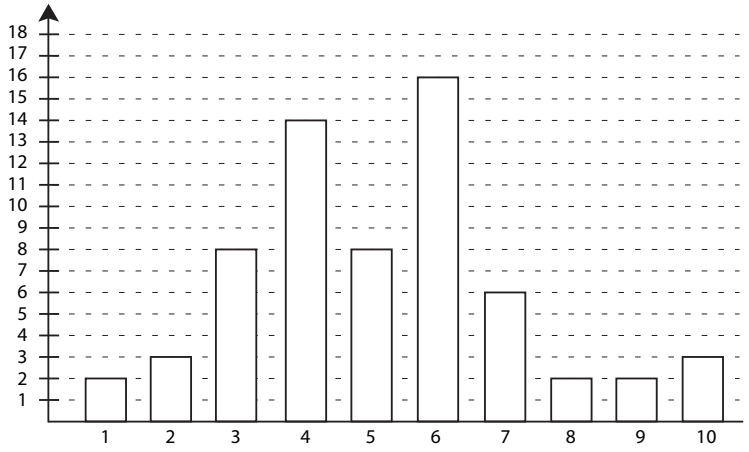


Figure 9.2 The histogram of the length of the chains from different sources in Omaha to the common destination in Boston in Milgram's experiment. The median value is 6, leading to the famous 6 degree separation observation.

online social media era, much empirical evidence suggested the same: from Erdos number among mathematicians to co-starring relationships in IMDB.

Should we be surprised by this seemingly universal observation of social networks? There are two issues here, echoing the dichotomy between topology and functionality we saw in Chapter 8.

- One is *structural*: there are short paths in social networks.
- Two is *algorithmic*: with very limited local information a node can navigate through a social network and find a short path to a given destination.

The second is more surprising than the first and requires more careful modeling of the functionality of **social search**. For example, a report in November 2011 computed the degrees of separation on Facebook to be 4.74. That only concerned with the existence of short paths, not the more relevant and more surprising discoverability of short paths from local information. As we will see, it is also more difficult to create a robust explanation to the observation of an algorithmic small world.

But first, we focus on the existence of short paths. On the surface, it seems fascinating that you can reach anyone likely in 6 steps or less. Then, on second thought, you may reason that, if I have 20 friends, and each of them has 20 friends, then in 6 steps, I can reach to 20^6 people. That is already 64 million people. So of course 6 steps often suffice.

But then, on an even deeper thought, you realize that social networks are filled with"triangles," or **triad closures**, of social relationships. This is illustrated in Figure 9.3: if Alice and Bob both know Chris, Alice and Bob likely know each other directly too. This is called **transitivity** in a graph (not to be confused with transitivity in voting). In other words, the catch of the 20^6 argument above is that you need your friend's 20 friends to not overlap with your own set of 20 friends. Otherwise, the argument fails. But of course, many of your friend's friends are your own friends too. There is a lot of overlap. The phenomenon of people who are alike tend to form social links is called **homophily**, and it can be quantified by the **clustering coefficient** as discussed later. Now, six degrees of separation is truly surprising.

Milgram-type experiments suggest something even stronger: not only are there short paths, but they can be discovered by each individual node using very limited information about the destination and its local view of the network. Compared to routing packets through the Internet, *social search* in this sense is even harder since nodes do not pass messages around to help each other construct some global view of the network topology. That help is implicitly embedded in the address and occupation of the recipient, and possibly the name that can reveal something about the destination's sex and ethnicity. Some kind of **distance metric** must have been constructed in each person's mind throughout Milgram's experiment. For example, New York is closer to Boston than Chicago is, on the geographic proximity scale measured in miles. Or, a financial adivser is perhaps closer to a stock broker than a nurse is, on some occupation proximity scale, which is more vague but nonetheless can be grossly quantified. Suppose each person uses a simple, "greedy" algorithm to forward the letter to her friend who is closest to the destination, where "closeness" is defined by a composite of these kinds of scales. Is it a coincidence this social search strategy discovers a short path?

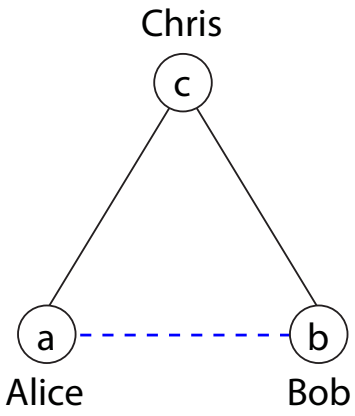


Figure 9.3 An illustration of triad closure in social networks. If Alice knows Chris and Bob knows Chris, it is likely that Alice and Bob also know each other. If so, the connected triple forms a triangle. The clustering coefficient quantifies the ratio between connected triples and triangles in a graph.

We will walk through several models that can address the above issues.

9.2 A Long Answer

9.2.1 Structural small worlds: Short paths

There are several ways to measure how "big" a (connected) graph is. One is **diameter**: it is the length of the longest shortest path between any pair of nodes. Here, "shortest" is with respect to all the paths between a given pair of nodes, and "longest" is with respect to all possible node pairs.

When we think of a network as small world, however, we tend to use the *median* of the shortest paths between all node pairs, and look at the growth of that metric as the number of nodes increases. If it grows on the order of the log of the number of nodes, we say the network is (structurally) small world.

Suppose we are given a fixed set of n nodes, and for each pair of nodes decide with probability p if there is a link between them. This is the basic idea of a **Poisson random graph**, or the **Erdos Renyi model**.

Of course this process of network formation does not sound like most real networks. It turns out that it also does not provide the same structures we encounter in many real networks. For example, while in a random graph, the length of the average shortest path is small, it does not have the right **clustering coefficient**. For a proper explanatory model of small world networks, we need the shortest path to be small and the clustering coefficient to be large.

What is the clustering coefficient? Not to be confused with the density of a

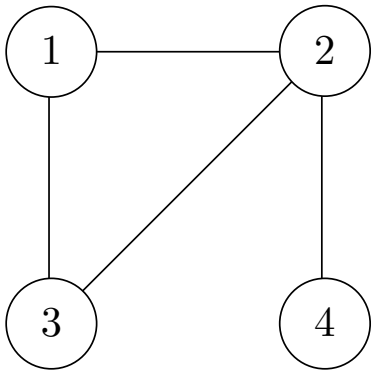


Figure 9.4 A small example for the cluster coefficient calculation. $C = 3/5$ in this graph.

cluster from Chapter 7, it is a metric to quantify the notion of triad closure. As in Figure 9.3, we define a set of three nodes (a, b, c) in an undirected graph as a **connected triple**, if there is a path connecting them:

- If there are links $(a, b), (b, c)$ (whether there is a link (a, c) or not), we have a connected triple (a, b, c).
- Similarly, if there are links $(a, c), (c, b)$, we have a connected triple (a, c, b).
- If there are links $(b, a), (a, c)$, we have a connected triple (b, a, c).

But if there are links (ab, bc, ca), we have not just a connected triple, but also a triangle. We call this a **triad closure**.

The clustering coefficient C summarizes the above countings in the graph:

$$C = \frac{\text{Number of triangles}}{\text{Number of connected triples}/3}. \tag{9.1}$$

The division by 3 normalizes the metric, so that a triangle's clustering coefficient is exactly 1, and that $C \in [0, 1]$.

For example, consider the toy example in Figure 9.4. It has one triangle $(1, 2, 3)$, and the following five connected triples $(2, 1, 3)$, $(1, 2, 3)$, $(1, 3, 2)$, $(1, 2, 4)$, $(3, 2, 4)$. Hence, its clustering coefficient is

$$C = \frac{1}{5/3} = \frac{3}{5}.$$

It is easy to see that the *expected* clustering coefficient of a random graph is

$$C = \frac{c}{n-1},$$

where c is the *average* degree of a node and n the total number of nodes. This is because the probability of any two nodes being connected is $c/(n-1)$ in a random graph, including the case when the two nodes are known to be indirectly connected via a third node. So if there are 100 million people, and each has 1000 friends, the clustering coefficient is about 10^{-3}, way too small for a realistic social network.

What about a very regular ring like in Figure 9.5 instead? This **regular graph**

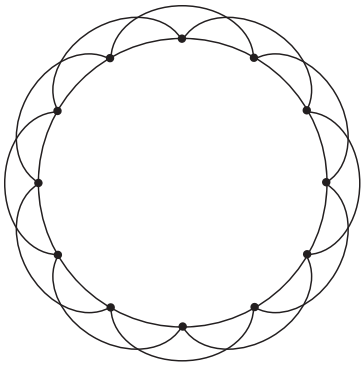


Figure 9.5 An example of a regular graph, with n nodes living on a ring and each having c links. In this example, $n = 12$ and $c = 4$. Each node has 2 links pointing to the right and 2 to the left. The clustering coefficient C is independent of n and grows as c becomes larger.

is parameterized by an even integer c: the number of neighbors each node has. Because all the nodes "live" on a ring, each node can have $c/2$ number of left-pointing links to its closest $c/2$ neighbors, and similarly $c/2$ on the right.

It is easy to see that the clustering coefficient is large for a regular graph.

- To form a triangle, we need to go along one direction on the ring two steps, then take one step back towards where we started. And the farthest we can go along the ring in one direction and still be back in one hop is $c/2$. So, the number of triangles starting from a given node is simply $c/2$ choose 2, the number of distinct choices of picking two nodes out of $c/2$ of them. This gives us $\frac{1}{2}\frac{c}{2}(\frac{c}{2} - 1)$ triangles per node.
- On the other hand, the number of connected triples centered on each node is just c choose 2, *i.e.*, $\frac{1}{2}c(c-1)$.

So the clustering coefficient is:

$$C = \frac{\frac{1}{2}\frac{c}{2}(\frac{c}{2} - 1)}{(\frac{1}{2}c(c-1))/3} = \frac{3(c-2)}{4(c-1)}. \tag{9.2}$$

It is independent of the number of nodes n. This makes intuitive sense since the graph is symmetric: every node looks like any other node in its neighborhood topology.

- When $c = 2$, the smallest possible value of c, we have a circle of links, so there are obviously no triangles. Indeed, $C = 0$.
- But as soon as $c = 4$, we have $C = 1/2$, which is again intuitively clear and shown in another way by drawing a regular ring graph with $c = 4$ in Figure 9.6.
- When c is large, the clustering coefficient approaches the largest it can get on

a regular ring topology: 3/4. This is many orders-of-magnitude larger than that of a random graph.

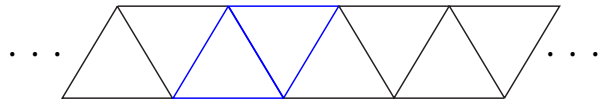


Figure 9.6 Another way to draw a regular ring graph with $c = 4$. This visualization clearly shows that half of the triad closures are there in the graph, and C should be 0.5.

The regular (ring) graph model stands in sharp contrast to the random graph model: it has a high clustering coefficient C, which is realistic for social networks, but has a large value of the (median or average, across all node pairs) shortest path's distance L, since there are only short-range connections. Random graph networks are exactly the opposite: small L but also small C. If only we could have a "hybrid" graph that combines both models to get a small L and a large C on the same graph.

That is exactly what the **Watts Strogatz model** accomplished in 1998. It is the canonical model explaining small world networks with large clustering coefficients. As shown in Figure 9.7, it has two parameters:

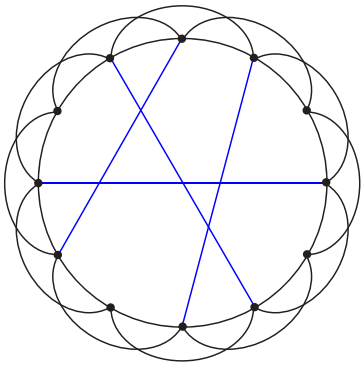


Figure 9.7 An example of the Watts Strogatz model. It adds random links, possibly long-range ones, to a regular ring graph. When there is a short-range link (a link in the regular ring graph), there is now a probability p of establishing a long-range link randomly connecting two nodes. The resulting graph has both a large clustering coefficient C and a small expected shortest distance L.

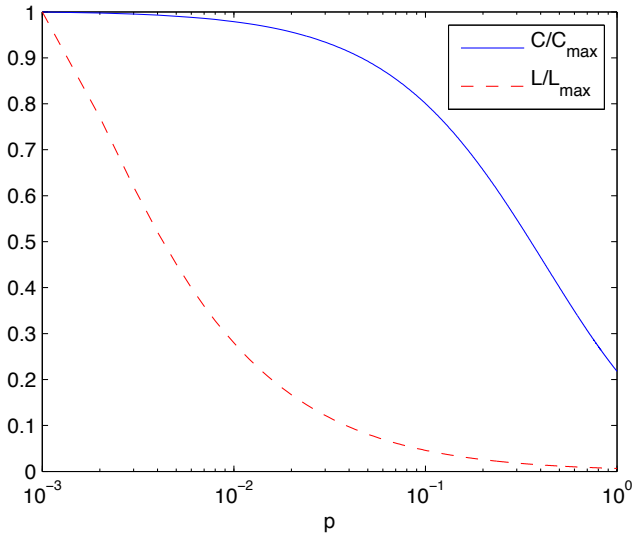


Figure 9.8 A numerical example of the impact of p on C and L in the Watts Strogatz model. When p is small, like 0.01, C is almost the same as in a regular ring graph but L is substantially smaller.

- c is the number of nearest neighbors each node has.
- p is the probability that any pair of nodes, including those far apart, are connected by a link whenever there is a short-range link.

Actually, the Watts Strogatz model deletes one regular link for each of the random links added. The model we just showed does not include the deletion step. But the main analysis for our purpose remains effectively the same.

The key point is that with the additional links, we get to preserve the large clustering coefficient of a regular graph while achieving the small world effect. With just a little randomization p, the expected shortest path's distance can be reduced substantially.

How much randomization do we need? While we postpone the detailed calculation of C and L to Advanced Material, it suffices to summarize at this point that, as long as p is small, *e.g.*, 0.1, its impact on the clustering coefficient is almost negligible. Yet the average shortest path distance behaves like a small world: it grows in the order of $\log n$. It fundamentally changes the order of growth of L with respect to the number of nodes in the network. This is illustrated in Figure 9.8. Fixing $n = 600$ and $c = 6$, we plot C/C_{max} and L/L_{max} against p, where C_{max} and L_{max} are their respective maxima over all computed values. A large C and small L is the first definition of a small world network.

Where does this asymmetry in p's impact come from? Fundamentally it has to do with the very definition of our metrics:

- Shortest path is an *extremal* quantity: we only care about the *shortest* path, there is no need to reduce all the paths' lengths. Just add a few long-range links, and even add them randomly, and then the shortest path will be much shorter.
- In contrast, clustering coefficient is an *average* quantity: it is defined as the average number of triangles involving a node divided by the average number of connected triples centered at the node. So adding a small proportion of non-triangular, connected triples does not hurt the clustering coefficient that much.

There lies the magic of small world with a large clustering coefficient: we have triad closure relationships with most of our friends, but a very small fraction of our friends are outside our normal social circle. All Milgram needed to observe six degrees of separation was that very small fraction of long-range links.

However, this begs a deeper question: to be "fair" with respect to how we define clustering coefficients, why not define a small world network as one where, for most node pairs, the *average*, not just the shortest, path length between these two nodes is small? "Average" here refers to the average over all the paths between a given pair of nodes. Well, we believe that for social search, we do not *need* that stringent a definition. The existence of some short paths suffices. This means we implicitly assume the following: each node can actually *find* a short path with a hop count not much bigger than that along the shortest path. How *that* can be accomplished with only local information per node is a deeper mystery than just the existence of short paths.

Before we move on to look at models explaining this, we should mention that a little randomization also showed up in Google pagerank in Chapter 3, and a locally dense but globally sparse graph will be a key idea in constructing near-optimal peering graphs in P2P content distribution in Chapter 15.

9.2.2 Algorithmic small worlds: Social search

If you were one of those people participating in Milgram's experiment, or one of those along the paths initiated by these people, how would you decide the next hop by just looking at the destination's name, address, and occupation? You probably would implicitly define a metric that can measure distance, in terms of both geographic proximity (relatively easy) and occupational proximity (relatively hard). And then you would look at all your friends whom you know by first name, and pick the one closest to the destination in some combination of these two distances. This is a **greedy social search**, with an average length (*i.e.*, hop count) of l. And we wonder if it can discover very short paths: can l be close to L? One could also have picked the friend with the highest degree, and we will soon reach the hub. But we will not consider this or other alternative strategies.

Compared to IP routing in the Internet in Chapter 13, social search is even

harder, since people do not pass messages to tell each other exactly how far they are from the destination. But here, we are not asking for an exact guarantee of discovering the shortest path either.

There have been several models for social search in the past decade beyond the original Watts Strogatz model. The first was the **Kleinberg model**, and was defined for any dimension d. We examine the 1-dimensional case now, like a ring with n nodes, which is really a line wrapped around so that we can ignore edge effect. Links are added at random as in the Watts Strogatz model. But the random links are added with a refined detail in the Kleinberg model. The probability of having a random link of length r is proportional to $r^{-\alpha}$, where $\alpha \geq 0$ is another model parameter. The longer the link, the less likely it will show up. When $\alpha = 0$, we are back to the Watts Strogatz model.

It turns out that only when $\alpha = 1$ will the small world effect appear: l is upper bounded by $\log^2 n$. It is not quite $\log n$, but at least it is an upper bound that is a polynomial function of $\log n$.

When $\alpha \neq 1$, l grows as a polynomial of n, and thus is not a small world. This is even under the assumption that each node knows the exact locations of the nodes it is connected to, and therefore the distances to the destination.

In the homework, we will see that the exponent α needs to be exactly two for a network where nodes live on a 2-dimensional rectangular grid. Generally, the exponent α must be exactly the same as the dimension of the space where the network resides. The intuition is clear: in k-dimensional space, draw spheres with radius $(r, 2r, \ldots)$ around any given node. The number of nodes living in the space between a radius-r sphere and radius-$2r$ sphere is proportional to r^k. But according to the Kleinberg model, the probability of having a link to one of those nodes also drops as r^{-k}. These two terms cancel each other out, and the probability of having *some* connection d hops away becomes independent of d. This independence turns out to give us the desired **searchability** as a function of n: l grows no faster than a polynomial function of $\log n$.

The underlying reasoning is therefore as follows: the chance of you having a friend outside of your social/residential circle gets smaller and smaller as you go farther out, but the number of people also becomes larger and larger. If the two effects cancel each other out, you will likely be able to find a short path.

A similar argument runs through the **Watts, Dodds, Newman model**. In this hierarchical model shown in Figure 9.9, people live in different "leaf nodes" of a binary tree that depicts the geographic, occupational, or their combination's proximity. If two leaf nodes, A and B, share the first common ancestry node that is lower than that shared by A and C, then A is closer to B than to C.

For simplicity, let us assume each leaf node is a group of g people. So there are n/g leaf nodes (for notational simplicity, assume this is an integer), and $\log_2(n/g)$ levels in the binary tree. For example, if there are $n = 80$ people in the world, and $g = 10$ people live in a social circle where they can reach each other directly, we have a $\log(80/10) = 3$-level tree.

In this model, each person A measures distance to another person B by count-

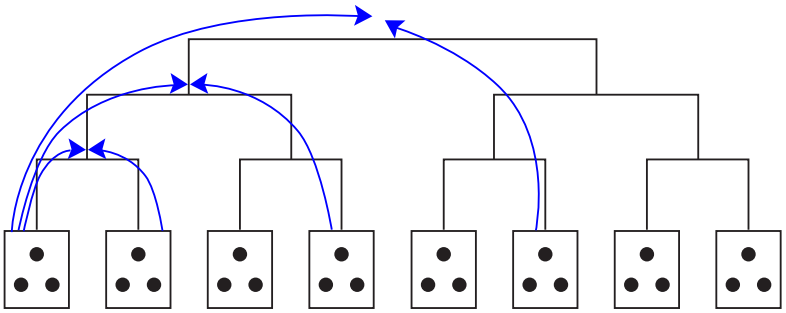


Figure 9.9 An example of Watts Dodds Newman's model. The 3-level hierarchy is represented through a binary tree. Each leaf node of this tree has a population of g people who are directly connected to each other. The probability that two people know each other decays exponentially with m, the level of their first common ancestry node.

ing the number of tree levels it needs to go up before finding the first common ancestry node between A and B. We also assume that the probability that two people know each other decays *exponentially* with m, the level of their first common ancestry node, with an exponent of α:

$$p_m = K2^{-\alpha m}, \tag{9.3}$$

where K is a normalization constant so that $\sum_m p_m = 1$. You probably can recognize the similarity between this way of assigning probabilities and the random link probabilities in Kleinberg's model. Indeed, the results are similar.

The *expected* number of people that a person can be connected to through her m-th ancestry, denoted as N_m, is clearly the product of two numbers: $g2^m$, the number of people connectable through the m-th ancestry, and p_m, the probability that the level of first common ancestry is indeed m:

$$N_m = g2^{1\times m}p_m.$$

We highlight the fact that the number of levels in a tree grows exponentially in m and the exponent of this growth is 1. This turns out to be the mathematical root cause for the main result below.

Plugging in the formula for p_m (9.3), we have:

$$N_m = gK2^{(1-\alpha)m}. \tag{9.4}$$

In passing the message toward the destination, if the first common ancestry level shared by a node with the destination is m, the expected number of hops it needs to pass the message (before getting to someone who is on the same side of

the m-th hierarchy as the destination) is simply $1/N_m$. Summing over all levels m, we determine the expected length of the path by greedy search is

$$l = \sum_m \frac{1}{N_m} = \frac{1}{Kg} \sum_{m=0}^{\log(n/g)-1} 2^{\alpha-1} m = \frac{1}{Kg} \frac{(n/g)^{\alpha-1} - 1}{2^{\alpha-1} - 1}. \tag{9.5}$$

But we cannot directly use (9.5) yet, because there is a normalization constant K that depends on n/g. We need to express it as a function of observable quantities in the tree. Here is one approach: summing N_m in (9.4) over all m levels, we get the average degree $\bar{d}$ of a person:

$$\bar{d} = \sum_m N_m = \sum_{m=0}^{\log(n/g)-1} Kg 2^{(1-\alpha)m} = Kg \frac{(n/g)^{1-\alpha} - 1}{2^{1-\alpha} - 1}. \tag{9.6}$$

Now we express K in terms of $(\bar{d}, g, n, \alpha)$ from (9.6), and plug it back into (9.5):

$$l = \frac{1}{\bar{d}} \frac{(n/g)^{\alpha-1} - 1}{2^{\alpha-1} - 1} \frac{(n/g)^{1-\alpha} - 1}{2^{1-\alpha} - 1}. \tag{9.7}$$

Since we want to understand the behavior of l as a function of n when n grows, we take the limit of (9.7) as n becomes large.

- If $\alpha \neq 1$, by (9.7), clearly l as a function of n grows like $(n/g)^{|\alpha-1|}$, a polynomial in n.
- If $\alpha = 1$, we can go back to the formula of N_m in (9.4), which simplifies to just $N_m = gK$ independent of the level m. This independence is the critical reason why the small world property now follows: it becomes straightforward to see that l now grows like $\log^2(n/g)$, *i.e.*, the square of a logarithm:

$$\frac{1}{Kg} \sum_{m=0}^{\log(n/g)-1} 1 = \frac{1}{g} \log(n/g) \frac{1}{K} = \frac{1}{\bar{d}} \log^2(n/g).$$

One interpretation of the above result is as follows: since a binary tree is really a 1-dimensional graph, we need α to be exactly the same as the number of dimensions to get an algorithmic small world.

This condition on α makes these algorithmic small world's explanatory models brittle, in contrast to the robust explanatory model of Watts and Strogatz where p can be over a range of values and still lead to a structural small world. In our three dimensional world, α does not always equal 3, and yet we still observe algorithmic small worlds. In Advanced Material, we will summarize an alternative, less brittle explanatory model of algorithmic small world.

9.3 Examples

We numerically illustrate the social search model by Watts Dodds and Newman.

Fix $g = 100$ (as in the original paper), $\bar{d} = 100$ (a number close to Dunbar's

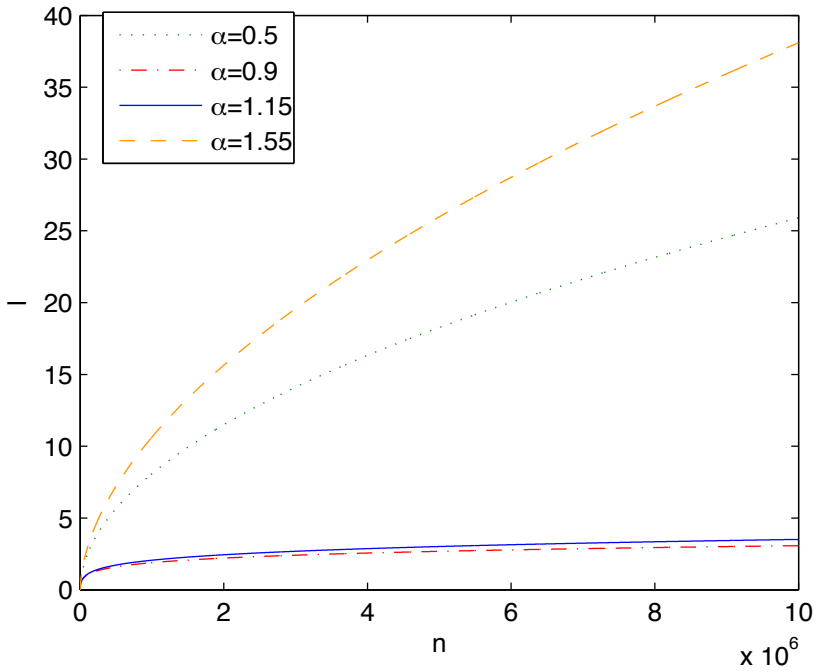


Figure 9.10 Impact of n in Watts Dodds Newman model. When the population grows from 1 million to 10 million, a factor of two increase on log scale, the length of greedy social search's path barely increases, as long as α is close to 1. But when α is a little farther from 1, l increases almost 20-fold.

number). Figure 9.10 shows how l grows with an increasing population n. Then Figure 9.11 shows the scaling behavior of l with respect to n for different values of α. l grows much more slowly when α is right around 1.

9.4 Advanced Material

9.4.1 Watts Strogatz model: Clustering coefficient and shortest path length

Consider the Watts Strogatz model, with n nodes, c short-range links per node, and probability p of long-range link. We wan to first count C and then approximate L.

The number of triangles remains the same whether it is before or after the long-range links are added. But there are more connected triples thanks to the long-range links:

- The expected number of long-range links is $\frac{1}{2}ncp$, since for each node's short-range links, there is a probability p of establishing a long-range link, and

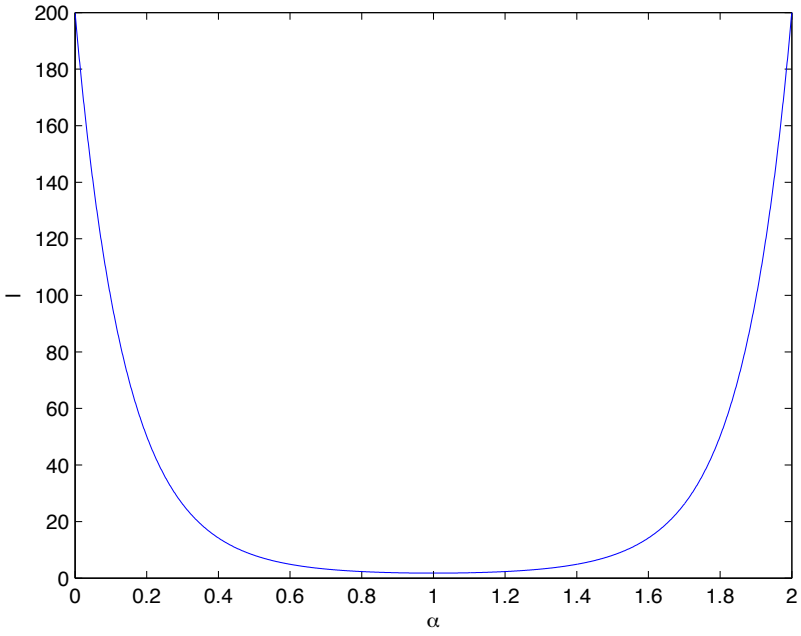


Figure 9.11 Impact of α in Watts Dodds Newman model. The value of n is 1 million. When α is very close to 1, the dimension of the binary tree, l remains very small. But as α moves away from 1, l quickly becomes very large.

there are nc such opportunities altogether. The factor $1/2$ just avoids double counting, since each link has two end points. Each of these long-range links, together with the c short-range links per node, can form two connected triples. So we multiply $\frac{1}{2}ncp$ by $2c$, and this gives rise to nc^2p connected triples.

- The long-range links themselves can also create connected triples similar to the short-range links, with an expected total number of $\frac{1}{2}nc^2p^2$, since starting from each of the n nodes, we need two long-range links to be there and the probability of that is c^2p^2.

So the clustering coefficient now becomes:

$$C = \frac{\frac{n}{2}\frac{c}{2}(\frac{c}{2}-1)}{\left(\frac{n}{2}c(c-1)+nc^2p+\frac{1}{2}nc^2p^2\right)/3} = \frac{3(c-2)}{4(c-1)+8cp+4cp^2}. \tag{9.8}$$

Staying with the Watts Strogatz model, the derivations of L are too technically involved for inclusion here. But the following is one of the possible approximations when ncp is large:

$$L \approx \frac{\log(ncp)}{c^2p}. \tag{9.9}$$

We can clearly see that for small p, $C(p)$ behaves like $1/p$. But $L(p)$ is almost constant since $\log p$ is like p and cancels p in the denominator when p is small. This is the mathematical representation of our discussion before: C becomes large but L is small as we add just a few long-range links.

9.4.2 Generalized Watts Strogatz model: search path length

We have argued intuitively why the dimension of the space in which the graph lives must be equal to the rate of exponential decay in the probability of having a long-range link. But they must be *exactly* the same. Maybe if the probability decay model is not exactly exponential, we will not need α to be exactly the same as the space dimension.

It turns out that we do not need p to decay like $r^{-\alpha}$ (for some definition of distance r) in order to get searchability and an algorithmic small world. Back to the Watts Strogatz model, where α is effectively 0. Long-range links are created independent of distance, and follow some probability distribution, *e.g.*, binomial, geometric, Poisson, or Pareto (which we will discuss when introducing scale free network in the next chapter). This we call a **Generalized Watts Strogatz model**.

For this model, we can show that l is small. In fact, we can show this through an analytic recursive formula for the entire distribution of search path lengths (not just the average), and for any general metric space (not just a ring or grid). For example, consider a ring network with n nodes and c short-range links per node. Suppose each long-range link is independently established, and the number of long-range connections is drawn from Poisson distribution with rate λ. If the source-destination distance is between kc and $(k+1)c$ for some integer k, the expected search length for such a source-destination pair is

$$l_k = 1 + \sum_{j=1}^{k-1} \prod_{i=1}^{j} \exp(-\lambda(1-\beta_i)M), \tag{9.10}$$

where M is the largest number of long-range links, and

$$\beta_i = \frac{\pi - ic/2n}{\pi - c/2n}.$$

What happens in this model, and in empirical data from experiments, is that short-range links are used, and l increases linearly as the distance between the source and destination rises but remains small. When this distance becomes sufficiently large, long-range links start to get used, often just 1 or 2 of them and often at the early part of the social search paths. Then l quickly saturates and stays at about the same level even as the source-destination distance continues to rise.

We have seen 4 small world models. In the end, it is probably a combination of the Generalized Watts Strogatz model and the Kleinberg model that matches reality the best, where we have a nested sequence of social circles, with

distance-dependent link formation within each circle and distance-independent link formation across the circles.

Further Reading

There are many interesting popular science books on six degrees of separation, including *Six Degrees*, *Linked*, *Connected*, and many more. The analytic models covered in this chapters come from the following papers.

1. The experiment by Travers and Milgram was reported in the following heavily cited paper:
[TM69] J. Travers and S. Milgram, "An experimental study of the small world problem," *Sociometry*, vol. 32, no. 4, pp. 425-443, 1969.

2. The Watts Strogatz model is from the following seminal paper explaining the six degrees of separation:
[WS98] D. J. Watts and S. H. Strogatz, "Collective dynamics of small-world networks," *Nature*, vol. 393, pp. 440-442, 1998.

3. The Kleinberg model explaining the short length of social search's path is from the following paper:
[Kle00] J. M. Kleinberg, "The small world phenomenon: An algorithmic perspective," *Proceedings of ACM Symposium on Theory of Computing*, 2000.

4. The related Watts Dodds Newman model is from the following paper:
[WDN02] D. J. Watts, P. S. Dodds, and M. Newman, "Identity and search in social networks," *Science*, vol. 296, pp. 1302-1304, 2002.

5. The following recent paper presented Generalized Watts Strogatz model (called Octopus Model in the paper) that explains short length of social search's path without imposing the condition of α equal to the dimension d of the space that the graph resides in.
[ICP12] H. Inaltekin, M. Chiang, and H. V. Poor, "Delay of social search on small-world random geometric graphs," *Journal of Mathematical Sociology*, 2012.

Problems

9.1 *Computation of C and L* $\star$

(a) Manually compute C and L for the graph in Figure 9.12.

(b) Compute the clustering coefficient C and the average shortest path length L for the two graphs in Figure 9.13. There is no need to use any code, and you

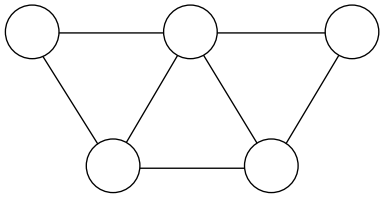


Figure 9.12 A simple graph for computing network measures.

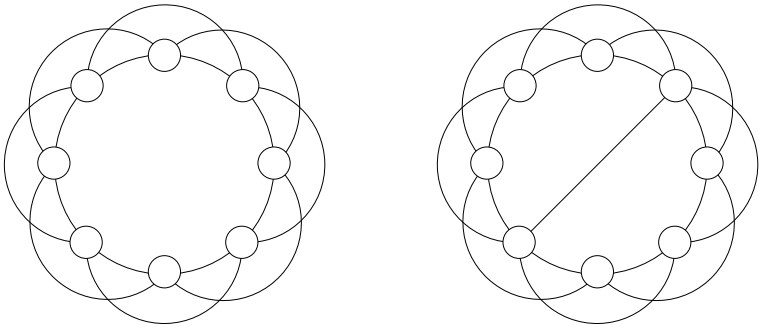


Figure 9.13 The Watts-Strogatz model with $n = 8$, $c = 4$.

can just easily calculate these by hand. Contrast their values between the graphs.

9.2 *Generalization of Triadic Closure* ⋆⋆

We have seen the definition of the clustering coefficient, which quantifies the amount of triadic closure. In general, "closure" refers to the intuition that if there are many pairwise connections among a set of nodes, there might be connection for any pair in the set as well. There is no specific reason to limit closure to only node triples as in triadic closure.

Here we consider the simple extension called "quad closure". As shown in Figure 9.14: if node pairs (a, b), (a, c), (a, d), (b, c), (b, d) are linked, then the pair (c, d) is likely to be linked in the future.

To quantify the amount of quad closure, we define a "quad clustering coefficient" as

$$Q = \frac{\text{Number of cliques of size 4}}{\text{Number of connected quadruples with 5 edges}/k}$$

where k is some normalizing constant. But this definition is incomplete unless we specify the value of k to normalize Q. Find the value of k such that the value of Q for a clique is exactly 1.

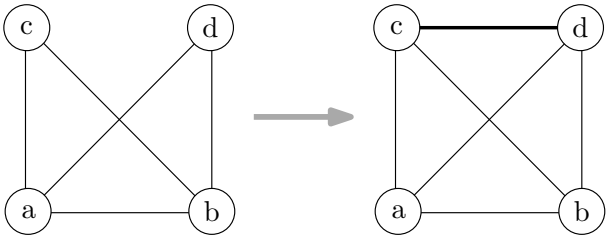


Figure 9.14 Quad closure: for nodes a, b, c and d if currently 5 out of all 6 possible links exist (Left), then the remaining link is likely to appear in the future (Right).

9.3 *Metrics of class social graph* ⋆⋆

In a class graph, where each student is a node, and a link between A and B means that A and B know each other on a first name basis before coming to this class. Download an anonymized class graph from
`http://scenic.princeton.edu/network20q/hw/class_graph.graphml`
and use your favorite software (*e.g.*, NodeXL, gephi, Matlab toolboxes), or by hand, to:

(a) Compute C and L.
(b) Compute eigenvector centrality.
(c) Partition the graph into communities.
Attach a few screenshots to show the results.

9.4 *Kleinberg model* ⋆⋆

In this question, $\log(\cdot)$ is in base 2 and $\ln(\cdot)$ is in base e.

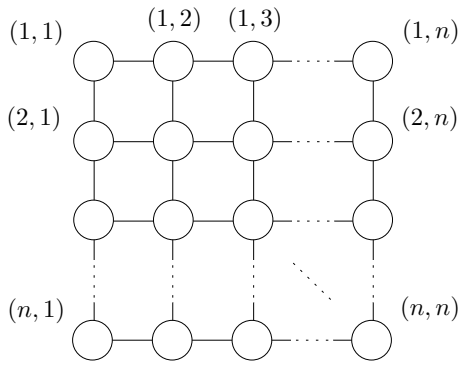


Figure 9.15 A 2D lattice to illustrate the Kleinberg model.

Consider the 2D lattice in Figure 9.15 with n^2 nodes labelled in 2D coordinates $(1,1),(1,2),\ldots,(n,n)$. The distance between two nodes $u = (x_1, y_1)$ and

$v = (x_2, y_2)$ is $d(u, v) = |x_1 - x_2| + |y_1 - y_2|$.

(a) What is the number of nodes of distance 1 from node $(1, 1)$, excluding itself? How about of distance 2, 3? And of general distance i, where $1 \leq i \leq n$?

(b) Let B_j be the set of nodes of distance at most 2^j from node $(1, 1)$, excluding itself. Calculate the size of B_j for $0 \leq j \leq \log n$. Use the summation identity

$$\sum_{k=1}^{r} k = \frac{k(k+1)}{2}.$$

(c) Let R_j be the set of nodes contained in B_j but outside B_{j-1}, *i.e.*, $R_j = B_j \setminus B_{j-1}$ for $j \geq 1$ and $R_0 = B_0$. Calculate a lower bound on the size of R_j. Specifically, if you obtain something of the form

$$2^s + 2^t$$

with $s > t > 0$, you should report 2^s as your answer.

(d) Let $\alpha = 2$. By the Kleinberg model, given two nodes u and v are separated by distance r, the probability of the nodes being linked by a random link is lower bounded as follows:

$$\Pr(u \leftrightarrow v) \geq \frac{r^{-\alpha}}{4\ln(6n)}.$$

Given that node $(1, 1)$ has only one random link, use the result from (c) to lower bound the probability that node $(1, 1)$ has a random link to a node in R_j, $\Pr(u \leftrightarrow R_j)$, for $0 \leq j \leq \log n$. Does the answer depend on the value of j?

(e) What happens to the result in (d) if $\alpha = 1$ or $\alpha = 3$?

9.5 *De Bruijn sequence, Eulerian cycle, and card magic* ⋆⋆⋆

A *de Bruijn sequence* of order k is a binary vector of 2^k 0's or 1's, such that a sequence of k 0's or 1's appears only once in the vector (wrapping around the corner). For example, 0011 is a de Brujin sequence of order $k = 2$ because each of the sequences $00, 01, 10, 11$ appears only once as shown in Figure 9.16.

There are two questions regarding de Brujin sequences: for any order k, (1) do they exist? and (2) how to find one?

The answers can be obtained by studying a *de Bruijn graph*. Consider the case of $k = 3$ with the corresponding 2-dimensional de Bruijn graph shown in Figure 9.17. We traverse the graph (starting at any node) while writing down the labels (0 or 1) on the edges. It is not difficult to see that for any **Eulerian cycle** on the graph, *i.e.*, a traversal of the graph using every edge once and only once, the corresponding sequence of edge labels is a de Bruijn sequence. Hence the problem of finding a de Brujin sequence reduces to finding an Eulerian cycle in

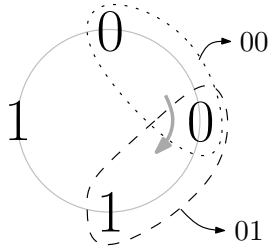


Figure 9.16 Illustration of de Bruijn sequence. Dotted/dashed area is a window which shifts in clockwise direction, covering all possible combinations of 0's and 1's of length 2.

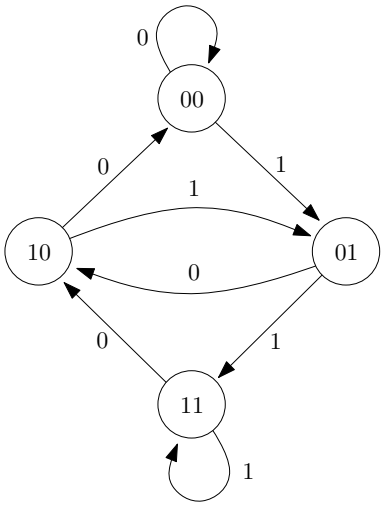


Figure 9.17 A de Bruijn graph.

the corresponding de Bruijn graph, and this answers question (2). For question (1), the answer is affirmative if every de Bruijn graph has an Eulerian cycle, which indeed is true because each node's in-degree and out-degree are equal (a basic result in graph theory due to Euler).

So what are de Bruijn sequences good for? Among their important applications is a famous card trick, where the magician can name the card held by k people in the audience even after random cuts have been made to the deck of cards. (The details can be found in a unique book by Persi Diaconis and Ron Graham: *Magical Mathematics: The Mathematical Ideas that Animate Great Magic Tricks*, Princeton University Press, 2011.)

Now, your task for this homework problem: For $k = 3$ there are two distinct de Brujin sequences. Sequences 01011 and 00111 are distinct, but sequences 01011 and 10101 are not (try to write out the sequences as in Figure 9.16). Draw clearly

two distinct Eulerian cycles on the graph in Figure 9.17, and report the two distinct de Bruijn sequences found.

10 Does the Internet have an Achilles' heel?

10.1 A Short Answer

It does not.

10.2 A Long Answer

10.2.1 Power law distribution and scale-free networks

Sure, the Internet has many security loopholes, from cyber attack vulnerability to privacy intrusion threats. But it does not have a few highly connected routers in the center of the Internet that an attacker can destroy to disconnect the Internet. So why would there be rumors that the Internet has an Achilles' heel?

The story started in the late 1990s with an inference result: the Internet topology exhibits a **power law distribution** of node degrees. Here, the "topology" of the Internet may mean any of the following:

- The graph of webpages connected by hyperlinks (like the one we mentioned in Chapter 3).
- The graph of Autonomous Systems (AS) connected by peering relationship (we will talk more about that in Chapter 13).
- The graph of routers connected by physical links (the focus of this chapter).

For the AS graph and the router graph, the actual distribution of the node degrees (think of the histogram of the degrees of all the nodes) are not clear due to measurement noise. For example, the AS graph data behind power law distribution had more than 50% links missing. Internet exchange points further lead to many peering links among ASs. Theses are shortcuts that enable settlement-free exchange of Internet traffic, and cannot be readily measured using standard network measurement probes.

To talk about the Achilles' heel of the Internet, we have to focus on the graph of routers as nodes, with physical links connecting the nodes. No one knows for sure what that graph looks like, so people use proxies to estimate it through measurements like trace-route. Studies have shown that such estimates lead to biased sampling in the first place, due to the way the Internet protocol reacts to trace-route measurements. In addition, there are other measurement deficiencies

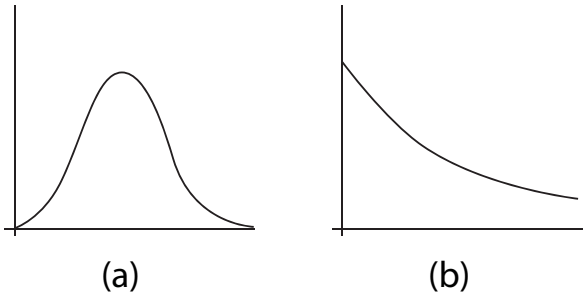


Figure 10.1 (a) Gaussian vs. (b) long tail distribution. Gaussian distribution has a characteristic scale, *e.g.*, standard deviation from the mean, whereas long tail distribution does not.

arising from resolving address ambiguity. There is also no scalable measurement platform with enough vantage points at the network edge to detect the high-degree nodes there. Therefore, it remains unclear if the Internet router graph has a power law degree distribution or not.

But in this chapter we will assume that these graphs exhibit a power law distribution of their node degrees. Even then, the actual graphs and their properties can be tricky to analyze

So, what is a power law distribution? Many distributions, like Gaussian and Exponential distributions, have a characteristic scale, defined by the mean and standard deviation as shown in Figure 10.1(a). And the probability that the random variable following such distribution has a value above a given number x, *i.e.*, the *tail probability*, becomes small very quickly as x becomes large. It is not likely to be far away from the mean. This leads to what is called a **homogeneous** network, defined here as a network where the degrees of the nodes are more or less similar.

In sharp contrast, as shown in Figure 10.1(b), a **long tail distribution** does not have a characteristic scale, leading to the so-called **scale-free network** that is **inhomogeneous** in its node degrees. The tail probability $\text{Prob}[X \geq x]$ of a long tail distribution exhibits the following characteristics:

$$\text{Prob}[X \geq x] \approx kx^{-\alpha},$$

where k is a normalization constant, α is the exponent of the exponential decay, and $\approx$ here means "equal" in the limit as x becomes large, *i.e.*, it *eventually* follows a power law distribution.

The most famous special case of long tail distribution is the Pareto distribu-

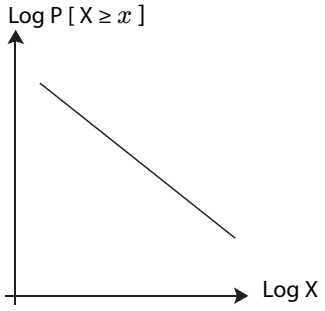


Figure 10.2 Pareto distribution on a log-log plot is a straight line. This is the visual signature of any long tail, or power law, distribution.

tion, with the following tail distribution for $x \geq k$:

$$\text{Prob}[X \geq x] = \left(\frac{x}{k}\right)^{-\alpha}. \tag{10.1}$$

Differentiating the above expression, we see that the Probability Density Function (PDF) for Pareto distribution also follows the power law, with the power exponent $-(\alpha + 1)$:

$$\text{Prob}[X = x] = p(x) = \alpha k^{\alpha} x^{-(\alpha+1)} \tag{10.2}$$

In sharp contrast, for any $x \in (-\infty, \infty)$, the Gaussian probability distribution function (pdf) follows:

$$p(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x-\mu)^2}{2\sigma^2}},$$

where μ is the mean and σ the standard deviation.

Just to get a feel (ignoring the normalization constant) for the difference between exponential of square and 1 over square: following the Gaussian distribution's shape, we have $e^{-x^2} = 0.018$ when $x = 2$. But following the Pareto distribution's shape, we have $x^{-2} = 0.25$ when $x = 2$, a much larger number.

We can plot either the tail distribution (10.1), or the probability density function (10.2), on a log-log scale, and get a straight line for the Pareto distribution. This can be readily seen:

$$\log \text{Prob}[X \geq x] = -\alpha \log x + \alpha \log k.$$

The slope is $-\alpha$, as shown in Figure 10.2. A straight line on log-log plot is the visual signature of power law distribution. It has been reported that the power exponent is -2.1 for in-degree of webpage graph, -2.4 for out-degree of webpage graph, and -2.38 for router graph.

Just to clarify: scale-free network is *not* the same as small-world network. As we saw in the last chapter, a network is called small-world if short paths between any pair of nodes exist and can be locally discovered. A network is called scale-free if its node degree distribution follows a power law, such as the Pareto distribution.

- Scale-free is a topological property concerning just the node degree distribution. It is not a functionality property. A scale-free network does not have to be small world.
- Small world is a topological *and* functionality property that can arise from different node degree distributions. For example, the Watts Strogatz graph is small world but not scale-free, as its node degree distribution does not follow a power law.
- In many social networks, evidence suggests that they are both small world and scale-free. For example, the last model in Chapter 9, the Generalized Watts Strogatz model, can generate such networks.

Back to scale-free networks. Researchers have worked out different generative models, but one of them, the preferential attachment model, gained the most attention. As explained in detail in Advanced Material, preferential attachment generates a type of scale-free network that has highly connected nodes in the center of the network.

In 2000, researchers in statistical physics suggested that, unlike networks with an exponential distribution of node degrees, scale-free networks are robust to random errors, since chances are that the damaged nodes are not highly connected to cause too much damage to the network. But a deliberate attack that specifically removes the most connected nodes will quickly fragment the network into disconnected pieces. The nodes with large degrees sitting in the center of the network become easy attack targets to break the whole network. Since the Internet, even at router level, follows power law degree distribution too, it must be vulnerable to such attacks on its Achilles' heel.

The argument sounds plausible and the implication alarming, except that it does *not* fit reality.

10.2.2 Internet reality

Two networks can have power law degree distribution and yet have very different features otherwise, with very different implications to functional properties, such as robustness to attacks.

For example, what if the high variability of node degrees happens at the network *edge* rather than at the center? That is unlikely if networks were randomly generated according to the model of preferential attachment, where nodes attach to more popular nodes. In the space of all graphs with power law degree distributions, degree variability tends to arise out of the difference between highly connected core nodes and sparsely connected edge nodes.

But what if this unlikely topology is the actual design? Cisco cannot make a router that has *both* a large degree and a large bandwidth per connection. Each router has a limitation on its total bandwidth: the maximum number of packets it can process at each time. So there is an inevitable tradeoff between the number

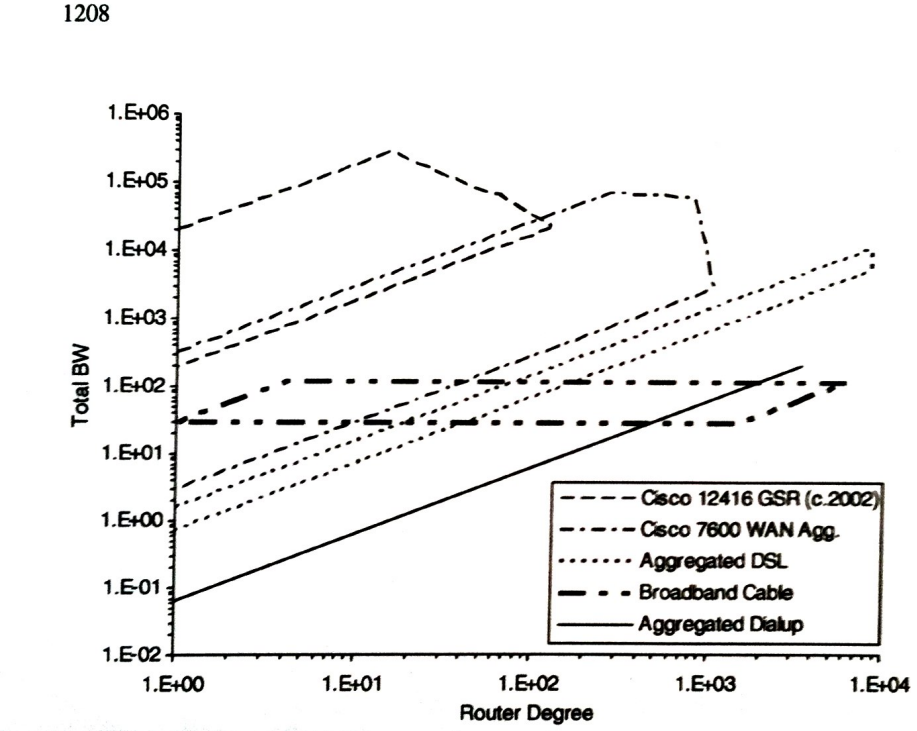


Figure 10.3 Bandwidth vs. connectivity constraint sets for different routers from [Ald+05]. State-of-the-art core routers' capabilities in 2003 are shown. As the degree of the node (*i.e.*, port number per router) goes up, the total bandwidth rises initially, but then drops as more ports force the per-port bandwidth to decrease faster.

of ports (node degree) and speed of each port (bandwidth per connection) on the router. This is illustrated in Figure 10.3.

In addition, the Internet takes layers of aggregation to smooth individual users' demand fluctuations through statistical multiplexing. A user's traffic goes through an access network like WiFi, cellular, DSL, or fiber networks, then through a metropolitan network, and finally, enters the core backbone network. Node bandwidth is related to its placement. The bandwidth per connection in the routers goes up along the way from edge to core.

This implies that nodes in the core of the Internet must have large bandwidth per connection, and thus small degree. For example, an AT&T topology in 2003 showed that the maximum degree of a core router was only 68 while the maximum degree of an edge router was almost five times as large: 313.

In summary, access network node degrees have high variability. Core network node degrees do not. Attacks on high degree nodes can only disrupt access routers and do not disrupt the entire network. Attacks on medium to small degree nodes

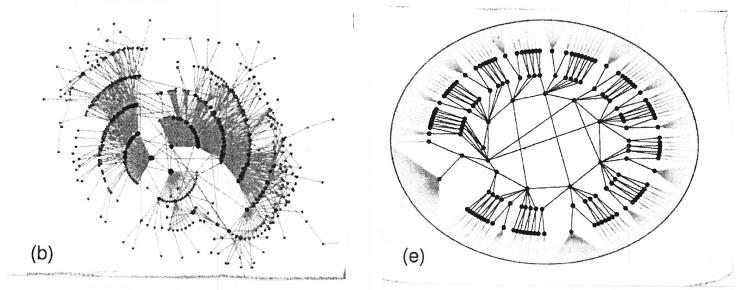


Figure 10.4 Achilles' heel or not, from [Ald+05]. (b) is a typical topology from preferential attachment mechanism. (e) is a typical topology of the real Internet. Both satisfy a power law degree distribution. But (b) has Achilles' heel whereas (e) does not.

have a high chance of hitting the access routers because there are many more of them than core routers.

Moreover, there are also protocols that take care of detecting and mitigating failures even when routers are down, as we will briefly discuss in a homework problem. Robustness of a network is not just about connectivity pattern in the graph.

To summarize, the flaws of "The Internet has an Achilles' heel" are three-fold:

- Incomplete measurements skews the data.
- Power law degree distribution does not imply preferential attachment.
- Functional protection sits on top of topological property.

10.2.3 Functional model

The Internet might be viewed as "self organizing," but that is achieved by design and protocols based on constrained optimization. There is a way to more precisely define the tradeoff between *performance* and *likelihood* of vastly different topologies all exhibiting power law degree distributions.

One of the several ways to capture the aggregate throughput of the Internet, as its performance metric, is through the following optimization problem:

$$\begin{array}{ll} \text{maximize} & \sum_i x_i \\ \text{subject to} & \sum_i R_{ki} x_i \le b_k, \ \ \forall k \\ & x_i = \rho y_{S_i} y_{D_i}, \ \ \forall i \\ \text{variables} & \rho, \{x_i\}. \end{array} \tag{10.3}$$

Here, an end-to-end session i's rate x_i is proportional to the overall traffic demand y_{S_i} at its source node S_i and y_{D_i} at its destination node D_i (*i.e.*, $\mathbf{x}$ is generated by $\mathbf{y}$ in this traffic model), ρ being the proportionality coefficient and the actual optimization variable. Each entry in the routing matrix R_{ki} is 1 if session i passes through router k, and 0 otherwise. The constraint values $\{b_k\}$ capture the router

bandwidth-degree feasibility constraint. The resulting optimal value of the above problem is denoted as $P(G)$: the (throughput) **performance** of the given graph G. Clearly, $P(G)$ is determined by the throughput vector $\mathbf{x}$, which is in part determined by the routing matrix $\mathbf{R}$, which is in turn determined by the graph G.

On the other hand, we can define the **likelihood** $S(G)$ of a graph G with node degrees $\{d_i\}$ as

$$S(G) = \frac{s(G)}{s_{max}}, \tag{10.4}$$

where

$$s(G) = \sum_{(i,j)} d_i d_j$$

captures the pairwise connectivity by summing the degree products $d_i d_j$, over all (i, j) node pairs that have a link between them, similar to the metric of modularity in Chapter 8. And s_{max} is simply the maximum $s(G)$ among all the (simple, connected) graphs that have the same set of nodes and the same node degrees $\{d_i\}$. These $\{d_i\}$, for example, can follow a power law distribution.

Now we have $P(G)$ to represent performance and $S(G)$ to represent the likelihood of a scale-free network. As shown in Figure 10.5, if we were drawing a network from the set of scale-free networks at random, high performance topologies like the one exhibited by the real Internet are much less likely to be drawn. But the Internet was *not* developed by such a random drawing. It came through constrained-optimization-based design. Performance of the topology generated by (10.3) is 2 orders of magnitude higher than that of the random graph generated by preferential attachment, even though it is less than one-third likely to be picked at random.

The poor performance of graphs with highly connected core nodes is actually easy to see: a router with a large degree cannot support high bandwidth links, so if it sits in the core of the network it becomes a performance bottleneck for the whole network. Since "Archilles' heel nodes" would also have been performance bottleneck nodes, they were simply avoided in the engineering of the Internet. We will see this problem again in Chapter 16, when we discuss cloud services and large-scale data centers.

In summary, the Internet router graph is performance driven and technologically constrained. Concluding that there is an Achilles' heel is a story that illustrates the risk of overgeneralizing in a "network science" without domain-specific functional models.

As to the AS graph, the topology can be quite different as it is driven by inter-ISP pricing economics. The webpage graph exhibits yet another topology, increasingly shaped by search engine optimizers in response to Google search methods in Chapter 3, and the extremely popular aggregation websites like Wikipedia and Youtube that we saw in Chapters 6 and 7.

The story of the Internet's (non-existing) Achilles' heel is part of a bigger pic-

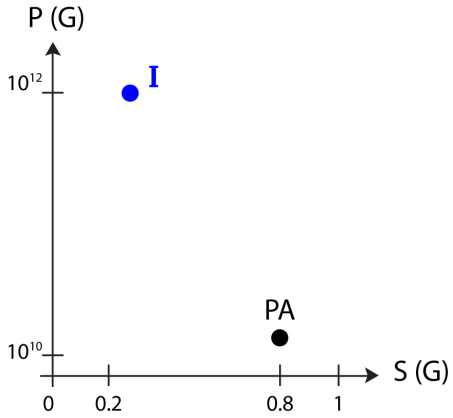


Figure 10.5 Performance vs. likelihood of two topologies, both following a power law degree distribution. Topology I, which stands for the Internet, is much less likely to be picked up if a graph is drawn at random from the set of graphs satisfying power law degree distribution. But it also has much higher performance, as measured by total throughput. Topology PA, which stands for Preferential Attachment, is much more likely but has much lower performance.

ture about generative models of power law distribution: what might have given rise to these ubiquitous power law distributions in graphs we find in many areas? It dates back to over a century ago, when linguist Zipf first documented that the kth popular word in many languages roughly has a frequency of $1/k$ in its appearance. Since then Pareto, Yule, and others documented more of these power laws in the distributions of income and of city size. In the 1950s, there was an interesting debate between Mandelbrot and Simon on the choice of generative models for power law, and on the modeling abilities of fractal/Pareto vs. Poisson/Gaussian. These questions have continued to the present day.

There are several explanatory models of scale-free networks, two of which are well-established and complementary:

- *Preferential Attachment*: the key idea is that as new nodes are added to a graph, they are more likely to connect to those with a lot of connections already. Conceptually, it says that a self-organizing growth mechanism of graphs leads to power law distribution. Mathematically, it turns out that sampling by density leads to a difference equation whose equilibrium satisfies power law, as we will see in Section 10.4.1.
- *Constrained Optimization*: the key idea is that the graph topology is designed with some objective function and constraint in mind, and the resulting topology shows power law distribution. Conceptually, it says that power law is a natural outcome of constrained optimization. Mathematically, it turns out that either entropy or isoelastic utility maximization under linear

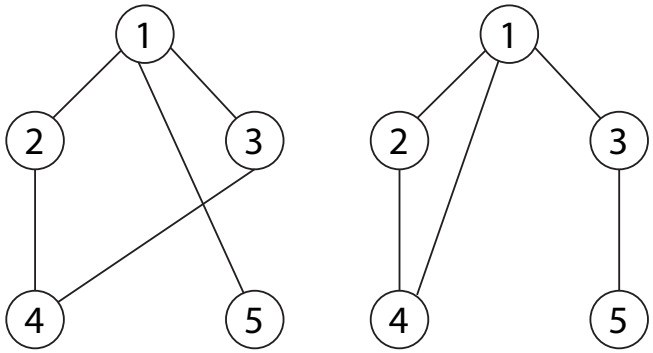


Figure 10.6 Two network topologies G_1 and G_2, both with node degrees {3,2,2,2,1}. G_1 has a smaller S, but much larger P compared to G_2 that maximizes S but suffers from a small P.

constraints gives rise to power law distribution, as we will see in Section 10.4.2. In fact, constrained optimization is not just one single model but a general approach to model functionalities.

Advanced Material will continue this discussion with the debate between preferential attachment and constrained optimization as two options of generative models.

10.3 Examples

Suppose we have a network with graph G_1 shown in Figure 10.6(a). We will calculate $S(G_1)$ and $P(G_1)$.

First of all, we have

$$\begin{aligned} s(G_1) &= d_1d_2 + d_1d_3 + d_1d_5 + d_2d_4 + d_3d_4 \\ &= (3)(2) + (3)(2) + (3)(1) + (2)(2) + (2)(2) \\ &= 23. \end{aligned} \tag{10.5}$$

Among all the (connected) graphs with node degree distribution the same as G_1, the graph with the greatest $S(G)$ is shown in Fig. 10.6(b). Call this graph G_2. A graph that maximizes $S(G)$ generally has more connections between nodes of higher degree. For example, we should swap the 1 in the third term in the sum above with the 2 in the fifth term. We can readily reason that the largest

$s(G)$ must be:

$$\begin{aligned} s_{max} &= s(G_2) \\ &= d_1d_2 + d_1d_3 + d_1d_4 + d_2d_4 + d_3d_5 \\ &= (3)(2) + (3)(2) + (3)(2) + (2)(2) + (2)(1) \\ &= 24. \end{aligned} \tag{10.6}$$

Therefore, $S(G_1) = \frac{s(G_1)}{s_{max}} = \frac{23}{24}$.

Now we turn to calculating $P(G_1)$. Assume we have sessions that wish to connect from each node to each other node, so 10 sessions in total. Let session 1 denote the connection between node 1 and 2, session 2 denote the connection between node 1 and 3, and so forth. Let the node demands be denoted by y_j Gbps, with values $y_1 = 5, y_2 = 2, y_3 = 4, y_4 = 4, y_5 = 2$. Let the bandwidth-degree constraint be $b_k = d_k$ for all k.

To build the routing matrix $\mathbf{R}$, we determine the shortest path between each source-pair destination and write down which routers the path uses. Ties are broken arbitrarily. For example, the shortest path in graph G_1 from node $2 \to 5$ is $2 \to 1 \to 5$. That translates to the 7th column $[1\ 1\ 0\ 0\ 1]^T$ in the routing matrix.

After the above construction, the routing matrix is as follows, with 10 sessions, one per column, and 5 nodes, one per row, *e.g.*, the first column corresponds to session 1, from node 1 to node 2:

$$\mathbf{R} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 0 & 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 \end{bmatrix} \tag{10.7}$$

The demand vector is found by multiplying appropriate entries of y_k. For example, since session 10 routes from node 4 to node 5, we multiply $y_4 \times y_5$ to find the demand constraint $y_{S_{10}} y_{D_{10}}$.

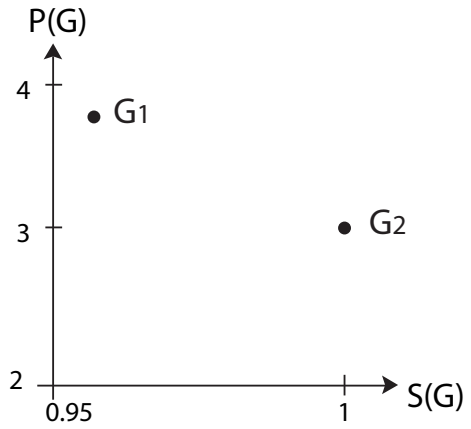


Figure 10.7 P(G) versus S(G) for the simple topology in our numerical example. The less likely graph is the one with much higher total throughput.

The optimization problem is therefore:

$$\begin{aligned}
&\text{maximize} && x_1 + x_2 + x_3 + x_4 + x_5 + x_6 + x_7 + x_8 + x_9 + x_{10} \\
&\text{subject to} && \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 0 & 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \\ x_8 \\ x_9 \\ x_{10} \end{bmatrix} \leq \begin{bmatrix} 3 \\ 2 \\ 2 \\ 2 \\ 1 \end{bmatrix} \\
& && \mathbf{x} = \rho[10\ 20\ 20\ 10\ 8\ 8\ 4\ 16\ 8\ 8]^T \\
&\text{variables} && \rho, \mathbf{x}.
\end{aligned}$$

Solving this numerically, we find that

$$\mathbf{x}^* = \begin{bmatrix} 0.33 & 0.67 & 0.67 & 0.33 & 0.27 & 0.27 & 0.13 & 0.53 & 0.27 & 0.27 \end{bmatrix}^T \text{ Gbps},$$

and $\rho^* = 0.033$. The maximized objective function value gives $P(G_1) = 3.73$ Gbps.

Now, we repeat the procedure to find $S(G_2)$ and $P(G_2)$. By definition, $S(G_2) =$

1. For $P(G_2)$, the routing matrix changes and the optimization problem is:

$$\begin{aligned}
&\text{maximize} && x_1 + x_2 + x_3 + x_4 + x_5 + x_6 + x_7 + x_8 + x_9 + x_{10} \\
&\text{subject to} && \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \\ x_8 \\ x_9 \\ x_{10} \end{bmatrix} \leq \begin{bmatrix} 3 \\ 2 \\ 2 \\ 2 \\ 1 \end{bmatrix} \\
& && \mathbf{x} = \rho[10\ 20\ 20\ 10\ 8\ 8\ 4\ 16\ 8\ 8]^T \\
&\text{variables} && \rho, \mathbf{x}
\end{aligned}$$

Solving this numerically, we find that

$$\mathbf{x}^* = \begin{bmatrix} 0.27 & 0.54 & 0.54 & 0.27 & 0.22 & 0.22 & 0.11 & 0.43 & 0.22 & 0.22 \end{bmatrix}^T \text{ Gbps},$$

and $\rho^* = 0.027$. Now the total throughput is $P(G_2) = 3.02$ Gbps.

We compare graph G_1 and G_2's performance metric to the likelihood metric in Figure 10.7. We are constrained to a small example that allows us to run through a detailed numerical illustration. So the $S(G)$ dynamic range is small. But qualitatively, we see that, compared to G_2, G_1 has a smaller likelihood of getting picked if a graph is randomly drawn from the set of all the graphs with power law degree distributions, but a much higher performance in terms of total throughput.

10.4 Advanced Material

Underlying the Achilles-heel-Internet conclusion is the following chain of logical implications: scale-free networks imply they arise from the mechanism of preferential attachment, and preferential attachment always leads to an Achilles heel. The logical fallacy in this chain of reasoning is that, while preferential attachment generates scale-free networks, scale-free networks do not imply the necessity of preferential attachment. There are other mechanisms that generate scale-free networks just like preferential attachment, but lead to other properties that fit reality better. In the case of Internet router graph, preferential attachment necessarily leads to performance bottlenecks, whereas the Internet must avoid such bottlenecks because core routers with large degree implies that the per-port bandwidth must be small.

10.4.1 Preferential attachment generates power law

Suppose a graph starts with just one node with a link back to itself. At each of the discrete timeslots, a new node shows up. With probability $\theta \in [0, 1]$, it picks an existing node in proportion to the in-degree of each node. With probability $\bar{\theta} = 1 - \theta$, it picks an existing node randomly (similar to the randomization component of pagerank). Then it connects to the node picked. The bigger θ, the more prominent is the preferential attachment effect: a new node is more likely to connect to a popular node. The smaller θ, the more prominent is the random attachment effect.

Now, look at the evolution of $N_i(t)$, the number of nodes with in-degree i when there are t nodes in the network (t is also the timeslot counter since we add one node each time). We will see that attaching to more popular nodes naturally makes them even more popular, which makes them even more likely to be attached to by future new nodes. This leads to the power law distribution.

Mathematically, this can be readily reasoned. N_i increases by 1, if the new node picks an existing node with $i - 1$ in-degree, thus pushing its in-degree to i. The probability of that happening is

$$\theta \frac{(i-1)N_{i-1}}{t} + \bar{\theta}\frac{N_{i-1}}{t} = \frac{\theta(i-1)N_{i-1} + \bar{\theta}N_{i-1}}{t}.$$

The key observation is that $(i - 1)$ multiplies N_{i-1} in the first term. This is the source of the"more popular gets more popular" phenomenon.

On the other hand, N_i decreases by 1, if the new node picks an existing node with i in-degree, thus pushing its in-degree to $i + 1$. The probability of that happening is

$$\frac{\theta i N_i + \bar{\theta} N_i}{t}.$$

The net change in N_i is the difference of the above two expressions:

$$\frac{\theta((i-1)N_{i-1} - iN_i) + \bar{\theta}(N_{i-1} - N_i)}{t}. \tag{10.8}$$

At equilibrium of the growth rates of nodes with different degrees, we can express the number of nodes with i in-degrees as follows:

$$N_i(t) = p_i t,$$

where $p_i \in [0, 1]$ is the proportion of nodes with in-degree i. So the net change of N_i with respect to t is just p_i:

$$\frac{\partial N_i(t)}{\partial t} = p_i.$$

But we just showed that the same quantity can be expressed as in (10.8) too. Setting $p_i =$(10.8) and simplifying the equation, we have:

$$p_i(1 + \bar{\theta} + i\theta) = p_{i-1}(\bar{\theta} + (i-1)\theta).$$

The key observation is that $p_i i$ shows up on the left side, and $p_{i-1}(i-1)$ on the right.

From this balance equation, we can examine the ratio p_i/p_{i-1}:

$$\frac{p_i}{p_{i-1}} = 1 - \frac{1+\theta}{1+\bar{\theta}+i\theta}.$$

Since we care about the tail of the distribution, we focus on the asymptote where i becomes large and dominates the denominator:

$$\frac{p_i}{p_{i-1}} \approx 1 - \left(\frac{1+\theta}{i\theta}\right). \tag{10.9}$$

Let us see what kind of distribution of $\{p_i\}$ satisfies the above equation. Maybe power law distribution? If the distribution of $\{p_i\}$ follows power law:

$$p_i \approx k_1 i^{-(1+\theta)/\theta}, \tag{10.10}$$

where k_1 is a normalization constant independent of i, then the asymptote (10.9) is indeed satisfied, since for large enough i,

$$\frac{p_i}{p_{i-1}} = \left(\frac{i-1}{i}\right)^{\frac{1+\theta}{\theta}} = \left(1-\frac{1}{i}\right)^{\frac{1+\theta}{\theta}} \approx 1 - \left(\frac{1+\theta}{\theta}\right)\left(\frac{1}{i}\right),$$

where the approximation $\approx$ is by Taylor's expansion. We can also verify that (10.10) is also the only way to satisfy this asymptote.

In addition to the probability distribution $\{p_i\}$, the tail of the distribution $q_j = \sum_{i \geq j} p_i$ also follows the power law and is proportional to

$$k_2 j^{-1/\theta}, \tag{10.11}$$

where k_2 is another normalization constant independent of j.

For example, for the power exponent to be -2, θ should be 0.5: follow preferential attachment as much as random attachment.

10.4.2 Constrained optimization generates power law

While preferential attachment is a plausible model that can generate power law distribution, it is not the only one. Another major generative model, especially for engineered networks, is constrained optimization. The power law distribution of node degrees then follows as the consequence of a design that maximizes some objective function subject to technological and economic constraints. The optimization model for Internet topology earlier in this chapter is such an example.

There are more examples, mathematically simple and precise. We will go through one now (maximizing entropy subject to linear cost constraints), and save another for homework (maximizing utility functions subject to linear resource constraints). We can also turn a constrained optimization to a multi-objective optimization, where, instead of forming a constraint, we push it into

the objective function. Either way, it is a model of a tradeoff: the tradeoff between performance and resource cost.

Suppose we want to explain the empirically observed power law distribution of word lengths in a code or a language. For simplicity of exposition, assume each word has a different length. A word with length i has a probability p_i of being used, and the cost of the word is the number of bits to describe it: $\log i$. The average cost is $\sum_i p_i \log i$. The information conveyed by the code is often measured by its entropy: $-\sum_i p_i \log p_i$. We now have the following constrained optimization, where C is a given upper bound on the average cost:

$$\begin{array}{ll} \text{maximize} & -\sum_i p_i \log p_i \\ \text{subject to} & \sum_i p_i \log i \le C \\ \text{variables} & \{p_i\}. \end{array} \tag{10.12}$$

We can normalize $\mathbf{p}$ later .

We can figure out the structure of the optimizer $\{p_i^*\}$ to the above problem by looking at the Lagrange dual problem, an important approach that we will motivate and explain in detail in Chapter 14. But briefly, we weight the constraints with Lagrange multipliers, and hope that

- When these multipliers are chosen properly, maximizing a weighted sum of the objective function and these constraints will be equivalent to solving the original constrained optimization problem.
- Even when these multipliers are not picked to be exactly the "right" ones, useful structures of the optimal solution can still be revealed.

First we form the weighted sum L, with a Lagrange multiplier $\lambda > 0$:

$$L(\mathbf{p}, \lambda) = -\sum_i p_i \log p_i - \lambda \left(\sum_i p_i \log i - C \right).$$

From Lagrange duality theory, it is known that the optimizer $\{p_i^*\}$ to (10.12) must be a maximizer of L. So taking the partial derivative of L with respect to p_j and setting it to 0, we have:

$$\begin{aligned} -\frac{\partial L}{\partial p_j} &= \frac{\partial (p_j \log p_j)}{\partial p_j} + \lambda \frac{\partial (p_j \log j)}{\partial p_j} \\ &= \log p_j + 1 + \lambda \log j \\ &= 0. \end{aligned}$$

Therefore, we have

$$p_j^* = \exp(-\lambda \log j - 1) = \frac{1}{e} j^{-\lambda}, \quad \forall j,$$

a power law distribution with power exponent $-\lambda$, just as we had hoped for.

10.4.3 Broader implications

Which is the "true" explanation of power law: preferential attachment (PA) or constrained optimization (CO)? There is no universally true explanation for all networks. If the network at hand has some elements of design or optimization, CO is more insightful. If not, PA is the more natural explanation. Since they are "just" explanatory models and both options give rise to the same power law, either could work for that purpose, and it does not matter which we choose. The only way to differentiate between the two is to compare attributes beyond just the power law.

Here lies a relatively under-explored topic, as compared to gathering data and plotting the frequency on log-log scale to detect a straight line: what kind of *predictions* can generative models make about topological or functional properties of a given network? In the case of Internet topology, empirical data shows that CO correctly predicts the Internet topology, whereas PA does not.

This is part of an even bigger picture of the debate between **Self-Organized Criticality** (SOC) and **Highly Optimized Tolerance** (HOT) models in the study of networks and complexity. In short, in SOC theory, complex behaviors emerge from the dynamics of a system evolving through "meta-stable" states into a "critical" state. In HOT theory, complex behaviors emerge from constrained design aiming at high efficiency and robustness to designed-for uncertainty (but has high sensitivity to design flaws and unanticipated perturbations). If the system exhibits nongeneric, structured configurations, HOT is a better model. HOT also fits fractal and scaling slopes of power law better. A fundamental watershed is that the source of randomness is topological in SOC but functional in HOT.

Power laws can be found in very different types of networks:

- Biological networks, *e.g.*, brain, species population.
- Human or business networks, *e.g.*, webpage, AS, citation, city growth, income distribution.
- Technology networks, *e.g.*, the Internet at router level.
- A mixture of natural and engineered networks, *e.g.*, forest fire, language.

The third and fourth types of networks tend to have strong constraints and key objectives, and HOT tends to explain them better. Even the first and second types of networks often have constraints, *e.g.*, how many friends one individual can keep is upper bounded by a human being's ability to keep track of friends, thus cutting off the long tail of degree distribution.

The real question is whether the long tail is still there at the regime that matters to the network function under study. However, in some areas in economics and sociology, we cannot easily run controlled, reproducible experiments to falsify a theory, and there are too many self-consistent yet mutually-incompatible theories to differentiate based on historical data.

There are also other ways to generate power law beyond the two in this chapter. This begs the question: if power law is so universally observed and easily

generated, maybe we should no longer be surprised by discovering it, just like we are not surprised when observing that the sum of many independent random variables (with finite means and variances) roughly follow the Gaussian distribution. This is especially true when networks having power law can still have diagonally opposite behaviors in properties that matter, such as resilience to attack (in the case of router topology) and peering economics (in the case of AS topology).

Before leaving this chapter, we highlight two interesting messages. First, this and the previous chapter are mostly about *reverse engineering*, and we saw the use of optimization in explanatory models.

Second, we see the importance of domain-specific knowledge that can correct misleading conclusions drawn from a generic network science, the importance of reality check and falsification against generic topological properties, and the importance of protocol specifics on top of the generic principle of self-organization.

Further Reading

There is a wide literature on power law and long-tail distribution, and on scale-free graphs found in biological, social, and technological networks.

1. The following paper suggested the existence of Achilles' heel in Internet router level topology:

[AJB00] R. Albert, H. Jeong, and A. L. Barabasi, "Error and attack tolerance of complex networks", *Nature*, July 2000.

2. The following paper refuted the above paper in theory and through data. It also developed an optimization model of Internet router level topology that we followed in this chapter:

[Ald+05] D. Alderson, L. Li, W. Willinger, and J. C. Doyle, "Understanding Internet topology: Principles, models, and validation," *IEEE/ACM Transactions on Networking*, vol. 13, no. 6, pp. 1205-1218, 2005.

3. The following paper summarizes many of the key issues in both the collection and the interpretation of Internet measurement data, especially at AS level. The same special issue of this journal also contains other relevant articles.

[Rou11] M. Roughan, W. Willinger, O. Maennel, D. Perouli, and R. Bush, "10 lessons from 10 years of measuring and modeling the Internet's autonomous systems," *IEEE Journal of Selected Areas in Communications*, 2011.

4. The following survey paper traces the history back to Pareto's and Zipf's study of power law distribution, Yule's treatment of preferential attachment, and the Mandelbrot vs. Simon debate on generative models of power law distribution in the 1950s:

[Mit03] M. Mitzenmacher, "A brief history of generative models for power law and lognormal distributions", *Internet Mathematics*, vol. 1, pp. 226-249, 2003.

5. The following best-seller offers many interesting insights to modeling via Gaussian vs. modeling via heavy tail:

[Tal10] N. N. Taleb, *The Black Swan*, 2nd Ed., Random House, 2010.

Problems

10.1 *Probability distributions* ⋆

The **log-normal distribution** has probability density function given by $f(x) = \frac{1}{x\sqrt{2\pi\sigma^2}} e^{-\frac{(\ln x-\mu)^2}{2\sigma^2}}$ and cumulative density function given by $\Phi(\frac{\ln x-\mu}{\sigma})$, where Φ is the cumulative density function of the normal distribution. μ and σ are the mean and standard deviation of the corresponding normal distribution.

Plot the PDF of the following distributions on domain [1,5] with granularity 0.01, all on the same graph for contrast, first using linear and then (natural) log-log scale:

- Pareto distribution with $x_m = 1, \alpha = 1$.
- Normal distribution with $\mu = 1, \sigma = 1$.
- Log-normal distribution with $\mu = 1, \sigma = 1$.

Does the tail of the log-normal distribution look like the normal distribution or the Pareto distribution?

10.2 *Utility maximization under linear constraints* ⋆⋆

We will examine another generative model for power law distribution. Consider the case of nodes communicating to each other over a set of shared links. Each node's utility is a function of the rate it receives. Specifically, we choose the α-fair (also called isoelastic) utility function, where $\alpha \geq 0$ is a fairness parameter. We can formulate this as:

$$\begin{aligned} \underset{\mathbf{x}}{\text{maximize}} \quad & \sum_j \frac{x_j^{1-\alpha}}{1-\alpha} \\ \text{subject to} \quad & \mathbf{Ax} \leq \mathbf{b} \\ & \mathbf{x} \geq 0, \end{aligned}$$

where b_i is the capacity of link i, and x_j is the rate of session j, and $\mathbf{A}$ is the routing matrix between sessions and links:

$$A_{ij} = \begin{cases} 1 & \text{if session } j \text{ is present on link } i \\ 0 & \text{otherwise.} \end{cases}$$

Show that x_j follows a power law (for fixed α), and give an intuition for your

answer.

10.3 *Preferential attachment* ⋆⋆

Recall the preferential attachment model, where a new node attaches with probability θ to another node proportional to its indegree. Specifically, the probability of the new node attaching to node k is $\pi(k) = \frac{d_k}{\sum_j d_j}$ where d_k is the indegree of node k.

Run a simulation of preferential attachment with $\theta = 0.5$ for 500 time steps. Plot p_i versus i. Does it follow the power law distribution as expected?

10.4 *Backup routing topology design* ⋆⋆

Consider the topology in Fig. 10.8. Nodes a, b, c are end hosts and nodes A, B, C, D are routers. Each link is 10 Mbps. There are two sessions going on: node a sends to node c, and node b sends to node c. Each session can split traffic across multiple paths. A link's bandwidth is shared equally between the two sessions if they happen to go through the same link.

(a) For a fixed source and destination, **node-disjoint paths** are paths that do not share any nodes. How many node-disjoint paths are there between a and c? Between b and c? If a and b split their traffic evenly across their disjoint paths, how much bandwidth are a and b able to send to c concurrently?

(b) If router A fails, what happens? Repeat (a).

(c) If routers A and B both fail, what happens? Repeat (a).

10.5 *Wavelength assignment in optical networks* ⋆⋆⋆

In an **optical network** for the Internet backbone, each link has a number of wavelengths. An end-to-end path is called a **lightpath**. A lightpath must be assigned the same wavelength on all of the links along its route. And no two lightpaths can be assigned the same wavelength on any link. For a given graph G and routing, we want to solve this **wavelength assignment** problem.

(a) Show that the wavelength assignment problem on G is equivalent to the graph coloring problem on a related graph $\tilde{G}$. The gaph coloring problem asks for the most efficient (using the smallest number of colors) assignment of one color to each node of a graph so that no adjacent nodes have the same color. What is this $\tilde{G}$?

(b) Let L be the maximum number of lightpaths on any link in a graph. Many optical networks are rings. Show that for any set of lightpaths requests, at most

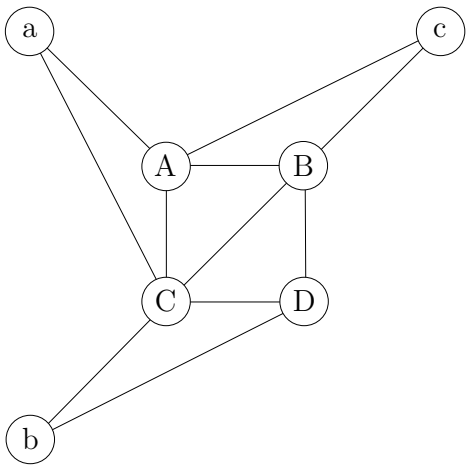


Figure 10.8 A network with 4 routers and 3 end hosts.

$2L - 1$ wavelengths are required, by constructing a greedy algorithm of wavelength assignment on a ring.

11 Why do AT&T and Verizon Wireless charge me $10 a GB?

11.1 A Short Answer

Almost all of our utility bills are based on consumption amount: water, electricity, gas, etc. But when it comes to pricing Internet access, including wireless cellular access where capacity is expensive to provide and difficult to crank up, consumers in some countries like the U.S. have been enjoying flat rate buffets for many years.

In April 2010, AT&T announced its usage-based pricing for 3G data users. This was followed in March 2011 by Verizon Wireless for its iPhone and iPad users, and in June 2011 for all of its 3G data users. In July 2011, AT&T started charging fixed broadband users on U-Verse services based on usage too. In March 2012, AT&T announced that those existing customers on unlimited cellular data plans will see their connection speeds throttled significantly once the usage exceeds 5 GB, effectively ending the unlimited data plan and leaving usage-based plan as the only option. Similar measures have been pursued, or are being considered, in many other countries around the world.

How much is 1 GB of content? If you watch 15 minutes of medium resolution YouTube videos a day, and do nothing else with your Internet access, that is about 1 GB a month. If you stream one standard definition movie, it is about 2 GB. With the proliferation of capacity-hungry apps, high resolution video content, and cloud services (we will discuss video and cloud networking in Chapters 17 and 16), more users will consume more GBs. With the 4G LTE speed much higher than 3G (we will also look into the details of speed calculation in Chapter 19), many of these GBs will be consumed on mobile devices and fall into the $10/GB bracket. Those who are used to flat rate buffet style pricing will naturally find this quite annoying. And if content consumption is suppressed as a result (which does not have to be the case as we will see in the next chapter), usage pricing influences the entire industry ecosystem, including consumers, network providers, content providers, app developers, device manufacturers, and advertisers.

And yet we will see there are several strong reasons, including those in the interests of consumers, that support usage based pricing as a better alternative to flat rate pricing. Whether $10/GB is the right price or not is another matter. We will investigate the pros and cons of usage based pricing from all these angles.

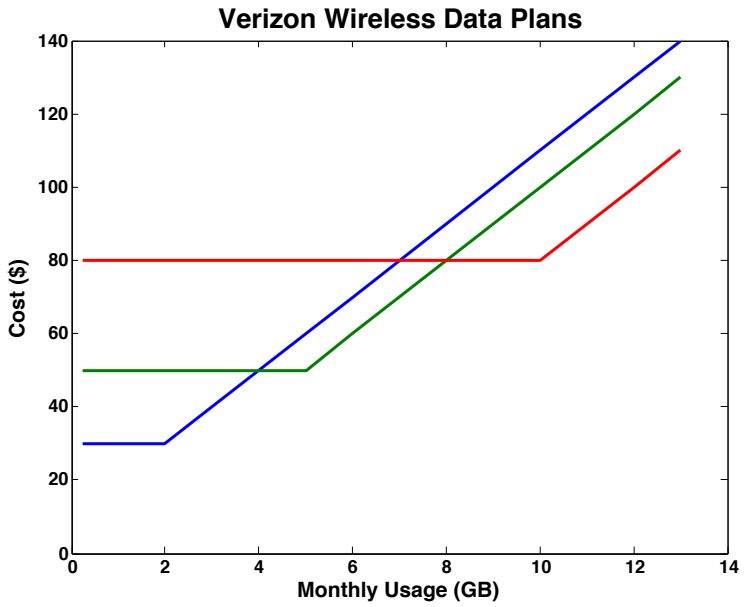


Figure 11.1 Verizon Wireless data plan pricing in 2012. The plans have a flat rate component then a usage based component, *e.g.*, $10 per GB, beyond that.

Despite the different names attached to them, there are two common characteristics of these pricing plans:

- Charge based on total monthly usage. It does *not* matter when you use it, or what you use it for.
- There is a baseline under which the charge is still flat rate. Then a single straight line with one slope, as the usage grows. The actual numbers in Verizon Wireless cellular data plans in 2012 are shown in Figure 11.1.

11.2 Factors behind pricing plan design

Charging based on consumption probably should have sounded intuitive. That is how utilities and commodities are charged. But to those who are used to flat rate Internet connectivity, it represents a radical break. There are two typical precursors to the introduction of usage pricing:

- Network usage surges across many demographics and is projected to climb even higher and faster, *e.g.*, after any mobile carrier introduces iPhones, Android smartphones, and iPads. These devices dramatically enhance the mobile Internet experience and offer many capacity-intensive applications. An

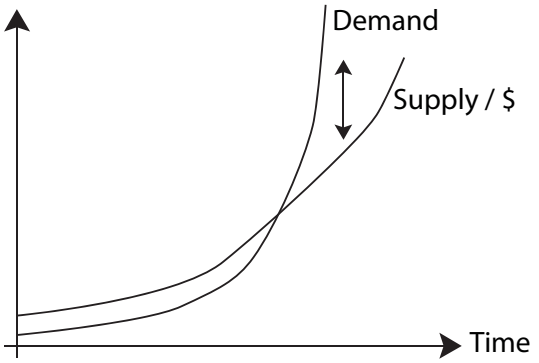


Figure 11.2 The trend of demand and supply/$ of wireless cellular access capacity over time. Demand has caught up with supply per dollar of cost in recent years and, more importantly, is projected to keep growing at ever faster pace.

ISP's profit is the difference between revenue and cost. While the amount of demand is rapidly increasing, per-unit-of-demand revenue also needs to catch up with the cost of supporting the rising demand.

- Government regulation allows pricing practices that match cost. There are other regulatory issues that we will discuss soon, but allowing the monthly bill to be proportional to the amount of usage is among the least controversial ones.

So why did the Internet Service Providers (**ISP**) in countries like the U.S. avoid usage pricing for many years? There were several reasons, including the following two:

- As the mobile Internet market picked up, each carrier had to fight to capture market share. A flat rate scheme is the simplest and easiest one to both increase the overall market acceptance and a particular carrier's market share.
- The growth in the supply of capacity per dollar (of capital and operational expenditure) could still match the growth in demand of capacity.

And why did the carriers change to usage pricing in 2011?

- As illustrated in Figure 11.2, demand growth is outpacing supply/$ growth, and the gap between the two curves is projected to widen even further in the coming years. Once the device suppliers and application communities, such as Apple and iOS app developers, figured out how to make it easy and attractive for users to consume mobile Internet capacity, innovation in those spaces proceeds faster than the supply side can keep up with.

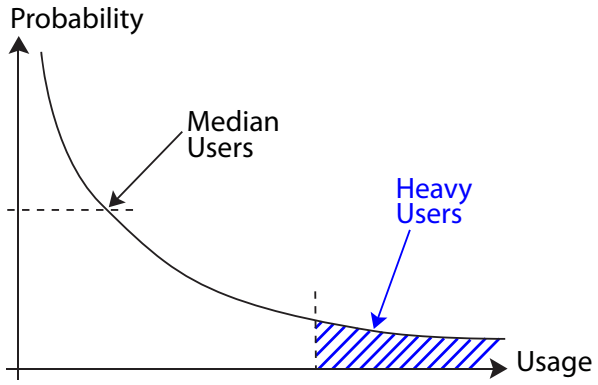


Figure 11.3 Distribution of users' capacity demand, with a heavy tail. The tail users dictate an ISP's cost structure in both capital expenditure and operational expenditure. If the revenue model is based on the median user, the mismatch between cost and revenue will grow as the tail becomes heavier.

Cisco predicts that the mobile Internet demand will keep doubling every year. That is more than 64 times after 5 years. No technology can double supply/$ each year forever. The gradient of human need for capacity has become much larger than the gradient of what Maxwell's equations and Shannon's formula provide.

- If we look at the distribution of capacity demand, the tail of that distribution, shown in Figure 11.3, is often the dominant factor in an ISP's cost structure. That tail has always been long, but is getting longer and longer now. If the ISP still collects revenue based on the median user, the difference between cost and revenue will be too big.

Usage pricing is not the only choices to tackle the above issues. ISPs have other choices, such as:

- Increase the flat rate for everyone as demand increases. With a sufficiently high flat rate, the revenue collected will be adequate. But clearly this creates affordability and fairness issues.
- Cap heavy users' traffic. Once you exceed a cap, you can no longer use the network, or throttle the speed to the point that the quality of service becomes too low for practical use. This is actually a special case of usage pricing: the pricing slope becomes infinite beyond the baseline.
- Slow down certain classes of traffic. For example, for a period of time, Comcast throttled BitTorrent users, who often had massive amounts of file sharing and movie downloads using the popular P2P service. This may raise net neutrality concerns.

- Smarter versions of usage pricing in the next chapter.

Most of the ISPs have realized the problem and started pursuing the usage pricing solution. What are the criteria that we can use to compare alternative solutions to the exploding demand of mobile Internet? There are too many to list here, but the top ones include the following:

- *Economic viability*: as profit-seeking companies, ISPs need to first recover cost and then maximize profit. Their profit margins are in general declining, as many other transportation businesses have seen in the past. Mobile and wireless networks are bright spots that they need to seize.
- *Fairness*: consumer A should not have to use her money to subsidize the lifestyle of consumer B.
- *Consumer choice*: consumers need to be able to choose among alternatives, *e.g.*, spend more money to get premium services, or receive standard services with a cost saving.

Along all of the above lines, usage pricing makes more sense than fixed pricing, although it can be further enhanced with more intelligence as we will describe in the next chapter. The key advantages of usage pricing are listed below and will be analyzed in detail in the next section.

- Usage pricing produces less "waste" and matches cost.
- Usage pricing does not force light users to subsidize heavy users.
- Usage pricing helps with better differentiation in the quality of using the Internet.

11.3 Network neutrality debates

Before we move to the next section, it is worthwhile to mention "network neutrality," a central policy debate especially in the U.S. Counter-productive to useful dialogues, this "hot" phrase has very different meanings to different people. Usually there are three layers of meanings:

- *Access/choice*: consumers should have access to all the services offered over the Internet, and a choice of how they consume capacity on the Internet.
- *Competition/no monopoly*: ISPs should have no monopoly power and the market place needs to have sufficient competition.
- *Equality/no discrimination*: All traffic and all users should be treated the same. This may actually contradict the requirement of access/choice.

While the last point might sound like an ideal target, it is sometimes neither feasible nor helpful to carry out. There are four types of "no discrimination," based on what "discrimination" means:

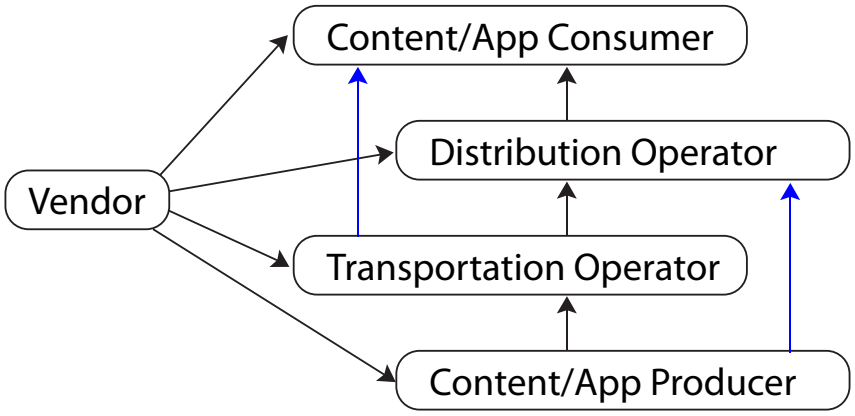


Figure 11.4 5-party interactions in the industry. Content/app producers include studios and YouTube, transportation operators include AT&T and Comcast, distribution operators include Akamai and Bit Torrent. "Shortcuts" have been created in the traditional food chain, from content/app producers directly to distribution operators, and from transportation operators directly to consumers, further complicating the industry interaction.

- *Service limitation*: *e.g.*, because of vertical integration, an ISP also becomes a content owner, possibly blocking access to other content. Or an ISP could block access to voice call apps on iPhone in order to generate more revenue for its own voice business.
- *Protocol-based discrimination*: *e.g.*, certain protocols generate a significant amount of heavy traffic, *e.g.*, Bit Torrent, and get blocked.
- *Differentiation of consumer behaviors*, *e.g.*, usage pricing is one of the simplest ways to correlate pricing with consumer behavior: if consumer A takes up more capacity, she pays more.
- *Traffic management and Quality-of-Service provisioning*, *e.g.*, have more than one queue in a router, schedule traffic with weighted fair queuing, allow emergency traffic like healthcare monitor signals to take higher priority than non-essential software updates. (We will discuss quality of service mechanisms in Chapter 17.)

While neutrality against service limitation is essential to have, neutrality against protocol-discrimination is debatable, neutrality against consumer behavior differentiation is harmful, and neutrality against traffic management is downright impossible: if having more than one queue is anti-neutrality, then the Internet has never been and will never be neutral.

In fact, a naive view of "equality" harms the tenet of providing access and choice to consumers, a more important component of neutrality. As summarized by the Canadian Radio, Television and Communications office: "economic practices are the most transparent Internet traffic management practices," and

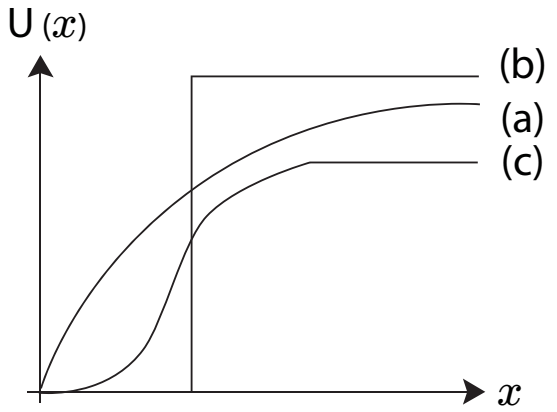


Figure 11.5 Three types of utility function shapes: (a) concave, (b) discontinuous, and (c) sigmoidal. Eventually utility functions all become concave as marginal returns diminish. Maximizing concave and smooth utility functions is easier than maximizing sigmoidal or discontinuous utility functions.

"match consumer usage with willingness to pay, thus putting users in control and allowing market forces to work."

There is much more to network neutrality than we have space for in this chapter. This is further complicated by the fairness and efficiency issues arising out of the 5-party interactions shown in Figure 11.4.

11.4 A Long Answer

11.4.1 Utility maximization model

In order to proceed further to understand Internet access pricing, we need to build some model of consumer demand. **Utility function** is a common modeling tool in economics to capture "how happy" a user would be if a certain amount of resource is allocated. In Chapters 1 and 3, we saw payoff functions in games. Utility functions are a generalization of these. They further lead to models of strategic thinking by users, based on assumptions in the expected utility theory and its many extensions.

A typical utility function is shown as curve (a), a log function, in Figure 11.5. We denote the utility function of session i as $U_i(x_i)$, where x_i is some performance metric like throughput. Maximizing the sum of utilities across all users, $\sum_i U_i(x_i)$, is referred to as **social welfare maximization**. It is that theme of scalarization of vectors again, as we saw in social choice theory in Chapter 6 and will see again in fairness definition in Chapter 20.

Where do these utility function models come from? One source is human

subject experiment. For example, researchers run tests with a focus group, trying different rates, delays, and jitters of voice calls, and ask them to quantify how happy they are in each case. This leads to utility models for various multimedia applications.

Another is **demand elasticity** modeling, which relies on observed consumer behavior. Given a utility function $U(x)$, we also have an associated **demand function**, capturing the volume of demand as a function of price offered. Think about the following **net utility maximization**: a user picks the x that maximizes the difference between utility and total price paid:

$$U(x) - px,$$

where p is the unit-price. If U is a concave function, it is easy to solve this optimization over one variable: just take the derivative with respect to x and let it be 0: $U'(x) = p$. Since U' is invertible, we can write x, the resulting demand, as a function of p. We call $U^{'-1}$ the demand function D, and it is always a decreasing function, with a higher price inducing a lower demand:

$$x = U^{'-1}(p) = D(p).$$

So, a utility function determines the corresponding demand function. It also determines demand elasticity, defined as (normalized) price sensitivity of demand:

$$\eta = -\frac{\partial D(p)/\partial p}{D(p)/p}.$$

For example, if utility is logarithmic, then the demand function is $x = 1/p$ and the elasticity is 1.

The third ground on which we pick utility models is **fairness**. We will later devote a whole chapter on fairness of resource allocation. At this point, we will just bring up one approach. There is a class of utility functions called **α-fair utility functions**, parameterized by a positive number $\alpha \geq 0$, and if you maximize them you will get optimizers that satisfy the definition of **α-fairness**. It is also called **isoelastic** utility functions.

Now the details. We call a feasible resource vector $\mathbf{x}$ α-fair, if any other feasible resource vector $\mathbf{y}$ satisfies the following condition:

$$\sum_i \frac{x_i - y_i}{x_i^\alpha} \leq 0. \tag{11.1}$$

That means, roughly speaking, that the (normalized) deviation from $\mathbf{x}$ does not pay off.

It turns out that if you maximize the following function parameterized by $\alpha \in [0, \infty)$:

$$U_\alpha(x) = \begin{cases} x^{1-\alpha}/1 - \alpha & \alpha \neq 1 \\ \log x & \alpha = 1, \end{cases} \tag{11.2}$$

the optimizer is α-fair. This result says that you can choose utility models by

looking at which fairness notion you would like to impose. Three points are particularly useful:

- $\alpha = 0$ is simply sum resource maximization and often unfair as some user i may receive 0 resource.
- $\alpha = 1$ is called **proportional fairness**: just look at (11.1) with $\alpha = 1$.
- $\alpha \to \infty$ is called **max-min fairness**: you cannot increase some x_i without reducing some other x_j that is already smaller than x_i. This is a "stronger" requirement than Pareto efficiency we saw in Chapter 1.

We have not really justified *why* bigger α is more fair, and this fairness metric is only achieved by the optimizer at the *equilibrium*. We will come back to these points in later chapters. And in a homework question, you will derive the elasticity of α-fair utility functions. The smaller α, the more elastic the demand.

However constructed, this utility function could be a function of all kinds of metrics of interest. Here we are primarily interested in either utility as a function of data rate (in bits per second) or of data volume (in bytes). But in general it could also be a function that depends on delay, jitter, distortion, or energy.

Utility functions are increasing functions ($U' \geq 0$). They are often assumed to be smooth (*e.g.*, continuous and differentiable) and concave ($U'' \leq 0$), even though that does not have to be the case. In some cases, utility is 0 when x is below a threshold and a constant otherwise, leading to a discontinuous utility function, like curve (b) in Figure 11.5. In other cases, it starts out as a convex function: not only is the user happier as x becomes larger, but the rate at which her happiness rises also increases. But after an inflection point, it becomes concave: more x is still better, but the incremental happiness for each unit of additional x drops as x becomes larger. Such functions are called sigmoidal functions, as shown in curve (c) in Figure 11.5. Due to the principle of **diminishing marginal return**, utility functions eventually become concave, possibly flat, for sufficiently large x.

11.4.2 Tragedy of the commons

The "network effect" is often summarized as follows: the benefit of having one more node in a network goes up as the square of the size of the network. The underlying assumptions are that the benefits increase proportionally as the number of links in the network, and that everyone is connected to pretty much everyone else.

There is also a famous downside of having a group of people: "tragedy of the commons." This concept was sketched by Lloyd in 1833 and made popular by Hardin's article in 1968. Consider a group of N farmers sharing a common parcel of land to feed their cows. If there are too many cows, the land will be overgrazed and eventually all cows will die. Should a farmer get a new cow? The benefit of having a new cow goes entirely to him, whereas the cost of overgrazing is shared by all N farmers, say, $1/N$ of the cost. So each farmer has the incentive to

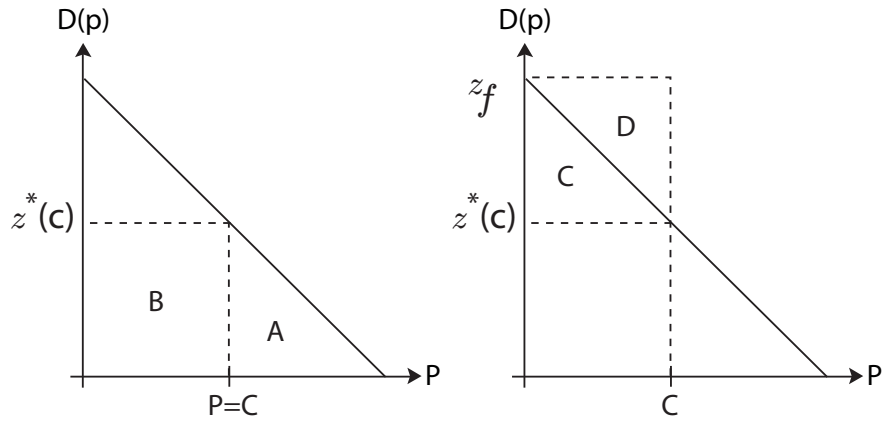


Figure 11.6 Under usage pricing, the surplus (utility bigger than cost) is area A. Flat rate creates waste (cost bigger than utility) and reduces consumer surplus by area D.

acquire more cows, even though this collectively leads to overgrazing, the worst case scenario for everyone. This is one more example of *negative externality*, since it is not represented by pricing signals. It is another of those mutually-destructive phenomena like the Nash equilibrium in the prisoner's dilemma in Chapter 1.

One solution is to charge each farmer a cost proportional to N, to compensate for the inadequate incentive. This amounts to changing the net utility calculation of each farmer from

$$\text{maximize } U(x) - x$$

to

$$\text{maximize } U(x) - Nx.$$

This process of assigning the right price is called **internalizing** the externality. If the utility function is logarithmic, the demand drops from $x^* = 1$ to $x^* = 1/N$ now.

We will see in Chapter 14 how TCP essentially uses congestion pricing to internalize the externality of congestion in the Internet shared by many users, and in Chapter 15 how P2P protocols use tit-for-tat to internalize the externality of free riders in file sharing networks.

11.4.3 Comparison between flat rate and usage price

There have been many studies comparing usage based pricing with flat rate pricing, from those in the late 1980s for the broadband network ISDN envisioned then to empirical studies like the UC Berkeley INDEX experiment in 1998. The main conclusions can be summarized in three graphs.

These graphs chart the demand function $D(p) = U'^{-1}(p)$, as a function of

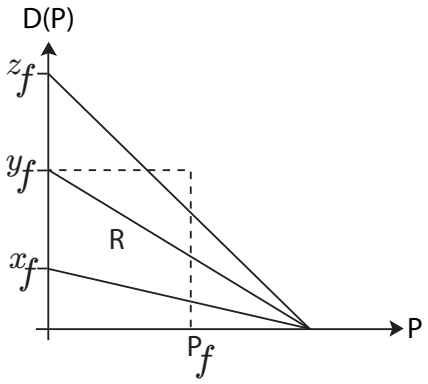


Figure 11.7 Flat rate pricing penalizes light users in favor of heavy users. The average user's demand curve intercepts y-axis at y_f, whereas the light user's intercepts at x_f and heavy user's at z_f. Light users receive negative surplus and subsidize the heavy user's utility.

price p. (In economics literature, we usually plot p vs. $D(p)$, which is the inverse demand function.) For simplicity, let us say the demand functions are linear.

In Figure 11.6, we see that if the charge is usage based and the price is p_u, the incremental cost for the ISP to provide this much capacity, then the corresponding demand is $x_u = D(p_u)$.

- Since $D^{-1} = U'$, the utility to the user is the area (integration) under the inverse demand curve, *i.e.*, area A+B.
- The cost to the user is the rectangle $p_u x_u$, *i.e.*, area B.
- So the **user surplus**, defined as the difference between utility and cost, is area A.

In contrast, under flat rate pricing instead of usage-based pricing, the user will consume all the way to x_f. The extra utility achieved compared to the case of usage pricing is area C, whereas the extra cost is area C+D, which creates a waste, *i.e.*, negative surplus, of area D.

Now we consider three types of users: an average user's demand curve is in the middle of Figure 11.7. A light user's demand curve is the lower one, and a heavy user's the higher one. In order for the ISP to recover cost, it has to set the flat rate charge so that the revenue based on the average user is large enough, *i.e.*, set p_f so that area R is large enough to recover capacity costs. Since the utility to a user is the triangle's area, clearly the surplus for light users can be negative, whereas that for heavy users is positive. Light users subsidize heavy users under flat rate.

This becomes even more of a problem if the ISP wants to offer an upgrade,

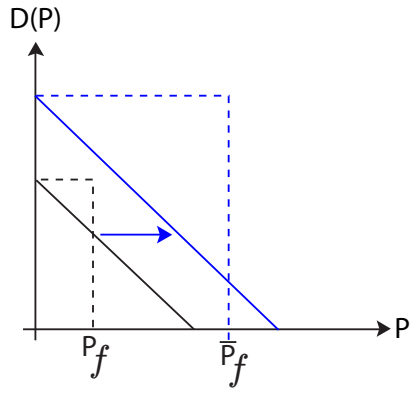


Figure 11.8 Flat rate pricing discourages higher quality service adoption by light users. A service upgrade is represented by pushing the demand curve to the right as a parallel line. Surplus can be bigger for lower quality service than that for higher quality service.

say, in the Internet access speed. This shifts the demand curve to the right, as there will be higher demand if the price remains the same while the service is enhanced. It shifts the cost-recovering flat rate price too, from p_f to $\bar{p}_f$. It is possible that, for a light user shown in Figure 11.8, the difference between utility (the triangle area) and the cost (the rectangle area) is bigger under the lower speed service than under the higher speed service. So the light user sticks to the worse service. This is due to the fact that recovering cost through a flat rate is inflexible.

11.5 Examples

We consider a simple numerical example with just one user, before moving into the more general case with many users sharing a link with fixed capacity in Advanced Material. The user has a weighted log utility function, with a positive multiplicative weight σ:

$$U(x) = \sigma \log x.$$

The usage-based price is the combination of baseline flat rate and usage fee:

$$p(x) = g + hx.$$

Strictly speaking, it should be $g+h(x-x_0)$ where x_0 is the baseline usage covered by the flat rate, but we will ignore this to simplify the presentation without losing the essence of the result.

The questions we face are: given (g, h) from the ISP, what would be user's

demand $x^*(g,h)$? And should the ISP charge a lower flat rate g dollar, but a steeper slope h dollars per GB? Or should it charge the other way around?

The first question is readily answered from the demand function for weighted log utility function. Differentiating

$$\sigma \log x - (g + hx)$$

with respect to x, and setting that to zero, we have

$$x^* = \frac{\sigma}{h}.$$

The second question depends on the market power of the ISP. If it is a monopoly ISP with price-setting power, it can push the user's net utility (the difference between utility and cost) to 0:

$$\sigma \log x = g + hx.$$

Using the expression for x^*, we have

$$\sigma \log \left(\frac{\sigma}{h}\right) = g + h\frac{\sigma}{h}.$$

This means we can express the flat rate fee as a function of the usage based fee:

$$g = \sigma \left(\log \left(\frac{\sigma}{h}\right) - 1\right).$$

In Advanced Material, we will see how a common h across all users can be set to avoid the sum of user demand exceeding a fixed capacity. For now, we can explore some typical h values. Suppose the user's utility level is $\sigma = 100$.

- If $h = \$2/GB$, g equals \$70, and the total revenue to the ISP is $g+hx^* = \$170$, with the flat rate component $g/(g + hx^*) = 41\%$ of the revenue.
- If $h = \$5/GB$, g equals \$30, and the total revenue to the ISP is $g+hx^* = \$130$, with the flat rate component $g/(g + hx^*) = 23\%$ of the revenue.

For this user's demand elasticity, it is better for the ISP to charge a higher flat rate fee g and a shallower slope of usage fee h.

In this example, you probably have spotted the trend: a smaller h means that g can afford to be higher and x will go up, to the point that the total revenue sees a net increase. So we might as well make h arbitrarily close to 0. This is an artifact of three simplifying assumptions in the model:

- The ISP, as a monopoly, has complete price-setting power.
- There is only one bottleneck link.
- There is no capacity constraint or cost, so it is always advantageous to the ISP to increase demand.

In the next section, we will still keep first two assumptions but eliminate the third one, which leads to a more insightful analysis. Then, in congestion control in Chapter 14, we will further remove the other two assumptions in a different timescale of machine protocol reaction.

11.6 Advanced Material

11.6.1 Structure of usage price

The congestion constraint faced by an ISP is on the peak aggregate consumer data-rate. In contrast, the access price is based on the volume of data consumed over a specified time period t. Pricing data volume is equivalent to pricing the *average* data rate over time t. Therefore an ISP faces a mismatch, because its revenue is accrued on the average data rate but congestion cost is incurred on the peak data rate. This mismatch will be addressed in the next chapter.

For now, we note that the difference between peak data rate and the average data rate is reduced when measured over smaller time periods. Consider a unit time interval t that is sufficiently small so that the peak data rate demand of a consumer in that time interval is a close approximation to the average data rate in that interval.

Let $f \in F$ be a consumer data flow, and the data rate for flow f in the interval $[(t-1), t]$ be given by x_f^t. The data volume consumption over time T is then given by $x_f = \sum_{t=1}^{T} x_f^t$, and the capacity constraint C applies at every time instant t on a single bottleneck link (often the access link) across all the flows:

$$\sum_f x_f^t \leq C, \ \ \forall t.$$

The shape of the utility function depends on the application's performance sensitivity to varying data rates, and the utility level represents the consumer's need for the application or content. This motivates us to assume that the consumer's utility level varies in time, but the shape of the utility function does not. Let $\sigma_f^t U_f(x_f^t)$ be the utility to a consumer associated with flow f at time instant t, with factor σ_f^t denoting the time dependency of consumer's utility level, leaving the utility shape independent of time t.

Faced with time-varying consumer utilities, the ISP can charge a time-dependent, flow-dependent price $r_f^t(x_f^t)$, as a function of the allocated data rate x_f^t. Consumers maximize the net utility for each flow f:

$$\begin{array}{ll} \text{maximize} & \sigma_f^t U_f(x_f^t) - r_f^t(x_f^t) \\ \text{variable} & x_f^t \end{array} \tag{11.3}$$

The most common form of the price r is a flat rate baseline g, followed by a linear increase as a function of data-rate with slope h, like what we saw in Figure 11.1:

$$r_f^t(x_f^t) = g_f^t + h_f^t x_f^t. \tag{11.4}$$

The *flat price* g_f^t is fixed for the duration of the time interval, irrespective of the allocated data rate. The *usage based* component is based on a price h_f^t per unit data consumption.

The demand function for this form of the price is

$$D_f^t(g_f^t, h_f^t) = \begin{cases} {U'_f}^{-1}(h_f^t/\sigma_f^t) & \text{if } g_f^t + h_f^t y_f^t \le \sigma_f^t U_f(y_f^t) \\ 0 & \text{otherwise.} \end{cases} \tag{11.5}$$

The condition $\sigma_f^t U_f(x_f^t) - g_f^t - h_f^t x_f^t \ge 0$ ensures that consumers have non-negative utility, by making g sufficiently small. To simplify notation, we often use $D_f^t(h_f^t) = {U'_f}^{-1}(h_f^t/\sigma_f^t)$, with the implicit assumption that the flat price is low enough to ensure non-negative consumer net-utility.

The revenue maximization problem for a **monopoly** ISP can be defined by the following: maximize total revenue subject to the capacity constraint and the consumer demand model:

$$\begin{array}{ll} \text{maximize} & \sum_t \sum_f (g_f^t + h_f^t x_f^t) \\ \text{subject to} & \sum_f x_f^t \le C, \ \ \forall t \\ & x_f^t = {U'_f}^{-1}(h_f^t/\sigma_f^t), \ \ \forall t, f \\ & \sigma_f^t U_f(x_f^t) - g_f^t - h_f^t x_f^t \ge 0, \ \ \forall t, f \\ \text{variables} & \{g_f^t, h_f^t, x_f^t\} \end{array} \tag{11.6}$$

The variables $\{g_f^t, h_f^t\}$ are controlled by the ISP, and $\{x_f^t\}$ are the reactions from the users to the ISP prices.

We still have not incorporated any routing matrix that couples the distributed demands in more interesting ways. In Chapter 14 we will bring routing (over a general network topology) into the formulation.

Since we assumed a monopoly ISP market, the ISP has *complete pricing power*, which is not the case in reality. We will see an example of competitive ISP market, the other end of abstraction of ISP market power, in the next chapter.

Obviously, ISP revenue increases with a higher flat fee component g_f^t, which can be set so that the consumer net utility is zero. The revenue from each flow is then $g_f^t + h_f^t x_f^t = \sigma_f^t U_f(x_f^t)$, which can be realized by any combination of flat and usage fee that can support a data-rate of x_f^t.

If the usage fee h_f^t is such that the consumer demand $D_f^t(h_f^t)$ is greater than the ISP provisioned data rate x_f^t, then flow's packets have to be dropped. However, the ISP can avoid packet drops by setting a sufficiently high usage price to reduce the consumer demand so that the aggregate demand is within the available capacity. It follows that $x_f^t = D_f^t(h_f^t)$.

Therefore, the ISP revenue maximization problem (11.6) simplifies to:

$$\begin{array}{ll} \text{maximize} & \sum_t \sum_f \sigma_f^t u_f(D_f^t(h_f^t)) \\ \text{subject to} & \sum_f D_f^t(h_f^t/\sigma_f^t) \le C, \ \ \forall t \\ \text{variables} & \{h_f^t\}. \end{array} \tag{11.7}$$

The capacity inequality should be achieved with equality at optimal prices. It suffices to have the optimal usage fee h^t be the *same* across all flows f, since it will be used to avoid capacity waste in the sum of demands across all the flows (an argument that will be rigorously developed in a homework problem).

The optimal flat fee g_f^t is flow dependent, allowing the ISP to fully extract the consumer net-utility.

Therefore, an optimal pricing scheme that achieves the maximum in (11.7) is given by the following: for each t, the per-unit usage price h^t is set such that the resulting demands fully utilize the link capacity C on the only bottleneck link in the network:

$$\sum_f x_f^t = \sum_f D_f^t(h^t) = C,$$

and the flat rate baseline prices $\{g_f^t\}$ are set such that the maximum revenue is generated out of the consumer demand:

$$g_f^t = \sigma_f^t U_f(x_f^t) - h^t x_f^t.$$

Let R_F^* be the revenue from the flat component of the optimal price, and R_S^* the revenue from the usage component. In a homework problem, by using the above solution structure, we can derive the ratio between the flat and usage components. In an exemplary special case, if utility functions are α-fair with $\alpha_f = \alpha$ for all f, the ratio of flat revenue to usage dependent revenue becomes:

$$\frac{R_F^*}{R_S^*} = \frac{\alpha}{1-\alpha}. \tag{11.8}$$

This reveals that usage dependent revenue dominates with linear utilities ($\alpha \to 0$), while revenue from flat rate components dominates with log utilities ($\alpha \to 1$). The flat price is a significant component in the extraction of consumer net-utility if the consumer price sensitivity is low.

Consider a monopolist ISP providing connectivity service to 10 flows over an access link of capacity $C = 10$Mbps. We generate the utility levels $\{\sigma_f^t\}$ randomly the range $[\sigma_0, \sigma_1]$.

The upper graph in Figure 11.9 illustrates the average revenue (per unit time and per unit flow) received by the monopolist ISP:

- The flat component of the revenue, which enables the monopolist ISP to completely extract the consumer net-utility, increases with decreasing elasticity of demand.
- The usage component of the revenue decreases with decreasing elasticity of demand.

The lower graph in Figure 11.9 plots the ratio of the flat component to the usage component of the revenue, demonstrating the increased reliance on revenue from flat price at low demand elasticity.

The ISP pricing flexibility, in practice, is restricted along *time* and across *flows*.

- Fixing $\{g, h\}$ to be the same for all times t leads to a tradeoff between oversubscribing link capacity and dropping packets on the one hand, and underutilizing link capacity and losing revenue on the other hand.

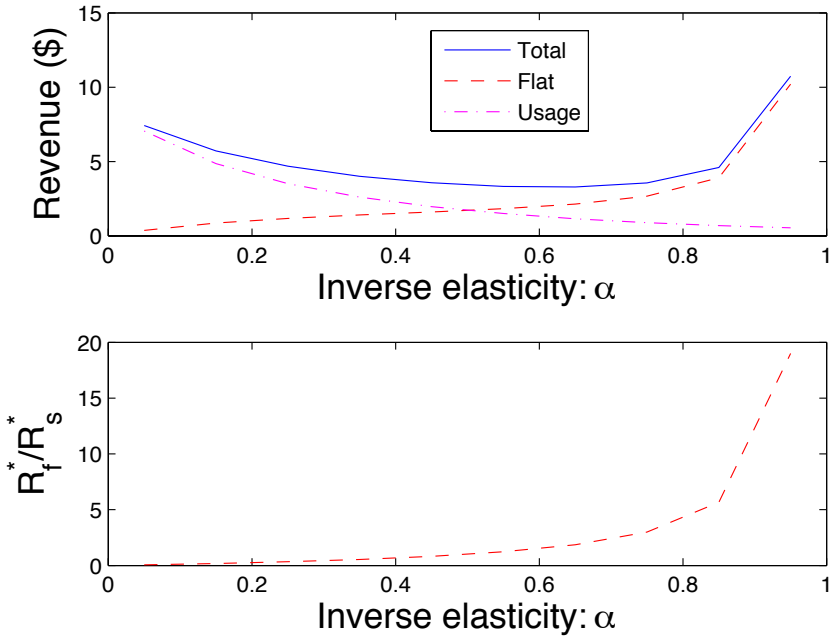


Figure 11.9 Comparison of revenue from flat pricing to usage based pricing at different consumer demand elasticity. All users have α-fair utility functions with the same α. As α increases and elasticity drops, the flat rate component's fraction of the overall revenue to the monopoly ISP increases.

- Fixing $\{g, h\}$ to be the same for all the flows leads to a loss of revenue, which can be mitigated by allowing nonlinear pricing: discounted per-GB pricing at high volume to encourage higher utilization. This is called the **second order price discrimination**.

So far we have focused only on the question of "How much to charge". In the next chapter and its homework problems, we will continue to discuss three other questions of consumer pricing on the Internet: "How to charge," "What to charge," and "Whom to charge."

Further Reading

There is a growing research literature on network economics, including Internet access pricing.

1. An excellent edited volume back in 1998 has many insightful chapters still relevant today, especially those written by Clark, by Kelly, and by Varian:

[MB98] L. W. McKnight and J. P. Bailey, Ed., *Internet Economics*, The MIT Press, 1998.

2. A more recent survey article of the subject can be found at:
[Wal09] J. Walrand, "Economic models of communication networks," 2009.

3. Our treatment of the basic intuitions between flat rate and usage based pricing follows those reported in the summary of the INDEX experiment in the late 1990s:
[EV99] R. Edell and P. Varaiya, "Providing Internet access: What we learn from INDEX", *IEEE INFOCOM Keynote*, 1999.

4. Our treatment of utility maximization model of flat rate vs. usage based pricing follows this recent paper:
[Han09] P. Hande, M. Chiang, A. R. Calderbank, and J. Zhang, "Pricing under constraints in access networks: Revenue maximization and congestion management," *Proc. IEEE INFOCOM*, 2009.

5. The following classical paper describes the tragedy of the commons:
[Har68] Hardin, "The tragedy of the commons," *Science*, vol. 162, pp. 1243-1248, 1968.

Problems

11.1 *α-fair utility function* $\star$

Plot α-fair utility functions with $\alpha = 0, \frac{1}{5}, \frac{1}{2}, 1, 2, 5, 100$ in one graph, and compare them.

11.2 *Demand function and demand elasticity* $\star$

(a) Derive the demand and the demand elasticity as functions of price, if utility function is $u(x) = \arctan(x)$

(b) Repeat (a) for α-fairness utility functions.

11.3 *Demand v.s. supply curves* $\star\star$

In microeconomics the demand $D(p)$ and **supply** $S(p)$ as a function of price p can be defined as follows:

$$D(p) = {u'}^{-1}(p)$$
$$S(p) = {c'}^{-1}(p)$$

where $u(x_b)$ is the utility of buyer as a function of the amount purchased (x_b), $c(x_s)$ is the cost of producer as a function of the amount produced (x_s). Fig.11.10 gives an illustration with capital latters indicating the corresponding areas.

(a) Assume $u(0) = 0$, prove that $u(x_0) = A + B + C$, therefore the buyer's net utility is $u(x_0) - p_0 x_0 = A$

(b) Assume $c(0) = 0$, prove that $c(x_0) = C$, therefore the seller's profit is $p_0 x_0 - c(x_0) = B$

(c) At which price p^* does the social welfare $u(x^*) - c(x^*)$ take maximum value, where $x^* = \min\{D(p^*), S(p^*)\}$?

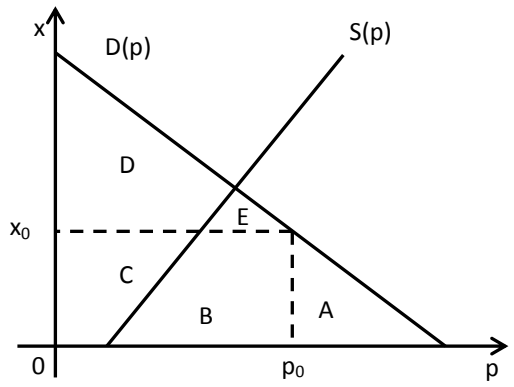


Figure 11.10 Demand and supply curves as a function of price.

11.4 *Flat component v.s. usage component* ⋆⋆⋆

As in Advanced Material, the simplified revenue maximization problem for the monopoly ISP is:

$$\begin{array}{ll} \text{maximize} & \sum_{t,f} \sigma_f^t u_f(D_f^t(h_f^t)) \\ \text{subject to} & \sum_f D_f^t(h_f^t) \le C, \ \ \forall t \\ \text{variables} & \{h_f^t\}. \end{array} \tag{11.10}$$

Let ${h_f^t}^*$ be the usage price that solves the above maximization problem, and the resulting volume consumption of consumption and the flat rate price are

$${x_f^t}^* = D_f^t({h_f^t}^*),$$

$${g_f^t}^* = \sigma_f^t u_f(D_f^t({h_f^t}^*)) - {h_f^t}^* {x_f^t}^*.$$

(a) Define $R_f^* = \sum_{t,f} {g_f^t}^*$ as the revenue from flat component, $R_s^* = \sum_{t,f} {h_f^t}^* {x_f^t}^*$

as the revenue from usage component. Prove that

$$\frac{R_f^*}{R_s^*} = \frac{\sum_{f,t} \sigma_f^t u_f(x_f^{t\,*})}{\sum_{t,f} \sigma_f^t u_f'(x_f^{t\,*}) x_f^{t\,*}} - 1.$$

(Hint: Use $\sigma_f^t u_f'(D_f^t(h_f^t)) = h_f^t$.)

(b) Show that if $u_f(x)$ is α-fair utility function ($\alpha \neq 1$) for all flow f, then

$$\frac{R_f^*}{R_s^*} = \frac{\alpha}{1-\alpha}.$$

(c) Argue that $h_f^{t\,*} = h^{t*}$, *i.e.*, the optimal price is independent of flow f.

11.5 *Braess' paradox* ⋆⋆⋆

Consider a road network as illustrated in Figure 11.11(b), on which 3000 drivers wish to travel from node Start to node End. Denote by x the number of travelers passing through edge Start $\to A$, and y the number of travelers passing through edge $B \to$ End. The travel time of each edge in minutes are labeled next to the corresponding edges. Suppose everyone chooses its route from Start to End by which the total travel time is minimized.

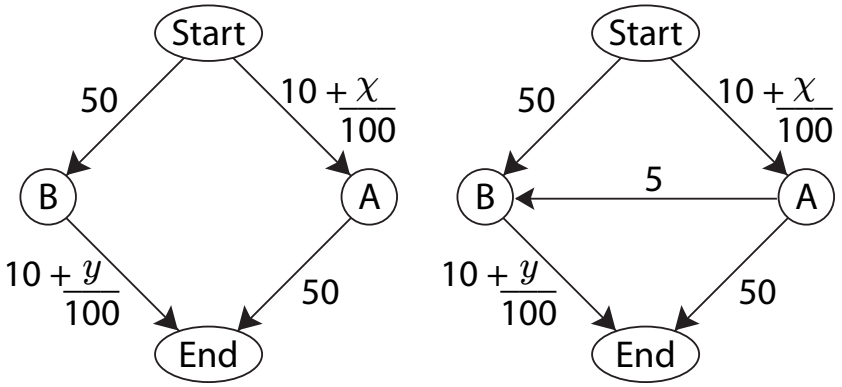


Figure 11.11 The Braess's paradox: dding a link hurts performance.

(a) What is the resulting traffic and the total travel time for each commuter?

(b) Suppose the government built a shortcut from node A and B with travel time labeled as illustrated in Figure 11.11(b). What is the resulting traffic and

the total traveling time for each commuter?

(c) This is the famous **Braess' paradox**. Suggest a way to avoid it.

12 How can I pay less for my Internet connection?

12.1 A Short Answer

12.1.1 Variants of Internet Pricing

ISPs charging residential consumers based on usage is just one corner of the overall landscape of Internet economics. There are many other key questions:

- The formation of the Internet is driven in part by economic considerations. Different ISPs form peering and transit relationships based on business and political decisions as much as on technical ones.
- The invention, adoption, and failure of Internet technologies are driven by the economics of vendor competition and consumer adoption.
- The investment of network infrastructure, from purchasing wireless licensed spectrum to deploying triple play broadband access, is driven by the economics of capital expenditure, operational expenditure, and return on investment.

The economics of the Internet are interesting because the technology-economics interactions are *bidirectional*: economic forces shape the evolution of technology, while disruptive technologies can rewrite the balance of economic equations. It is also challenging to study because of the lack of publicly available data on ISPs' cost structure and the difficulty of collecting well-calibrated consumer data.

There is a rapidly growing research field and industry practice on network access pricing. What we described on usage pricing in the last chapter, in the form of tiered and then metered/throttled plans, is just a starter.

In static pricing, there are several variants:

- The hourly rate model, *e.g.*, Mobinil in Egypt charges data connection by the hours of usage.
- Expected capacity pricing relies on resource allocation driven by the need of different sessions. Characterizing an application's needs, however, can be tricky, even after a period of performance observation during trials.
- Priority pricing, where you can pay more to get a higher speed, such as the priority pass service by SingTel in Singapore. A turbo mode of anti-throttling is also being considered in the U.S. for heavy users whose speed is throttled once usage exceeds some threshold. In an elaborate form, priority pricing

may even take the form of auction where the price reflects the negative externality imposed on other users by boosting your speed.

- Location-dependent pricing, which is also used in certain transportation industries, *e.g.*, downtown London and Singapore.
- Time-dependent pricing, which is also used in certain utility industries and will be elaborated in this chapter.
- Two-sided pricing, where an ISP charges either the content consumers or the content producers or both. It is used by *e.g.*, Telus in Canada and TDC in Denmark. This can also become an application-dependent pricing method.

In dynamic pricing, network access prices are continuously adjusted to reflect the state of the network. We will see that congestion control in Chapter 14 can be interpreted as a type of dynamic pricing. If user applications' performance needs, such as delay elasticity, are also taken into account, we can view congestion-dependent pricing as a generalization of time-dependent pricing.

In this chapter, we bring up several topics that illustrate some central themes in the field. One is charging based on *when* the Internet is used, and the other is *differentiating service qualities* by simply charging different prices. We will also explore the question of whom to charge through two-sided pricing. These are some of the possibilities to help the entire ecosystem, from consumers to ISPs, from content providers to ad agencies, move from the shadow of $10/GB to a win-win system.

12.1.2 Time-dependent pricing

Pricing based just on monthly bandwidth usage still leaves a timescale mismatch: ISP revenue is based on monthly usage, but peak-hour congestion dominates its cost structure. Ideally, ISPs would like bandwidth consumption to be spread evenly over all the hours of the day. **Time-Dependent usage Pricing** (TDP) charges a user based on not just "how much" bandwidth is consumed but also "when" it is consumed, as opposed to *Time-Independent usage Pricing* (TIP), which only considers monthly consumption amounts. For example, the day-time (counted as part of minutes used) and evening-weekend-time (free) pricing, long practiced by wireless operators for cellular voice services, is a simple 2-period TDP scheme. Multimedia downloads, file sharing, social media updates, data backup, and non-critical software downloads all have various degrees of time elasticity.

An idea as old as the cyclic patterns of peak and off-peak demand, TDP has the potential to even out time-of-day fluctuations in bandwidth consumption. As a pricing practice that does not differentiate based on traffic type, protocol, or user class, it also sits lower on the radar screen of network neutrality scrutiny. TDP time-multiplexes traffic demands. We will encounter traffic multiplexing several times in future chapters, such as cloud services in Chapter 16. It is a

counterpart to spatial multiplexing in Chapter 13 and to frequency multiplexing in Chapter 18.

Much of the pricing innovation in recent years has occurred outside the US. Network operators in highly competitive and lucrative markets, *e.g.*, in India and Africa, have adopted innovative dynamic pricing for voice calls:

- An African operator MTN started "dynamic tariffing," a congestion-based pricing scheme in which the cost of a call is adjusted every hour in each network cell depending on the level of usage. Using this pricing scheme, instead of a large peak demand around 8 pm, MTN Uganda found that many of its customers were waiting to take advantage of cheaper call rates, thus creating an additional peak at 1 am.
- A similar congestion-dependent pricing scheme for voice calls was also launched in India by Uninor. It offers discounts to its customers' calls based on the network traffic condition in the location of the call's initiation (*i.e.*, location-based pricing).
- Orange has been offering "happy hours" data plans during the hours of 8-9am, 12-1pm, 4-5pm, and 10-11pm.

The question we have to face here are: Can we effectively parametrize **delay sensitivity** in setting the right prices? Are users willing to defer their Internet traffic in exchange for a reduced monthly bill?

12.2 A Long Answer

12.2.1 Thinking about TDP

Usage-based pricing schemes use penalties to limit network congestion by reducing demand from individual heavy users. However, they cannot prevent the peak demand from all users concentrating during the same time periods. ISPs must provision their network in proportion to these peak demands, leading to a timescale mismatch: ISP revenue is based on monthly usage, but peak-hour congestion dominates its cost structure. Empirical usage data from typical ISPs show large fluctuations even on the timescale of a few minutes. Thus, usage can be significantly evened-out if TDP induces users to shift their demand by a few minutes. However, a simple two-period, time-dependent pricing is inadequate as it can incentivize only the highly price-sensitive users to shift some of their non-critical traffic. Such schemes often end up creating two peaks; one during the day and one at night.

In general, all static pricing schemes suffer from their inability to adapt prices in real time to respond to the usage patterns, and hence fail to exploit the dynamic range of delay tolerance as many types of applications proliferate.

Dynamic pricing, on the other hand, is better equipped to overcome these issues and does not require pre-classification of hours into peak and off-peak

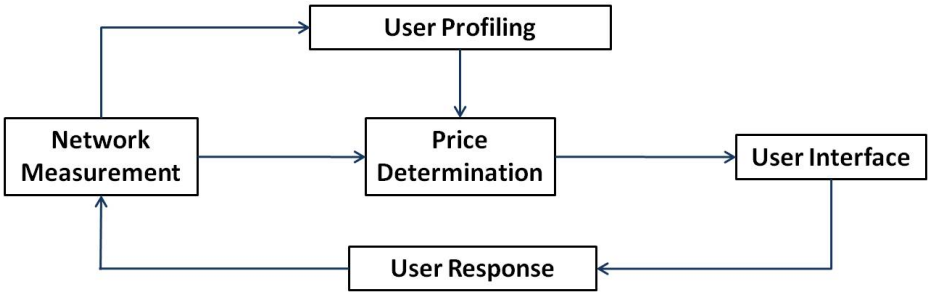


Figure 12.1 A simplified schematic of TDP architecture. The core computational module is price optimization that takes into account prediction of user reaction through the profiling module. The user interface needs to be user-friendly, and allow both user-generated decisions and auto-pilot decisions to run in real time on behalf of the user.

periods. However, the current dynamic time- or congestion-dependent pricing schemes are for voice traffic. They rely on simple heuristics and have been explored mainly for voice traffic, which is very different from data in its delay sensitivity, activity patterns, and typical duration. Unlike voice calls, certain classes of mobile data traffic offer greater delay tolerance. They can be completed either pre-emptively or in small chunks whenever the congestion conditions are mild. Users of such applications can therefore be incentivized to shift their usage with optimized, time-dependent pricing for their mobile data traffic.

Time-dependent pricing can be further extended to **congestion-dependent pricing** by shrinking TDP's timescale. Instead of on a timescale of hours or minutes, each period may be only several seconds when channel conditions or mobility may rapidly change congestion conditions. Even during busy hours and over heavily used spectra, there are occasional periods of time with little usage, which we call **flashy whitespaces**. ISPs can offer low spot prices in these less congested time slots, enabling cost-conscious users to wait for these low prices. In such cases (and for general timescales), TDP can be put on "auto-pilot" mode, where a user need not be bothered in real time once she preconfigures her usage requirements and expectations, *e.g.*, the maximum monthly bill, which applications should never be deferred, etc.

Pushing the auto-pilot TDP approach further, ISPs can offer intelligent flat-rate data plans to complement more traditional TDP pricing plans. Users may pay a flat rate in exchange for automated delaying of their traffic. The auto-pilot mode adjusts a user's traffic profile so that the user's charge under TDP is less than or equal to the flat rate which the user pays.

TDP, whose schematic is shown in Figure 12.1, is a traffic shaper that time-shifts capacity demands so that the **multiplexing effect** of a network can be most effectively leveraged. And the degree of freedom in this shaper is the pricing signals that change consumer behavior.

When determining optimal prices, an ISP tries to balance the cost of demand

exceeding capacity, *e.g.*, the capital expenditure of capacity expansion, with the cost of offering reduced prices to users willing to move some of their sessions to later times.

12.2.2 Modeling TDP

A user is modeled here as a set of application sessions, each with a **waiting function**: the willingness to defer that session for some amount of time under a price incentive for doing so. Pictorially, an ISP uses TDP to even-out the "peaks" and "valleys" in bandwidth consumption over the day. The ISP's problem is then to set its prices to balance capacity costs and costs due to price incentives, given its estimates of user behavior and the willingness to defer sessions at different prices.

Waiting functions are functions of both the reward and the amount of time deferred: $w = w(p, t)$, where p is the reward offered to the user for waiting t amount of time. Each application of each user in each period has a different waiting function; the users' willingness to defer applications depends on the type of application being deferred, the user's patience, and the period that the application is being deferred from. For instance, I might be generally more willing to defer downloading a software update than a YouTube video, and more willing to defer a YouTube video for 10 minutes at 8pm than at 11am.

For a constant p, $w(p, t)$ should decrease in t. And for a constant t, $w(p, t)$ should be increasing in p. Following the principle of diminishing marginal utility, we can say that w is concave in p with constant t. For waiting functions to be useful, we need some method for estimating them. To make this estimation easier, we *parametrize* the waiting functions. Thus, each waiting function has the same form but a different parameter; our job is then to estimate the waiting function parameters. For instance, we might take

$$w(p, t) = \frac{p}{(t+1)^{\beta}},$$

with the parameter $\beta \geq 0$ specifying how much the users' willingness to shift their traffic falls off with time. With large β, users do not want to defer for a long time; with smaller β, users are more willing to defer their traffic. Table 12.1 shows realistic β values for different applications. These are estimated from consumer surveys conducted in the United States and India.

In practice, it is impossible to estimate waiting function parameters for each type of application for each user during each period because there are too many of them. Instead, we use an aggregate approach: several applications for some users are assumed to have the same waiting function. For instance, I might be equally willing to defer a YouTube video at 9pm as my friend is willing to defer watching Hulu at 10pm. We then have the same waiting function parameters.

Now we know exactly what parameters need to be estimated: the waiting function parameters in each period and the fraction of traffic that they correspond to.

Table 12.1 Estimated patience indices from survey

	YouTube	Software Updates	Movie Downloads
U.S.	2.027	0.5898	0.6355
India (DP)	2.796	1.486	
India (no DP)	2.586	1.269	

DP: data plan

The next step is to do the actual estimation. We can estimate using the difference between traffic before TDP and traffic after TDP. This difference must be equal to the amount of traffic "deferred in", less the amount "deferred out". But we can now calculate the amount deferred in: it is the sum of the amount deferred from each period, which can be easily calculated using the waiting functions in these periods. Given the period in which an application session originated, we know how long the traffic was deferred and what reward was offered.

We can then write the difference between traffic volume before and after TDP as a function of the parameters to be estimated. These differences are now functions of the rewards offered and the parameters to be estimated. In a trial, we can offer a range of rewards and observed the corresponding difference in traffic before and after TDP, we can choose the waiting function parameters to fit with the data we observe (*e.g.*, using least-squares).

Then, given the estimation of waiting functions, the ISP needs to decide the price per time period. The ISP's decision can also be formulated in terms of rewards, *i.e.*, price discounts, as in our formulation. These rewards are defined as the difference between TIP and optimal TDP prices.

Finally, the ISP's objective is to minimize the weighted sum of the cost of exceeding capacity and of offering reduced prices (*i.e.*, rewards). The optimization variables are these rewards, which give users incentives to defer bandwidth consumption.

12.2.3 Implementing TDP

Taking a pricing idea from its mathematical foundation to a deployment in operational networks is not easy. The process involves computer simulations, testbed emulations, proof-of-concept implementations, small scale user trials, and eventually to large scale commercial trials. For example, in the case of TDP for mobile (and wireline) networks, an end-to-end system was implemented and consumer trials were carried out at Princeton University in 2012. The system involves many components:

- Graphic User Interface connected into the operating systems of iPhones, iPads, Android phones and tablets, and Windows PC.

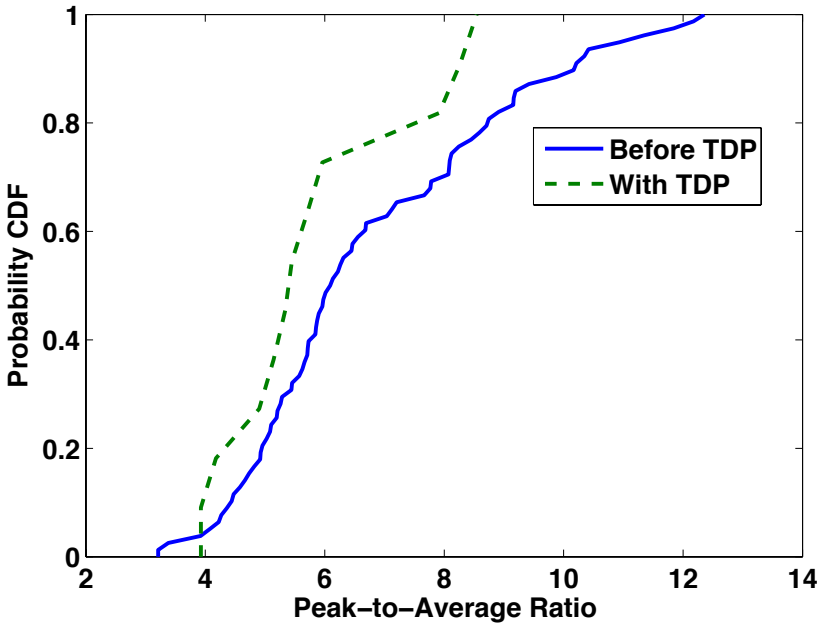


Figure 12.2 A cumulative distribution chart, using empirical data from a Princeton trial of TDP in 2012, shows that TDP reduces the peak-to-average ratios.

- Traffic measurement partly on end user devices and partly in network gateways, so as to ensure security and privacy needed.
- A database of usage, traffic, prices, and network conditions on servers.
- The computational engines of user profiling and price optimization, again split across end user devices, network gateways, and servers.
- Softwares that take the output of these computational engines to control network scheduling.

Initial results from this trial of 50 users indicate that TDP can be effective in reducing the network load, as measured by the *peak-to-average ratio* (PAR). This metric is defined as the peak amount of traffic in one hour, divided by the average traffic over that day. Figure 12.2 shows the distribution of daily PARs both before and after TDP was introduced. The maximum PAR decreases by 30% with TDP, and approximately 20% of the PARs before TDP are larger than the maximum PAR with TDP. Thus, TDP significantly reduced the peak-to-average ratio, flattening bandwidth demand over the day.

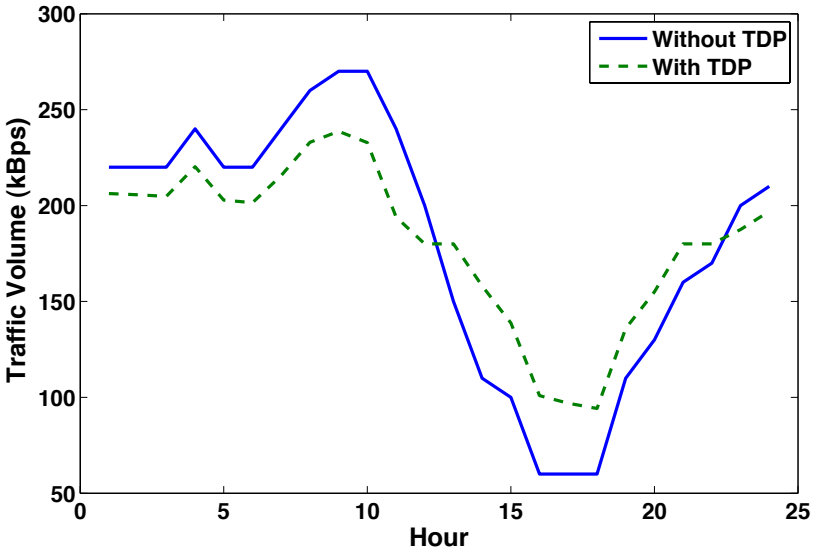


Figure 12.3 TDP and TIP traffic patterns for a sample mix of three classes of traffic with different delay tolerance. The aggregate traffic volume with TDP is much flatter than without.

12.3 Examples

We present some numerical examples first, before turning to a general formulation of the price optimization module in Advanced Material.

12.3.1 An illustration

We first walk through a larger scale simulation to visualize TDP, using the patience indices of 0.59 for cloud synchronization, 0.64 for multimedia download, and 2.03 for YouTube streaming. The usage distribution of the different traffic classes were taken from recent estimates, and the TIP data estimates was taken from empirical data from an ISP. We consider a system with 100 users and 24 one-hour time periods in each day. The ISP's marginal cost of exceeding capacity is set to \$0.30 per MB.

The results of the simulation are shown in Fig. 12.3, which gives the demand patterns before and after the use of time-dependent pricing. It demonstrates that TDP incentivizes users to shift their traffic, which brings the peaks and valleys closer, *i.e.*, improves the smoothness of the demand over time. The daily cost per user decreases from \$0.21 with TIP to \$0.16 with TDP, a 23% savings.

Figure 12.4 shows the optimal rewards (incentives) awarded for different times of the day. As might be expected, all hours with positive rewards are at or under capacity with TDP. Rewards are slightly higher in hours 14 and 15 than in

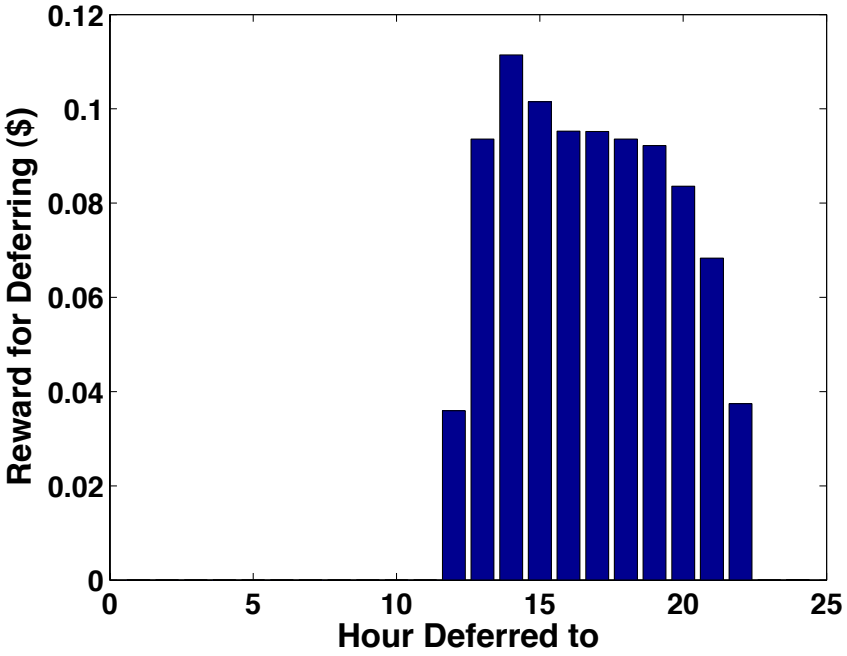


Figure 12.4 Rewards offered at different periods, computed by solving the time dependent price optimization problem that minimizes the overall ISP cost and incorporates user reaction predicted through waiting function estimation.

subsequent under-capacity hours; hours 14 and 15 represent the under-capacity times closest to the high-usage hours 1-13.

To quantify traffic's unevenness over 24 hours, we define the **residue spread** as the area between a given traffic profile and the ideal one, where the total usage remains the same but with usage constant (*i.e.*, "flattened") across periods in 24 hours. The residue spread decreases 44.8% (from 502.8 MB to 280.3 MB) with TDP. Maximum usage decreases from 270 to 239 kBps, and minimum usage increases from 60 to 94 kBps with TDP. Overused periods closer in time to underused ones have the greatest traffic reduction; users more easily defer for shorter times. Although TDP does help to even out traffic profiles, some users are impatient and some sessions are simply too time-sensitive to be deferred; thus the usage will never be perfectly "flat."

We next measure our model's sensitivity. Suppose that demand under TIP is unchanged, but the ISP incorrectly inferred users' waiting functions. We use $\beta = 2.586$, 0.8, and 0.3 for YouTube streaming, multimedia downloads, and cloud synchronization, respectively. The rewards for deferring change as in Table 12.2. They do not change significantly, because the estimated and perturbed patience

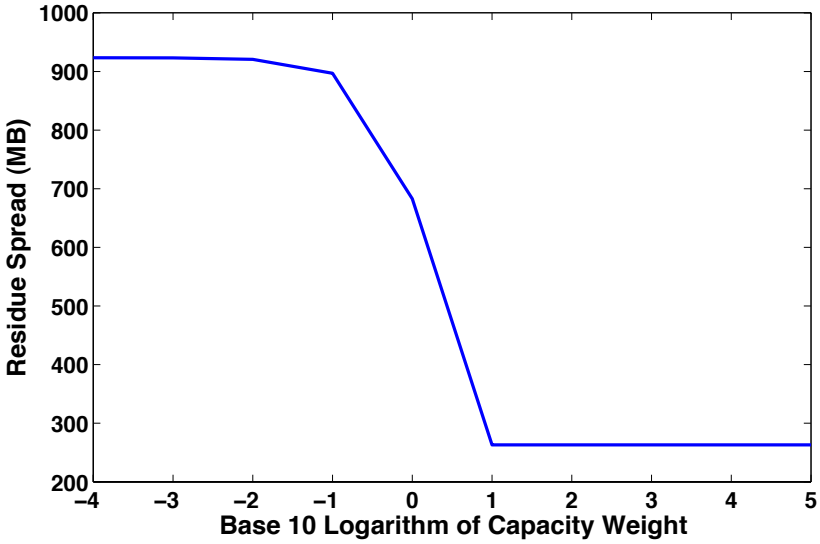


Figure 12.5 Residue spread for different costs of exceeding capacity. The ISP never entirely evens out traffic, even at the very high cost of exceeding capacity.

Table 12.2 Optimal rewards ($) and waiting function perturbation.

Period	1-11, 23-24	12	13	14	15	16
Original	0	0.04	0.09	0.11	0.10	0.10
Adjusted	0	0.04	0.08	0.11	0.10	0.10
Period	17	18	19	20	21	22
Original	0.10	0.09	0.09	0.08	0.07	0.04
Adjusted	0.10	0.09	0.09	0.08	0.06	0

indices are roughly the same. Residue spread decreases by 40.0% from 502.8 MB under TIP to 298.1 MB under TDP.

You would expect that when exceeding capacity is expensive, the ISP will offer large rewards to even out demand. Figure 12.5 shows residue spread with TDP versus the logarithm of a, the weight on the cost of exceeding capacity relative to the cost of handing out rewards. Residue spread decreases sharply for $a \in [0.1, 10]$, then levels out for $a \geq 10$. For $a \geq 10$, demand never exceeds capacity because the cost of exceeding capacity is too big.

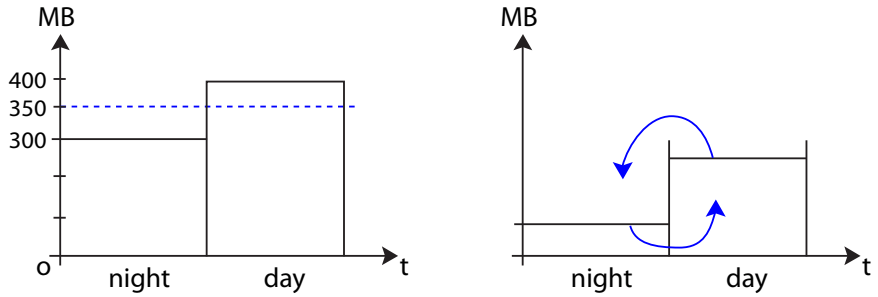


Figure 12.6 In a small numerical example illustrating time-dependent pricing over a single bottleneck link, the link capacity can handle 300 MB of demand. But day time demand exceeds that while night time demand under-utilizes capacity. TDP help provide pricing incentives for users to move some of their day time demand to night time. In general, we need to keep track of the demand shifted away from a given period and the demand shifted into that period.

	Night	Day
Email	200 MB	200 MB
File Downloads	100 MB	200 MB

Table 12.3 Volume of email and file download traffic during the night and day before TDP.

12.3.2 A small example

As illustrated in Figure 12.6, we now walk through a small, simplified numerical example that bridges the general discussion in A Long Answer to the symbolic representation in Advanced Material. Suppose we have just two periods (*e.g.*, night and day) and that we are trying to determine the optimal prices for these periods.

First, we must characterize the types of traffic found in these two periods. We assume two types of applications, one which users are very impatient to use and one for which they are more patient. For instance, we could have email or twitter (impatient) and movie downloads or cloud synchronization (patient).

The volume of traffic taken up by these two applications before TDP is shown in Table 12.3. We assume that the ISP is at present charging users \$0.01/MB (\$10/GB) during both the night and the day.

We now need to quantify users' willingness to shift some of their traffic from one period to the other. This willingness is of course influenced by the reward

	Shift to Night	Shift to Day
Email	$\frac{p_n}{4}$ probability, $200\frac{p_n}{4}$ shifted	$\frac{p_d}{4}$ probability, $200\frac{p_d}{4}$ shifted
File Downloads	$\frac{p_n}{2}$ probability, $200\frac{p_n}{2}$ shifted	$\frac{p_d}{2}$ probability, $100\frac{p_d}{2}$ shifted

Table 12.4 Probability of shifting and expected amount of shifting of email and file download traffic to the night or day.

offered in the other period. Let p_n be the reward during the night and p_d the reward during the day. We express users' "willingness to wait" as the probability that the user will wait. To simplify the presentation, let us say this probability is proportional to the reward reaped by waiting. The probabilities of shifting and the corresponding expected amount of traffic shifted are summarized in Table 12.4.

We now formulate the optimization problem. First, consider the cost of offering rewards, which is just the reward per unit of traffic times the amount shifted into a given period. This cost can thus be expressed as

$$p_n\left(200\frac{p_n}{4}+200\frac{p_n}{2}\right)+p_d\left(200\frac{p_d}{4}+100\frac{p_d}{2}\right)=150p_n^2+100p_d^2. \tag{12.1}$$

Next, we quantify the cost of exceeding capacity. We model this cost as linear and assume that the capacity is 350 MB. Thus, the ISP exceeds capacity during the day but not during the night. For each GB over capacity, the ISP must face a cost of \$1.

We now need to find expressions for the volume of traffic during the night and day under TDP. Consider the volume of traffic during the night. Before TDP, it is $200+100=300$ MB. The amount of traffic shifted into the night is

$$200\frac{p_n}{4}+200\frac{p_n}{2}=150p_n,$$

and the amount of traffic shifted from the night into the day is

$$200\frac{p_d}{4}+100\frac{p_d}{2}=100p_d.$$

Thus, the amount of traffic during the night under TDP, in MB, is

$$300+150p_n-100p_d,$$

and the cost of exceeding capacity during the night is

$$\max\left\{0,300+150p_n-100p_d-350\right\}=\max\left\{0,-50+150p_n-100p_d\right\},$$

where we use monetary units of \$10/GB and traffic volume units of MB. We can find a similar expression for the cost of exceeding capacity during the day using the same line of reasoning.

Finally, we have the optimization problem of minimizing the following objective function:

$$150p_n^2+100p_d^2+\max\left\{0,-50+150p_n-100p_d\right\}+\max\left\{0,50-150p_n+100p_d\right\},$$

over the non-negative variables of $\{p_n, p_d\}$, *i.e.*, the per-unit reward amount, or equivalently, the time-dependent prices.

Solving this optimization problem, we obtain $p_d = 0$, $p_n = 0.33$. Thus, the ISP discounts prices during the night by \$3.33/GB. Intuitively, this makes sense: rewards during the day should be lower, so that users are induced to shift their traffic from the day into the night. Indeed, with this pricing scheme, the ISP operates at capacity, with a traffic volume of 350 MB during both the day and the night.

12.4 Advanced Material

12.4.1 TDP price optimization formulation

Computing the optimal prices per time period, with user reaction anticipated through waiting function estimation, is a key module in TDP. We consider a version of this problem formulation that generalizes the small numerical example we just saw.

Let X_i denote demand in period i under TIP. The phrase "originally in period i" means that under TIP, this session occurs in period i. Suppose that the ISP divides the day into n periods, and that its network has a single bottleneck link of capacity C, where C is the total amount of traffic that the link can carry in one time period. This link is often the aggregation link out of the access network, which has less capacity compared to aggregate demand and is often oversubscribed by a factor of five or more. The cost of exceeding capacity in each period i, capturing both customer complaints and expenses for capacity expansion, is denoted by $f(x_i - C)$, where x_i is usage in period i. This cost is often modeled as piecewise-linear and convex.

Each period i runs from time $i-1$ to i. A typical period lasts 10-30 minutes. Sessions begin at the start of the period, an assumption readily modified to a distribution of starting times. The time between periods i and k is given by $i-k$, which is the number $b \in [1, n]$, $b \equiv i - k \pmod{n}$. If $k > i$, $i - k$ is the time between period k on one day and period i on the next day.

For each session j originally in period i, define the **waiting function** $w_j(p, t) : \mathbb{R}^2 \to \mathbb{R}$, which measures the user's willingness to wait t amount of time, given reward p. Each session j has capacity requirement v_j, so $v_j w_j(p, t)$ is the amount of session j deferred by time t with reward p. To ensure that $w_j \in [0, 1]$ and that the calculated usage deferred out of a period is not greater than demand under TIP, we normalize the w_j by dividing it by the sum over possible times deferred t of $w_j(P, t)$. Here P is the maximum possible reward offered. The notation $j \in k$ refers to a session j originally in period k (in the absence of time-dependent pricing).

Now we are ready to state the optimization problem. First consider the cost of paying rewards in a given period i. The amount of usage deferred into period

i is $\sum_{k \neq i} y_{k,i}$, where $y_{k,i}$ is the amount of usage deferred from period k to period i. Consider a session $j \in k$. The amount of usage in session j deferred from period k to period i is $v_j w_j(p_i, i-k)$, since such sessions are deferred by $i-k$ amount of time. Thus, $y_{k,i} = \sum_{j \in k} v_j w_j(p_i, i-k)$, and the ISP's total cost of rewarding all sessions in period i is $p_i \sum_{k \neq i} \sum_{j \in k} v_j w_j(p_i, i-k)$.

Now consider the cost of exceeding capacity. It is $af(x_i - C)$, but how much is x_i? It is the original amount minus the amount moved out of period i by TDP plus the amount moved into period i by TDP:

$$x_i = X_i - \sum_{j \in i} v_j \sum_{k=1, k \neq i}^{n} w_j(p_k, k-i) + \sum_{k=1, k \neq i}^{n} \sum_{j \in k} v_j w_j(p_i, i-k). \tag{12.2}$$

The ISP's total cost function for period i is then

$$p_i \sum_{k \neq i} \sum_{j \in k} v_j w_j(p_i, i-k) + af(x_i - C),$$

and summing over all periods, indexed by i, yields the desired formulation below:

$$\begin{array}{ll} \text{minimize} & \displaystyle\sum_{i=1}^{n} p_i \left(\sum_{k=1, k \neq i}^{n} \sum_{j \in k} v_j w_j(p_i, i-k) \right) + a \sum_{i=1}^{n} f(x_i - C) \\ \text{subject to} & \displaystyle x_i = X_i - \sum_{j \in i} v_j \sum_{k \neq i}^{n} w_j(p_k, k-i) + \sum_{k \neq i}^{n} \sum_{j \in k} v_j w_j(p_i, i-k), \forall i \\ \text{variables} & p_i \geq 0, \ \ \forall i. \end{array} \tag{12.3}$$

If the $w(p,t)$ are increasing and concave in p, and f is convex, the ISP's price, or equivalently, reward optimization problem is a convex optimization problem, as defined in Chapter 4.

The above is a static traffic model, which does not include stochastic arrival of new sessions. There are also dynamic models with stochastic arrivals. The f ixed-size version can be extended to sessions with fixed duration and to online adjustments that track user behavior.

12.4.2 Paris metro pricing

We have been investigating "how to charge" and "how much to charge." There is also the related question of "what to charge." You would think that a different service, *e.g.*, express delivery, would command a different price. That is true, but it also works the other way: a different price can also lead to a different service. This idea of intelligent pricing is "pricing-induced quality differentiation." Whenever a service's quality depends on how crowded its consumers are, we can simply use higher prices to reduce demand in certain portions of the service. This creates a new category of service tier.

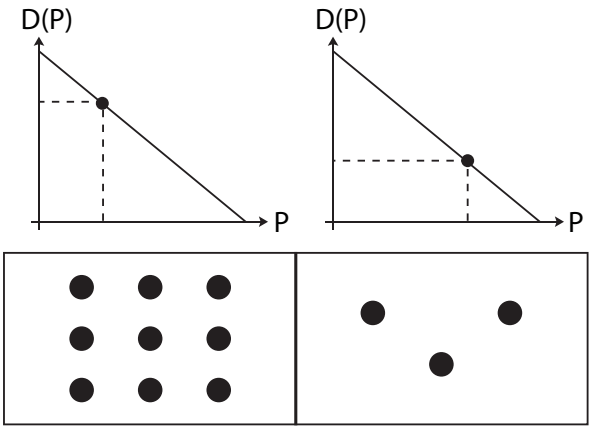


Figure 12.7 Paris metro pricing creates service differentiation through price differentiation. The left graph shows that lower price increases demand and the utilization, thus reducing utility for those in that service tier. The right graph shows that higher price reduces demand and utilization, thus increasing utility for those in that service tier.

Pricing changes consumer behavior and network congestion, and if different congestion levels imply different service grades, *i.e.*, utility depends on utilization, we can complete the feedback loop where different prices automatically lead to different services. This line of reasoning is exemplified in the Paris metro pricing, a thought experiment that Odlyzko presented in 1998 for Internet pricing.

Consider a metro (*i.e.*, subway or underground train) service where two passenger cars are identical, but are charged differently: car A's charge is twice as much as car B's. You might protest: how can they charge differently for the exact same service? Well, they are *not* the same as soon as consumers react to the different prices. Compared to car A, car B will be more crowded as the price is lower and demand is thus higher. Since the congestion level is a key component of the quality of service in riding a metro car, we can call car A first-class and car B coach-class, and their different prices self-justify. This phenomenon applies to all cases where utility depends on utilization.

Even though Paris metro pricing is not yet widely implemented in either the transportation or the Internet industry, it illustrates an interesting feedback loop between price-setters (a metro company, or an ISP) and intelligent agents (metro riders choosing between cars, or iPhones network interface cards choosing between WiFi and 3G connections) reacting to the prices. Let us examine Paris Metro Pricing from an efficiency point of view, and discover the *downside* of resource pooling.

Suppose an ISP creates two tiers of services on a link: tier A and tier B, and has two possible prices to offer: p_1 and p_2, with $p_2 > p_1$. Demands D_A and D_B can be such that the following statements are true:

- If the ISP offers p_1, the demand will be $D_A(p_1)+D_B(p_1)$, which is too big and causes too much congestion for tier A users to find the service A useful. So tier A drops out at this price p_1, and the demand becomes just $D_B(p_1)$, with revenue $p_1 D_B(p_1)$ to the ISP.
- If the ISP offers p_2, the demand will be $D_A(p_2) + D_B(p_2)$, which is clearly smaller than $D_A(p_1) + D_B(p_1)$ since price p_2 is higher than p_1. Let us say it is small enough that tier A users stay. The revenue becomes $p_2(D_A(p_2) + D_B(p_2))$ to the ISP.

So the ISP must choose between the two prices, and the revenue is

$$\max\{p_1 D_B(p_1), p_2(D_A(p_2) + D_B(p_2))\}. \tag{12.4}$$

Now, consider a different scheme. The ISP divides the link into two equal parts. In one part, it sets the price to be p_1 and gets revenue $p_1 D_B(p_1)$. In the other part, it sets the price to be p_2 and gets revenue $p_2(D_A(p_2) + D_B(p_2))$. For any demand functions (D_A, D_B), the prices can be set sufficiently high so that each of these demands can fit into half of the link capacity. Now the revenue becomes the *sum*:

$$p_1 D_B(p_1) + p_2(D_A(p_2) + D_B(p_2)), \tag{12.5}$$

clearly higher than (12.4).

This **anti-resource-pooling** property in revenue maximization through flexible pricing is the opposite of the principle of statistical multiplexing used in achieving the economy of scale, smoothing time-varying demand, and avoiding fragmentation of resources, as we will see in Chapter 13 and Chapter 16. In contrast to those chapters, the purpose here is to maximize revenue through multi-class pricing. Anti-resource-pooling turns out to be the right way to go as it provides more granular control of pricing of different parts of the overall resource.

12.4.3 Two-sided pricing

In two-sided Internet pricing models, the price of connectivity is shared between content providers (CP) and end users (EU). ISPs are just the middle man proving the connectivity between CP and EU. Some ISPs have started exploring this question of "whom to charge". A "clearing house" of connectivity exchange market will become a major extension of the 1-800 model of phone call services in the US, which charges the callee rather than the caller.

The tradeoff in the resulting economic benefits remain to be quantified. Intuitively, end-users' access prices are subsidized and the ISPs have an additional source of revenue. More surprisingly, content-providers may also stand to gain from two-sided pricing if subsidizing connectivity to end-users translates into a net revenue gain through a large amount of consumption. However, the gain to content providers is dependent upon the extent to which content-provider

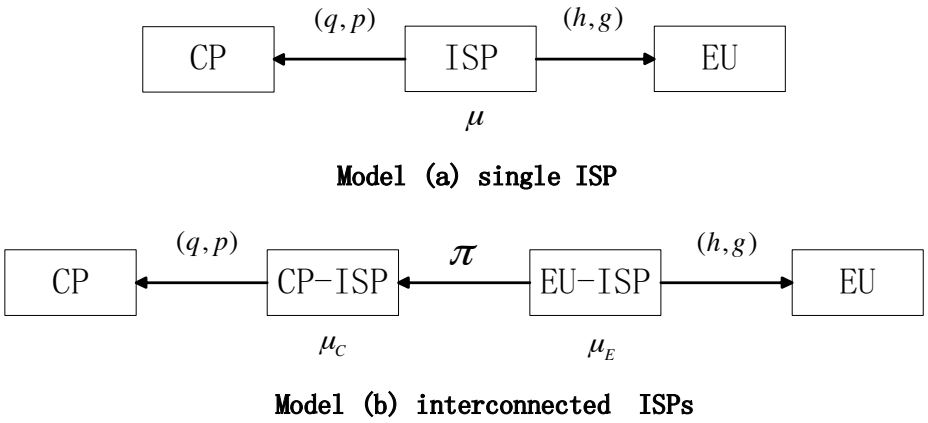


Figure 12.8 (a): Single representative ISP charging CP a usage-price q and flat-price p and charging EU a usage-price h and flat-price g. (b): EU-ISP charging a transit price π to CP-ISP. Capacity cost of EU-ISP is μ_E and and CP-ISP is μ_C.

payment translates into end-users' subsidy, and the demand elasticities of the consumers. The precise gains to the three entities will depend upon the interplay among them and their respective bargaining power.

The economic interaction on the flow between the EU and the CP includes peering and transit arrangements between multiple ISPs, operating the links between the CP and the EU, and the access fee charged by the ISPs to the EU and the CP. We can consider an eyeball ISP, referred to as EU-ISP, charging an access price of $g+hx$ to the EU and a content ISP, referred to as CP-ISP, charging an access price of $p+qx$ to the CP. The EU-ISP and the CP-ISP can collaborate when charging the access fees to the EU and CP, or act independently. In a homework problem, we will model the collaborate through a representative ISP as shown in Figure 12.8.

Further Reading

There is a growing literature on Internet access pricing, including congestion pricing that we focused on.

1. A short survey of some common topics in transportation networks and the Internet can be found at:

[Kel09] F. P. Kelly, "The mathematics of traffic in networks," *Princeton Companion to Mathematics*, Princeton University Press, 2009.

2. Our treatment of time dependent pricing for Internet access follows

[JHC11] C. Joe-Wong, S. Ha, and M. Chiang, "Time dependent pricing: Feasi-

bility and benefits," *Proceedings of IEEE International Conference of Distributed Computing Systems*, 2011.

3. A presentation of the Paris metro pricing method can be found at:
[Odl98] A. Odlyzko, "Paris metro pricing for the Internet," *Proceedings of ACM Conference on Electronic Commerce*, 1998.

4. A standard reference of two-sided pricing mentioned in a homework on whom to charge is
[RT06] J. C. Rochet and J. Tirole, "Two-sided markets: A progress report," *The RAND Journal of Economics*, vol. 35, no. 3, pp. 645-667, 2006.

5. We have assumed a lot on how people make decisions based on pricing signals in the last two chapters. These assumptions often do not hold. An accessible and insightful survey is the recent book:
[Kah11] D. Kahneman, *Thinking, Fast and Slow*, FSG Publisher, 2011.

Problems

12.1 *Time dependent usage pricing* ⋆⋆

An ISP tries to even out the capacity demand over day and night by rewarding its users for delaying their data transmission. There are two types of users, type A and type B, which have different willingness to delay their data transmission. Originally the demand during daytime is in total $v_{A,day} + v_{B,day} = 14$GB, which consists of $v_{A,day} = 8$GB from type A users and $v_{B,day} = 6$GB from type B users. The demand during night time is in total $v_{A,night} + v_{B,night} = 5$GB, which consists of $v_{A,night} = 2$GB from type A users and $v_{B,night} = 3$GB from type B user.

Since the ISP has capacity $C = 10$GB and the marginal cost of exceeding capacity is \$1 per GB, it provides reward \$$p$ per GB for daytime users to delay their data transfer until night time. Let $w_A(p), w_B(p)$ be the proportion of data from type A,B users respectively to be delayed from day time to night time, which is specified as follows:

$$w_A(p) = 1 - \exp\left(-\frac{p}{p_A}\right),$$

$$w_B(p) = 1 - \exp\left(-\frac{p}{p_B}\right).$$

where parameters $p_A = 4, p_B = 2$.

The ISP wishes to find the reward price p^* which minimizes its total cost, *i.e.*, sum of cost due to exceeding capacity and the rewards given out.

(a) What is the formulation of the minimization problem?

(b) Solve p^* numerically by plotting the curve of profit as a function of reward price p.

12.2 *User type estimation* ⋆

Consider the same model in the above problem, except that now the values $v_{A,day}, v_{B,day}, v_{A,night}, v_{B,night}$ are unknown. Suppose that originally the demand during daytime is in total 17 GB, and after announcing reward price of \$0.30 per GB the demand during daytime reduces to 15.2 GB in total. What is $v_{A,day}, v_{B,day}$?

12.3 *TDP for smart grid demand response* ⋆⋆

Smart grid providers often set time-dependent prices for energy usage. This problem considers a simplified example with two periods, the day-time and the night-time. The provider can set different prices for the two periods, and wishes to shift some night time usage to daytime. The energy provider always offers the full price during the night, and offers a reward of \$$p$/kWh during the day.

Suppose that with uniform (time-independent) prices, customers vacuum at night, using 0.2 kWh, and also watch TV, using 0.5 kWh, and do laundry, using 2 kWh. During the day, customers use 1 kWh. The probability of users shifting vacuum usage from the night to the day is

$$1 - \exp\left(-\frac{p}{p_V}\right), \tag{12.6}$$

where $p_V = 2$, and the probability of shifting laundry to the daytime is

$$1 - \exp\left(-\frac{p}{p_L}\right), \tag{12.7}$$

where $p_L = 3$. Users never shift their TV watching from the night to the day.

Suppose that the electricity provider has a capacity of 2 kWh during the night and 1.5 kWh during the day. The marginal cost of exceeding this capacity is \$1/kWh. Assume that energy costs nothing to produce until the capacity is exceeded.

(a) Compute the expected amount vacuum and laundry energy usage (in kWh) that is shifted from the night to the day, as a function of p.

(b) Find the reward p which maximizes the energy provider's profit.

(c) Suppose that if vacuum or laundry usage is shifted from the night to the day, it is shifted by 12 hours. Compute the expected time shifted of vacuum and

laundry using $p = p^*$, the optimal reward found above.

12.4 *A Paris metro pricing model* ⋆⋆⋆

Consider a metro system where two kinds of services are provided: Service class 1 and service class 2. Let p_1, p_2 be the one-off fee charged per user when accessing service class i. Suppose each user is characterized by a valuation parameter $\theta \in [0, 1]$ such that its utility of using service class i is

$$u_\theta(i) = (V - \theta K(Q_i, C_i)) - p_i,$$

where V is the maximum utility of accessing the service, $K(Q_i, C_i)$ measures the amount of congestion of service class i, given $Q_i \geq 0$ as the proportion of users accessing service class i, with $\sum_i Q_i = 1$, and $C_i \geq 0$ is the proportion of capacity allocated to service class i, with $\sum_i C_i = 1$.

At the equilibrium, *i.e.*, no user switches from his selection, $u_\theta(i)$ is merely a linear function of θ. Suppose the equilibrium is illustrated as in Figure 12.9.

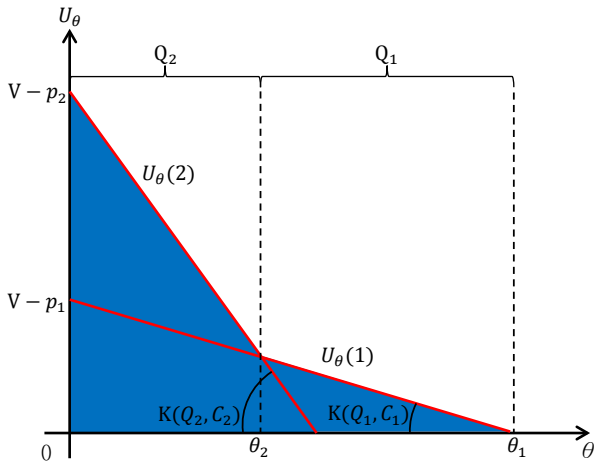


Figure 12.9 Illustration of equilibrium in PMP.

(a) Let θ_1 be the user who is indifferent to joining the first service class or opting out of all the services, θ_2 be the user who is indifferent to joining the first service class or the second service class, $F(\theta)$ be the cumulative distribution

function of θ. Show that

$$\begin{aligned} Q_1 =& F(\theta_1) - F(\theta_2), \\ Q_2 =& F(\theta_2), \\ V - p_1 =& \theta_1 K(Q_1, C_1), \\ p_1 - p_2 =& \theta_2 (K(Q_2, C_2) - K(Q_1, C_1)). \end{aligned}$$

(b) Assume θ be uniform distributed, *i.e.*, $F(\theta) = \theta$, and the congestion function defined as

$$K(Q, C) = \frac{Q}{C}.$$

Solve θ_1, θ_2 as a function of V, p_1, p_2.

(Hint: Try $\frac{p_1 - p_2}{V - p_1}$. You may define intermediate symbols such as $k = \frac{p_1 - p_2}{V - p_1}$ during derivation before the formulas become too complicated.)

(For details, see C. K. Chau, Q. Wang, and D. M. Chiu, "On the Viability of Paris Metro Pricing for Communication and Service Networks," *Proc. IEEE Infocom*, 2010.)

12.5 *Two-sided pricing* ⋆⋆

Consider the model where an ISP charges content provider (CP) usage-price h_{CP} and flat-price g_{CP} and charges end user (EU) usage-price h_{EU} and flat-price g_{EU}. Here we assume zero flat-price $g_{CP} = g_{EU} = 0$. Let μ be the cost of provisioning capacity. The demand functions of CP and EU, denoted as D_{CP}, D_{EU} respectively, are given as follows:

$$D_{EU}(h_{EU}) = \begin{cases} x_{EU,max}(1 - \frac{h_{EU}}{h_{EU,max}}) & \text{, if } 0 \le h_{EU} \le h_{EU,max} \\ 0, & \text{, if } h_{EU} > h_{EU,max} \end{cases}$$

$$D_{CP}(h_{CP}) = \begin{cases} x_{CP,max}(1 - \frac{h_{CP}}{h_{CP,max}}) & \text{, if } 0 \le h_{CP} \le h_{CP,max} \\ 0, & \text{, if } h_{CP} > h_{CP,max}. \end{cases}$$

The parameters are specified as follows:

$$\begin{aligned} h_{CP,max} &= 2.0\mu, \\ h_{EU,max} &= 1.5\mu, \\ x_{CP,max} &= 1.0a, \\ x_{EU,max} &= 2.0a. \end{aligned}$$

The ISP maximizes its profit by the following maximization problem

$$\begin{array}{ll} \text{maximize} & (h_{CP} + h_{EU} - \mu)x \\ \text{subject to} & x \leq \min\{D_{CP}(h_{CP}), D_{EU}(h_{EU})\} \\ \text{variables} & x \geq 0, h_{CP} \geq 0, h_{EU} \geq 0. \end{array} \tag{12.11}$$

Find x^*, h^*_{CP}, h^*_{EU} which maximizes the above maximization problem.

13 How does traffic get through the Internet?

We have mentioned the Internet many times so far, and all the previous chapters rely on its existence. It is about time to get into the architecture of the Internet, starting with these two chapters on the TCP/IP foundation of the Internet.

13.1 A Short Answer

We will be walking through several core concepts behind the evolution of the Internet, providing the foundation for the next four chapters. So the "short answer" section is going to be longer than the "long answer" section in this chapter. It is also tricky to discuss historical evolution of technologyies like the Internet. Some of what we would like to believe as inevitable result from careful design are actually the historical legacy of accidents, or the messy requirements of backward compatibility, incremental deployability, and economic incentives. It is hard to argue about what could have happened, what could have been alternative paths in the evolution, and what different tradeoffs might have been generated.

13.1.1 Packet switching

The answer to this chapter's question starts with a fundamental idea in designing a network: when your typical users do not really require a *dedicated* resource, you should allow users to *share* resources. The word "user" here is used interchangeably with "session." The logical unit is an application session rather than a physical user. For now, assume a session has just one source and one destination, *i.e.*, a **unicast session**.

In the case of routing, the resource lies along an entire path from one end of a communication session, the sender, to the other end, the receiver. We can either dedicate a fixed portion of the resources along the path to each session, or we can (a) mix and match packets from different sessions and (b) share all the paths. This is the difference between **circuit-switched** and **packet-switched** networks.

Before the 1960s, networking was mostly about connecting phone calls in the circuit-switched Public Switched Telephone Networks (**PSTN**). There continued

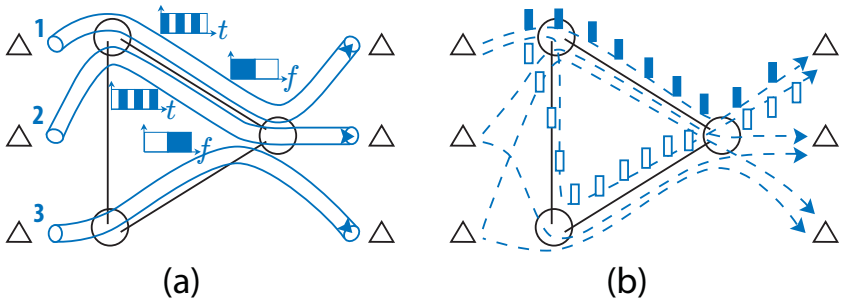


Figure 13.1 A simple network with 3 interconnected routers and 3 sessions. (a) circuit switching: each session gets a dedicated circuit, either a portion of each time slot t or a fraction of the frequency band f. (b) packet switching: each session sends packets along one or more paths (when there are packets to send) and all paths are shared.

to be active research all the way to the early 2000s, including dynamic routing as you will see in a homework problem.

A revolution, which came to be known as the Internet, started during the 1960s-1970s that shifted to packet-switching as the fundamental paradigm of networking. In the early 1960s, researchers such as Kleinrock formally developed the idea of chopping up a session's messages into small packets, and sending them along possibly different paths, with each path shared by other sessions. Figure 13.1 contrasts circuit switching with packet switching. Each circuit in circuit-switching may occupy either a particular frequency band or a dedicated portion of timeslots. In contrast, in packet switching, there is no dedicated circuit for each session. All sessions have their packets sharing the paths.

In 1969, sponsored by the US Advanced Research Project Agency (ARPA) through program manager Roberts, UCLA and three other institutions put together the first prototype of a packet-switched network, which came to be known as the **ARPANet**. The ARPANet started to grow. In 1974, Cerf and Kahn developed a **protocol**, *i.e.*, a set of rules for communication among the devices, for packet switched networks, called **TCP/IP**. This protocol enabled scalable connectivity in the ARPANet. In 1985, the US National Science Foundation (NSF) took over the next phase of development, sponsoring the creation and operation of an ever-increasing *network of networks*. This **NSFNet** grew dramatically in the following years, and by 1995 NSF decommissioned the NSFNet as commercial interests were strong enough to sustain the continuous expansion of this inter-connected network. Indeed, by 1994, the World Wide Web and web browser user-interface matured, and the world quickly moved into commercial

applications built on top of this network, known by then as the Internet. Now the Internet has blossomed into an essential part of how people live, work, play, talk, and think. There are now more Internet connected devices than people in the world, and it is projected that by 2020 there will be 6 times as many connected devices as people. It has been a truly amazing five decades of technology development.

The debate between dedicated resource allocation and shared resource allocation runs far and deep. In addition to circuit vs. packet switching here and orthogonal vs. non-orthogonal resource allocation in Chapter 1, we can also see three more special cases of this design choice: client-server vs. peer-to-peer, local storage vs. cloud services, and contention-free scheduling vs. random access: three important topics in Chapters 15, 16, and 18, respectively.

There is one big advantage of circuit switching, or dedicated resource allocation in general: *guarantee of quality*. As each session gets a circuit devoted to it, throughput and delay performance are accordingly guaranteed, and jitter (variance of delay) is very little. In contrast, in packet switching, a session's traffic is (possibly) split across different paths, each of which is shared with other sessions. Packets arrive out of order and need to be re-ordered at the receiver. Links may get congested. Throughput and delay performance become uncertain. Internet researchers call this **best effort** service that the Internet offers, which is perhaps more accurately described as *no effort* to guarantee performance.

On the other hand, there are two big advantages of packet switching: *ease of connectivity* and *scalability due to efficiency*.

Ease of connectivity is easy to see: there is no need to search for, establish, maintain, and eventually tear down an end-to-end circuit for each session.

Scalability here refers to the ability to take on many diverse types of sessions, some long-duration, others short-bursts, and to take on many of them. There are two underlying reasons for the efficiency of packet switching, which in turn leads to high scalability. These two reasons correspond to the "many sessions share a path" feature and the "each session can use multiple paths" feature of packet switching, respectively. We call these two features statistical multiplexing and resource pooling:

- *Statistical multiplexing*: packet switching can flexibly map demand of capacity onto supply of capacity. This suits the dynamic, on-demand scenarios with bursty traffic. In particular, when a source is idle and not sending any traffic onto the network, it does not occupy any resources.
- *Resource pooling*: this one takes a little math to demonstrate, as we will in a homework problem. But the basic idea is straightforward: instead of having two sets of resources (*e.g.*, two links' capacities) in isolation, putting them into one single pool lowers the chance that some demand must be turned down because one set of resources is fully utilized.

In the end, the abilities to easily provide connectivity and to scale up with many diverse users won the day, although that was not clear until the early 2000s.

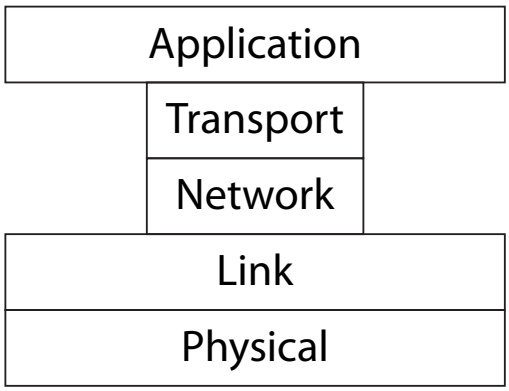


Figure 13.2 Modularization in networking: A typical model of layered protocol stack. Each layer is in charge of a particular set of tasks, using the service provided by the layer from below and in turn providing a service to the layer above. The horizontal lines that separate the layers represent some kind of limitation of what each layer can see and can do. Over the years, the applications have evolved from file transfer based on command line inputs to all that we experience today. The physical and link layer technologies have evolved from 32 kbps dial-up modem to 10 Gbps optic fibers and 100 Mbps WiFi. The two middle layers, however, remained largely unchanged over the years. They are the "thin waist" of the "hour-glass" model of the protocol stack.

Compared to quality guarantee, which is certainly nice to have, these properties are *essential* to have for a dynamic and large network like the Internet. Once the network has grown in an easy and scalable way, we can search for other solutions to take care of quality variation. But you have to grow the network *first*, in terms of the number of users and the types of applications. This is a key reason why IP took over the networking industry and packet switching prevailed, despite alternative designs in protocols (that we will not cover here) like X.25, ATM, frame relay, ISDN, etc.

13.1.2 Layered architecture

Managing a packet-switched network is complicated. There are many tasks involved, and each task requires a sequence of communication and computation called a **protocol** to control. It is a natural practice when engineering such a complex system to break it down into smaller pieces. This process of **modularization** created the **layered protocol stack** for the Internet. The idea of modularizing the design provided not just economic viability through the business models of different companies specializing in different layers, but also a special kind of robustness to unforeseen technologies and innovations that may ride on the Internet. This evolvability is further enhanced by the overlay net-

works that can create new network topologies and functionalities on top of the Internet connectivity, as we will see in Chapter 15.

A typical layered protocol stack is shown in Figure 13.2. TCP/IP sits right in the middle of it. Over the short span of Internet evolution, the physical medium's transmission speed went up more than 30,000 times, and the applications went from command-line-based file transfer to Netflix and Twitter. Yet the Internet itself continued to work, thanks in large part to the "thin waist" of TCP/IP that stayed mostly the same as the applications and the communication media kept changing.

Each layer provides a service to the layer above, and uses a service from the layer below. For example, the transport layer provides an end-to-end connection, running the services of session establishment, packet reordering, and congestion control, to the application layer above it that runs applications such as the web, email, and content sharing. In turn, the transport layer takes the service from the network layer below it, including the connectivities established through routing. The link layer is charged with controlling the access to the communication medium, and the physical layer controls the actual transmission of information on the physical medium.

There are functional overlaps across layers. For example, the functionality of error control is allocated to many layers: there is error control coding in the physical layer, hop-by-hop retransmission at the link layer, multipath routing for reliability in the network layer, and end-to-end error check at the transport layer. Functional redundancy is not a design bug, it is there by design, paying the price of efficiency reduction for robustness and clear boundary of layers.

How to allocate functionalities among the layers and put them back together at the right interface and timescale? That is the question of **network architecture** that we will continue to explore in later chapters. For example, the horizontal lines in Figure 13.2, denoting the boundaries between protocol layers, are actually very complicated objects. They represent limitation as to what each layer can *do* and can *see*. In the next chapter, we will get a glimpse of some methodologies to understand this architectural decision of "who does what" and "how to glue the modules together".

Just between the transport and network layers, there are already quite a few interesting architectural decisions made in TCP/IP, the dominant special case of layers 4/3 protocol. First, the transport layer, in charge of end-to-end management, is *connection-oriented* in TCP, whereas the network layer, in charge of connectivity management, runs hop-by-hop *connectionless* routing in IP. As an analogy, calling someone on the phone requires a connection-oriented session to be established first between the caller and the callee. In contrast, sending mail to someone only needs a connectionless session since the recipient does not need to know there is a session coming in. The design choice of connection-oriented TCP and connectionless IP follows the "end-to-end" principle that end-hosts are intelligent and the network is "dumb." Connectivity establishment should be entirely packet switched in the network layer, and end-to-end feedback run by

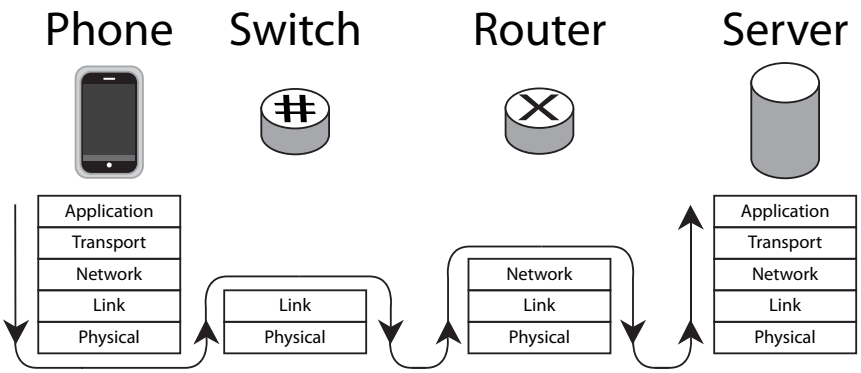


Figure 13.3 Different network elements process up to different layers in the protocol stack. The end hosts process all the way up to the application layer. Switches that forward frames process up to the link layer. Routers that move datagrams across the network process up to the network layer.

the layer above. But this design choice was *not* the only one that the Internet tried over its decades of evolution, *e.g.*, a connectionless transport layer on top of a connection-oriented network layer is also possible and indeed was used.

Second, routing in IP is independent of load condition on the links, whereas congestion control in TCP takes care of managing capacity demand at the end hosts in response to link loads. In addition to the end-to-end principle, this strategy assumes that rate adaptation at the end hosts is easier to stabilize when compared to route adaptation inside the network.

The application layer runs applications that generate a sequence of **messages**. Each of these is divided into **segments** at the transport layer, with a layer 4 **header** added in front of the actual content called **payload**. Then it is passed onto the network layer, which divides and encapsulates the segments as **datagrams/packets**, with a layer 3 header in the front. Each datagram is further passed on to the link layer, which adds another layer 2 header to form a **frame**. This is finally passed on to the physical layer for transmission. These headers are overheads, but they contain useful, sometimes essential, identification and control information. For example, the layer 3 header contains the source node's address and destination node's address, no doubt useful to have in routing. We will examine the impact of these semantic overheads on performance in Chapter 19.

Each network element, *e.g.*, your home gateway, your company's WiFi controller, the central office equipment near the town center, the big router inside the "cloud", runs a subset of the layered protocol stack. And each will decode

and read the header information associated with its subset. This is illustrated in Figure 13.3.

13.1.3 Distributed hierarchy

The Internet is not just complex in terms of the number of tasks it has to manage, but also big in terms of the number of users. While *modularization* helps take care of the complexity by "divide and conquer" in terms of functionalities, *hierarchy* helps take care of the large size by "divide and conquer" in terms of the physical span. This is a recurring theme in many chapters. In the current one, we see that the Internet, this network of networks with more than 30,000 **Autonomous Systems** (ASs), has several main hierarchical levels as illustrated in Figure 13.4.

- A few very large ISPs with global footprints are called **tier-1 ISP**, and they form a *full mesh* **peering** relationship among themselves: each tier-1 ISP has some connection with each of the other tier-1 ISPs. This full mesh network is sometimes called the Internet backbone. Examples of tier-1 ISPs include: AT&T, BT, Level 3, Sprint, etc.
- There are many more tier-2 ISPs with regional footprints. Each tier-1 ISP is connected to some tier-2 ISPs, forming a **customer-provider** relationship. Each of these tier-2 ISPs provides connectivity to many tier-3 ISPs, and this hierarchy continues. The points at which any two ISPs are connected are called the Point of Presence (PoP).
- An ISP of any tier could be providing Internet connectivity directly to consumers. Those ISPs that only take traffic to or from its consumers, but not any transit traffic from other ISPs, are called *stub ISPs*. Typically, campus, corporate, and rural residential ISPs belong to this group.

Another useful concept in distributed hierarchy is that of **domain**. Each business entity forms a domain called AS. There is often a centralized controller within each AS. As we will see later this chapter, routing *within* an AS and routing *across* ASs follow very different approaches.

13.1.4 IP Routing

Packet switching, layered architecture, and distributed hierarchy are three fundamental concepts of the Internet. Now with those topics discussed, we can move on to routing in the Internet.

Transportation networks often offer interesting analogies for communication and social networks. In this case, we can draw a useful analogy from the postal mail service. In order to route a letter from a sender to the receiver, we need three main functionalities:

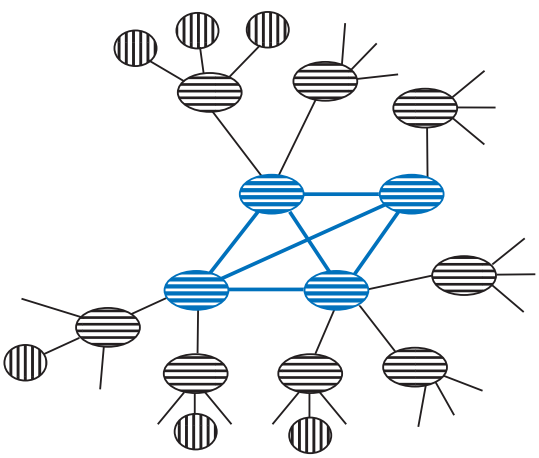


Figure 13.4 Hierarchy in networking: Multiple levels of ISPs and their relationships. Each node in this graph is an ISP, and each link represents a business relationship and physical connection between two ISPs. The four ISPs in the center are tier-1 ISPs, with peering links among themselves. Each of them provides connectivity to many customer ISPs. The stub ISPs at the edge do not provide transit service to any other ISPs. An ISP at any tier may also provide connection to the end users, which are not shown here.

- *Addressing.* We first need to attach a unique label to each node in the network, for otherwise we cannot even identify sources and destinations. In the mail system, the label is the postal address, like street address or mailbox number. Zip codes can quickly zoom you into a subnetwork of the country. In the Internet, we use the **IP address**, a 32 bit number often represented as four decimal numbers separated by dots. Each of these four numbers ranges from 0 to 255 since it is specified by 32/4=8 bits, for example, 127.12.5.88. "Zip codes" here are called subnet masks, for example, 127.12.5.0/24 means that the first 24 bits give the **prefix** of all this subnet's IP addresses: each IP address in this subnet must start with 127.12.5, and can end with any 8 bits. However, in the mail system, an address and a person's ID are separated. In the Internet, an IP address is both an address for establishing connectivity and an identifier of a device. This double loading of functionality onto IP addresses caused various control problems in the Internet.
- *Routing.* Then you have to decide the paths, either one path for each session (single path routing) or multiple paths for each session (multipath routing). Postal mail uses single path routing, and routing decides ahead of time which intermediate cities the mail goes through in order to reach, say, Princeton, NJ from Stanford, CA. There are two broad classes of routing methods: *metric*-based and *policy*-based routing. Inside an AS, routing is based on some kind of metric, either picking the shortest path between the

given source and destination, or distributing the traffic across the paths so that no single path is too loaded. In-between the ASs, however, routing is based on policies. For example, AS 1 might suspect there are hackers connected through AS 2, therefore it avoids routing packets along any path traversing AS 2.

- *Forwarding.* The actual action of forwarding happens each time a packet is received at a router, or each letter is received at an intermediate post office. Some forwarding mechanisms only look at the destination address to decide the next hop, while others read some labels attached to the packet that explicitly indicate the next hop. In any case, a forwarding decision is made and one of the egress links connected to the router is picked to send the packet. Forwarding implements the routing policy.

Let us look at each of the above in a little more detail now, before focusing the rest of the chapter on just the routing portion.

There are two versions of IP: version 4 and version 6. IPv4 uses 32 bits for addresses, which ran out as of early 2011. IPv6 uses four times as many bits, 128 bits, translating into 2^{128}, about 10^{39}, available addresses. That is enough to give, on average, more than a billion of a billion of a billion addresses to each person in the world. That might sound like a lot, but with the proliferation of Internet-connected devices, we are well on our way to using many of these addresses. One way to upgrade an IPv4 network into IPv6 is to create a "tunnel" between two legacy IPv4 network elements, where IPv6 packets are encapsulated in IPv4 headers.

How are these IP addresses allocated? They used to be given out in blocks belonging to different classes. For example, each class A address block has a fixed 8 bit prefix, so $2^{32-8} = 2^{24}$ addresses in a class A block. That is usually given to a national ISP or a large equipment vendor. Lower classes have fewer addresses per block. But this coarse granularity of 8-bit blocks introduced a lot of waste in allocated but unused IP addresses. So the Internet community shifted to Classless Interdomain Routing (**CIDR**), where the granularity does not have to be in multiples of 8 bits.

As a device, you either have a fixed, static IP address assigned to you, or you have to get one dynamically assigned to you by a controller sitting inside the operator of the local network. This controller is called the Dynamic Host Configuration Protocol (**DHCP**) server. A device contacts the DHCP server, receives a currently unused IP address, and returns it back to the IP address pool when no longer needed. You may wonder how a device can communicate with a DHCP server in the first place. We will address the protocols involved in Chapter 19. Sometimes the address given to a device within a local network, *e.g.*, a corporate intranet, is different from the one seen by the outside world, and a Network Address Translation (**NAT**) router translates back and forth.

As mentioned, inter-AS routing is very different from intra-AS routing. Border-Gateway Protocol (**BGP**) is the dominant protocol for inter-AS routing. It

"glues" the Internet together. However, as a policy-based routing protocol, it is a complicated, messy protocol, with many gray areas. We will only briefly describe it in Advanced Material.

Within an AS, there are two main flavors of metric-based routing protocols: Routing Information Protocol (**RIP**) uses the **distance vector** method, where each node collects information about the distances between itself and other nodes, and Open Shortest Path First (**OSPF**) uses the **linked state** method, where each node tries to construct a global view of the entire network topology. We will focus on the simpler RIP in the next section, saving OSPF for Advanced Material.

When a packet arrives at a router, it is put on one of the input ports. On the other end of the router is the set of output ports. In-between is the switching fabric that physically moves the packet from an input port to the right output port. If packets arrive too fast, congestion occurs inside the router. Sometimes it occurs because the intended output ports are occupied, sometimes because the switching fabric is busy, and sometimes because a packet is waiting for its turn at the input port's queue, thus blocking all the packets behind it in the same input queue.

Which output port is the "right" one? That is decided by looking up the forwarding table, either stored centrally in the router, or duplicated with one copy at each input port. The forwarding table connects the routing decisions to actual forwarding actions. A common type of forwarding table lists all the destination IP addresses in the Internet, and indicates which output port, thus the next hop router, a packet should go to based on its destination address written in the header. There are too many IP addresses out there, so the forwarding table often groups many addresses into one equivalent class of input.

We are now going to study one member of the intra-AS routing family, and then how forwarding tables are constructed from distributed messages passing among the routers.

13.2 A Long Answer

Consider a directed graph $G = (V, E)$ representing the topology inside an AS, where each node in the node set V is a router, and each link in the link set E is a physical connection from one router i to another router j. A path is just a set of connected links.

Each link has a cost c_{ij}. It is often a number approximately proportional to the length of the link. If it is 1 for all the links, then minimizing the cost along a path is the same as minimizing the hop count. If it were dynamically reflecting the congestion condition on that link, it would lead to dynamic, load-sensitive routing. IP does not practice dynamic routing, leaving load sensitivity to congestion control to TCP.

The **shortest path problem** is an important special case of the network

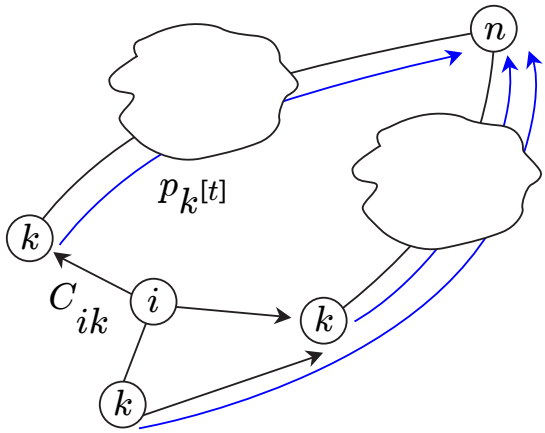


Figure 13.5 Bellman's equation for minimum cost routing. The minimum cost from a node i to destination n is the smallest, among all its neighbors, of the sums of the cost from i to a neighbor and the cost from that neighbor to n. Node i does not need to know how its neighbors get to n, just the cost of reaching n.

flow problem, which is in turn an important special case of linear programming. A more accurate name is the *minimum cost path problem*, since we are actually finding the minimum cost path for any pair of nodes in a given graph. "Minimum cost" would be equivalent to "shortest" only when the link costs are the physical distances. But still, "shortest path problem" sticks, and we will use that term.

The shortest path problem has been studied extensively since the early 1950s, and there are several famous approaches: the Bellman Ford algorithm, the Dijkstra algorithm, Lagrange duality, etc. We will focus on the **Bellman Ford algorithm**, because it is simple, distributed, and illustrates the fundamental principle of **dynamic programming**. It also leads to a fully distributed and asynchronous implementation used in RIP, in part of BGP, and in the routing method in the original ARPANet.

First, a little bit of notation. For now, fix one destination node n; we can generalize to multiple destinations readily. Let $p_i[t]$ be the length of the shortest path from node i to destination n using *at most* t links. It is not a coincidence that we are using the time symbol t to capture this spatial definition. We will soon see that t indeed indexes the iterations too.

If node i cannot reach destination n in t hops, we say $p_i[t] = \infty$. Obviously, at initialization $p_i[0] = \infty$ for all nodes i, unless it is the destination n itself.

Here comes the core idea behind Bellman Ford algorithm. Obviously, i needs to get to n via some neighbor. And we realize that $p_i[t+1]$ can be *decomposed* into two parts, as illustrated in Figure 13.5:

- The cost c_{ik} of getting from node i to one of its outgoing neighbors k. An

outgoing neighbor is a node where there is a link pointing from i to it, and we denote the set of these neighbors for node i as $O(i)$.

- The minimum cost of getting from that neighbor k to the destination n, using at most t hops, since we have already used 1 hop (out of $t+1$ hops) just to get to node k.

The minimum cost path takes the neighbor that minimizes the *sum* of the above two costs:

$$p_i[t+1] = \min_{k \in O(i)} \{c_{ik} + p_k[t]\}. \tag{13.1}$$

Let us assume each node knows the cost to reach to its outgoing neighbors. Then, by iteratively updating $p_i[t]$ and passing a vector describing these updated numbers to its neighbors, we can execute (13.1) in a distributed way. No wonder it is called **distance vector** routing algorithm. Passing the distance vectors around and initializing the c_{ik} values require a little protocol, as we will see soon with RIP.

A quick detour: there is actually a very broad idea behind the recursive **Bellman's equation** (13.1). Optimizing over a sequence of timeslots or spatial points belongs to the research area of **dynamic programming**. For many system models where the cost we want to minimize is *additive* over time or space, and the dependence between stages of the problem is *memoryless* (the next stage only depends on the current one), we know the "tail" of the optimal solution to cost minimization is the optimal solution to the "tail" of the cost minimization. The Bellman Ford algorithm is a special case of this general principle.

13.3 Examples

13.3.1 Centralized Bellman Ford computation

First, an example on the Bellman Ford algorithm. Suppose we have a network topology as in Figure 13.6. The negative link weights are just there to show that the Bellman Ford algorithm can accommodate them, as long as there are no negatively weighted *cycles* since those can reduce some path costs to negative infinity.

In this small example with four nodes (not counting the destination), we know we can stop after 4 iterations, since any path traversing 5 nodes or more will have to go around a cycle, and that can only add to the length, thus never optimal. But in a real, distributed implementation of distance vector protocol, we do not know how many nodes there are, so we have to rely on the lack of new messages to determine when it is safe to terminate the algorithm.

We try to find the minimum path from nodes A,B,C,D to destination n. We initialize distances $p_A[0] = p_B[0] = p_C[0] = p_D[0] = \infty$. And of course it takes zero cost to reach oneself: $p_n[t] = 0$ at all times t.

For $t = 1$, by Bellman's equation, we have

$$\begin{aligned}
p_A[1] &= \min\{c_{AB} + p_B[0], c_{AC} + p_C[0], c_{AD} + p_D[0]\} \\
&= \min\{8 + \infty, 6 + \infty, 4 + \infty\} \\
&= \infty \\
p_B[1] &= \min\{c_{BC} + p_C[0], c_{BD} + p_D[0]\} \\
&= \min\{-3 + \infty, 9 + \infty\} \\
&= \infty \\
p_C[1] &= \min\{c_{Cn} + p_n[0], c_{CD} + p_D[0], c_{CA} + p_A[0]\} \\
&= \min\{6 + 0, -5 + \infty, -2 + \infty\} \\
&= 6 \\
p_D[1] &= \min\{c_{Dn} + p_n[0]\} \\
&= 7 + 0 \\
&= 7
\end{aligned}$$

Notice that node D has only one outgoing link, so $p_D = 7$ and we do not need to continue calculating it.

Similarly, for $t = 2$, we have

$$\begin{aligned}
p_A[2] &= \min\{c_{AB} + p_B[1], c_{AC} + p_C[1], c_{AD} + p_D[1]\} \\
&= \min\{8 + \infty, 6 + 6, 4 + 7\} \\
&= 11 \\
p_B[2] &= \min\{c_{BC} + p_C[1], c_{BD} + p_D[1]\} \\
&= \min\{-3 + 6, 9 + 7\} \\
&= 3 \\
p_C[2] &= \min\{c_{Cn} + p_n[1], c_{CD} + p_D[1], c_{CA} + p_A[1]\} \\
&= \min\{6 + 0, -5 + 7, -2 + \infty\} \\
&= 2
\end{aligned}$$

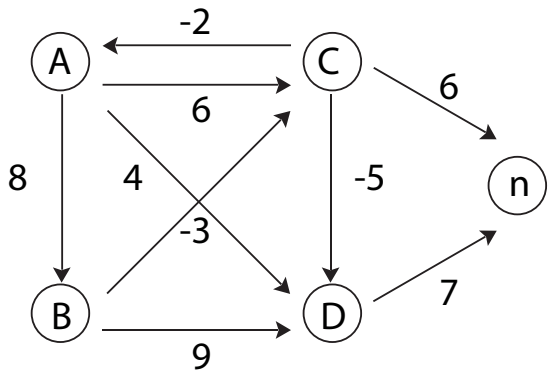


Figure 13.6 An example to illustrate the Bellman Ford algorithm. Here we want to find out the shortest path from nodes A, B, C, and D to a common destination node n. There are negatively-weighted links just to illustrate that the algorithm can handle them. But there are no negatively-weighted cycles, for they would make the problem ill-defined.

For $t = 3$:

$$\begin{aligned} p_A[3] &= \min\{c_{AB} + p_B[2], c_{AC} + p_C[2], c_{AD} + p_D[2]\} \\ &= \min\{8 + 3, 6 + 2, 4 + 7\} \\ &= 8 \end{aligned}$$

$$\begin{aligned} p_B[3] &= \min\{c_{BC} + p_C[2], c_{BD} + p_D[2]\} \\ &= \min\{-3 + 2, 9 + 7\} \\ &= -1 \end{aligned}$$

$$\begin{aligned} p_C[3] &= \min\{c_{Cn} + p_n[2], c_{CD} + p_D[2], c_{CA} + p_A[2]\} \\ &= \min\{6 + 0, -5 + 7, -2 + 11\} \\ &= 2 \end{aligned}$$

For $t = 4$:

$$\begin{aligned} p_A[4] &= \min\{c_{AB} + p_B[2], c_{AC} + p_C[3], c_{AD} + p_D[3]\} \\ &= \min\{8 - 1, 6 + 2, 4 + 7\} \\ &= 7 \end{aligned}$$

$$\begin{aligned} p_B[4] &= \min\{c_{BC} + p_C[3], c_{BD} + p_D[3]\} \\ &= \min\{-3 + 2, 9 + 7\} \\ &= -1 \end{aligned}$$

$$\begin{aligned} p_C[4] &= \min\{c_{Cn} + p_n[3], c_{CD} + p_D[3], c_{CA} + p_A[3]\} \\ &= \min\{6 + 0, -5 + 7, -2 + 8\} \\ &= 2 \end{aligned}$$

We can also readily keep track of the paths taken by each node to reach n: D directly goes to n, C goes through D to reach n, B goes through C, and A goes through B.

13.3.2 Distributed RIP

So far we have assumed centralized computation. But imagine you are one of the nodes trying to figure out how to reach the other nodes in the network. How do you know the cost to reach different nodes and how do you even start?

We now describe the message passing protocol in RIP, which allows the discovery and update of c_{ik} and $p_i[t]$ across the nodes. For simplicity of presentation, assume all the links are bi-directional: if node i can send messages to node j, so can j to i.

The message passed around in distance vector routing protocols has the following format: `[NodeID, DestinationID, Cost of MinCost Path]`.

At the very beginning, iteration 0, each node only knows its own existence, so each node i can only pass around this vector `[node i, node i, 0]`. But once each node receives the messages from its neighbors, it can update its list of vectors.

There are several key features of the message passing:

- *Short messages*: All the detailed topology information about who connects to whom and what are the link costs can all be *summarized* into these lists of distance vectors.
- *Local interaction*: Neighbor-to-neighbor message passing is enough to propagate this summary so that the shortest paths can be discovered. No need for broadcasting all the summaries to all the nodes.
- *Local view*: Even when these optimal paths are obtained, still each node has only a local view: it knows only which neighbor to send a packet with a given destination address, but has no idea what the actual end-to-end path looks like. As far as forwarding packets based on its destination IP address

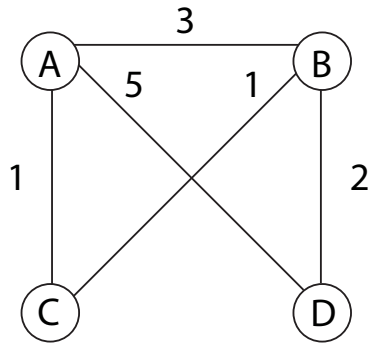


Figure 13.7 An example to illustrate the distributed message passing in RIP, where each node wants to find one of the shortest paths to every other node through message passing. Again, we choose an extremely small network so that we can go through the numerical steps in detail. A major challenge in routing in the Internet is the scale of the network, hence the desire for distributed solutions in the first place.

is concerned, it does not need to because the routers will forward packets hop by hop.

Here is an example to illustrate this iterative and distributed routing method, for the small network shown in Figure 13.7.

We try to find the minimum paths between all nodes. We can collect all the distance vectors `[NodeID, DestinationID, Cost of MinCost Path]`, together with the next hop decision, into a routing table for each node. Each node only stores the table where it is the NodeID.

At $t = 0$, we have the following 4 tables, one per node:

NodeID	DestinationID	Cost of MinCost Path	Next node
A	A	0	A

NodeID	DestinationID	Cost of MinCost Path	Next node
B	B	0	B

NodeID	DestinationID	Cost of MinCost Path	Next node
C	C	0	C

NodeID	DestinationID	Cost of MinCost Path	Next node
D	D	0	D

At each iteration, each node sends the distance vectors (the routing table above except the next hop information) to its neighbors. For example, at $t = 1$, node B receives messages from nodes A,C,D. Node B receives [A,A,0] from node A; [C,C,0] from node C; and [D,D,0] from node D.

In terms of Bellman's equation (13.1), this tells node B that $p_A[0] = 0$ for destination A, $p_C[0] = 0$ for destination C, and $p_D[0] = 0$ for destination C. All other distances are infinite. Node B then uses (13.1) to calculate the new distances in the routing table.

Let us work out the calculations for destination A at $t = 1$.

$$
\begin{aligned}
p_A[1] &= 0 \\
p_B[1] &= \min\{c_{BA} + p_A[0], c_{BC} + p_C[0], c_{BD} + p_D[0]\} \\
&= \min\{3 + 0, 1 + \infty, 2 + \infty\} \\
&= 3 \\
p_C[1] &= \min\{c_{CA} + p_A[0], c_{CB} + p_B[0]\} \\
&= \min\{1 + 0, 1 + \infty\} \\
&= 1 \\
p_D[1] &= \min\{c_{DA} + p_A[0], C_{DB} + p_B[0]\} \\
&= \min\{5 + 0, 2 + \infty\} \\
&= 5.
\end{aligned}
$$

For destination B at $t = 1$, we have:

$$
\begin{aligned}
p_A[1] &= \min\{c_{AB} + p_B[0], c_{AC} + p_C[0], c_{AD} + p_D[0]\} \\
&= \min\{3 + 0, 1 + \infty, 5 + \infty\} \\
&= 3 \\
p_B[1] &= 0 \\
p_C[1] &= \min\{c_{CA} + p_A[0], c_{CB} + p_B[0]\} \\
&= \min\{1 + \infty, 1 + 0\} \\
&= 1 \\
p_D[1] &= \min\{c_{DA} + p_A[0], c_{DB} + p_B[0]\} \\
&= \min\{5 + \infty, 2 + 0\} \\
&= 2.
\end{aligned}
$$

For destination C at $t = 1$, we have:

$$\begin{aligned} p_A[1] &= \min\{c_{AB} + p_B[0], c_{AC} + p_C[0], c_{AD} + p_D[0]\} \\ &= \min\{3 + \infty, 1 + 0, 5 + \infty\} \\ &= 1 \\ p_B[1] &= \min\{c_{BA} + p_A[0], c_{BC} + p_C[0], c_{BD} + p_D[0]\} \\ &= \min\{3 + \infty, 1 + 0, 2 + \infty\} \\ &= 1 \\ p_C[1] &= 0. \end{aligned}$$

There is no $P_D[1]$ since node C does not even realize the existence of node D yet.

For destination D at $t = 1$, we have:

$$\begin{aligned} p_A[1] &= \min\{c_{AB} + p_B[0], c_{AC} + p_C[0], c_{AD} + p_D[0]\} \\ &= \min\{3 + \infty, 1 + \infty, 5 + 0\} \\ &= 5 \\ p_B[1] &= \min\{c_{BA} + p_A[0], c_{BC} + p_C[0], c_{BD} + p_D[0]\} \\ &= \min\{3 + \infty, 1 + \infty, 2 + 0\} \\ &= 2 \\ p_D[1] &= 0. \end{aligned}$$

Although each set of calculations organized above is for various nodes to a single destination, during the execution of RIP, it is the other way around: each node performs calculations for various destinations. At $t = 1$, each node stores its own table shown below. C and D cannot reach each other because so far the message passing only reveals one hop paths and these two nodes require at least 2 hops to connect.

NodeID	DestinationID	Cost of MinCost Path	Next node
A	A	0	A
A	B	3	B
A	C	1	C
A	D	5	D

NodeID	DestinationID	Cost of MinCost Path	Next node
B	A	3	A
B	B	0	B
B	C	1	C
B	D	2	D

NodeID	DestinationID	Cost of MinCost Path	Next node
C	A	1	A
C	B	1	B
C	C	0	C

NodeID	DestinationID	Cost of MinCost Path	Next node
D	A	5	A
D	B	2	B
D	D	0	D

Now at $t = 2$, all nodes send updated distance vectors to their neighbors. For example,

- Node B receives [A,B,3] [A,C,1] [A,D,5] from node A. This tells node B that $p_A[1] = 3$ for destination B, $p_A[1] = 1$ for destination C, and $p_A[1] = \infty$ for destination D.
- Node B receives [C,A,1] [C,B,1] from node C. This tells node B that $p_C[1] = 1$ for destination A, $p_C[1] = 1$ for destination B, and $p_C[1] = \infty$ for destination D.
- Node B receives [D,A,5] [D,B,2] from node D. This tells node B that $p_D[1] = 5$ for destination A, $p_D[1] = 2$ for destination B, and $p_D[1] = \infty$ for destination C.

We update all routing tables at $t = 2$. We focus on node A's table to illustrate the derivation. For destination B at $t = 2$, we have:

$$\begin{aligned} p_B[2] &= \min\{c_{AB} + p_B[1], c_{AC} + p_C[1], c_{AD} + p_D[1]\} \\ &= \min\{3 + 0, 1 + 1, 5 + 2\} \\ &= 2 \end{aligned}$$

For destination C at $t = 2$, we have:

$$\begin{aligned} p_C[2] &= \min\{c_{AB} + p_B[1], c_{AC} + p_C[1], c_{AD} + p_D[1]\} \\ &= \min\{3 + 1, 1 + 0, 5 + \infty\} \\ &= 1 \end{aligned}$$

For destination D at $t = 2$, we have:

$$\begin{aligned} p_D[2] &= \min\{c_{AB} + p_B[1], c_{AC} + p_C[1], c_{AD} + p_D[1]\} \\ &= \min\{3 + 2, 1 + \infty, 5 + 0\} \\ &= 5. \end{aligned}$$

Each node stores its own routing information in a table at $t = 2$. The above explains the entries in the first table below, and we also see nodes C and D know the existence of each other now:

NodeID	DestinationID	Cost of MinCost Path	Next node
A	A	0	A
A	B	2	C
A	C	1	C
A	D	5	B

NodeID	DestinationID	Cost of MinCost Path	Next node
B	A	2	C
B	B	0	B
B	C	1	C
B	D	2	D

NodeID	DestinationID	Cost of MinCost Path	Next node
C	A	1	A
C	B	1	B
C	C	0	C
C	D	3	B

NodeID	DestinationID	Cost of MinCost Path	Next node
D	A	5	A
D	B	2	B
D	C	3	B
D	D	0	D

Similarly, each node stores its own routing information in a table at $t = 3$:

NodeID	DestinationID	Cost of MinCost Path	Next node
A	A	0	A
A	B	2	C
A	C	1	C
A	D	4	C

NodeID	DestinationID	Cost of MinCost Path	Next node
B	A	2	C
B	B	0	B
B	C	1	C
B	D	2	D

NodeID	DestinationID	Cost of MinCost Path	Next node
C	A	1	A
C	B	1	B
C	C	0	C
C	D	3	B

NodeID	DestinationID	Cost of MinCost Path	Next node
D	A	4	B
D	B	2	B
D	C	3	B
D	D	0	D

Further iterations produce no more changes. The routing tables have converged to the right solution through distributed message passing in RIP.

What about link failures? Is distance vector routing robust to events like a link breaking? We will find out in a homework problem.

13.4 Advanced Material

In this section, we will go into further detail of routing within an AS and across ASs. In a homework problem, we will also go through the essential ideas in switching within a small, local area network and a distributed protocol that determines a spanning tree to connect a given set of nodes.

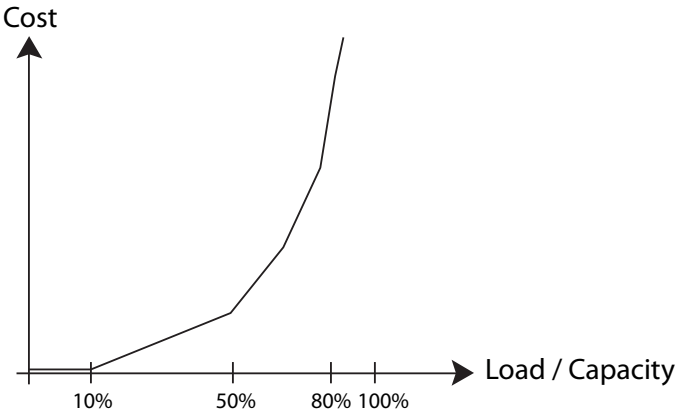


Figure 13.8 A typical cost function $C(f)$: increasing, convex, and piecewise-linear. It rises sharply as f becomes about 50% of the link capacity, and reaches very highly values by the time f is 80% of the link capacity.

13.4.1 Link state routing: OSPF

Picking the shortest paths is just one metric out of several reasonable ones. Another popular metric is the minimization of link loads, where the load of a link is defined as the percentage of its capacity used. This is carried out through a procedure called **traffic engineering** by the ISPs. The goal is to load-balance the traffic across the paths so that all the link loads are as low as possible under the traffic demand, or at least no one link load becomes too high and forms a bottleneck.

The benchmark of traffic engineering performance is defined by the **multi-commodity flow problem**. It is a basic optimization problem encountered in many networks to design a mapping of flows onto different paths in a network with a given topology $G = (V, E)$.

We assume that each destination n has one flow coming to it from various other nodes in the network. Let f_{ij}^n denote the amount of flow on link (i, j) destined to node n, and f_{ij} the sum of load on link (i, j) across all destinations. The objective function is to minimize some cost function: the higher the load, the higher the cost. A typical cost function as a piecewise linear, increasing, and convex function is shown in Figure 13.8. As the load approaches the link capacity, the cost rises sharply. The constraint is simply a flow conservation equality: the incoming traffic to a node v, plus $D(v, n)$, the traffic that originates at v and destined to n, must be equal to the outgoing traffic from node v.

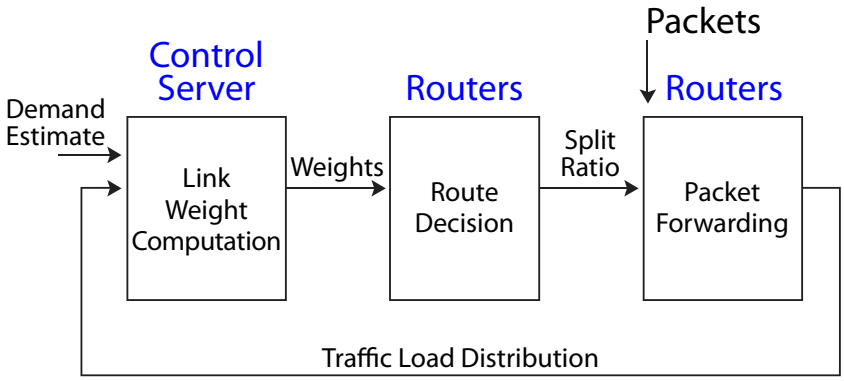


Figure 13.9 Three main modules in link state routing protocols like OSPF. Link weight computation is done in a central management server based on an estimation of traffic demand. These link weights are then given to the routers, each autonomously deciding the split ratio for load balancing based on these weights. When a packet actually arrives at a router, forwarding to the next hop is done based on the reading the destination IP address.

$$\begin{array}{ll} \text{minimize} & \sum_{(i,j)} C(f_{ij}) \\ \text{subject to} & \sum_{j:(v,j)\in E} f^n_{v,j} = \sum_{i:(i,v)\in E} f^n_{i,v} + D(v,n), \quad \text{for all } v \neq n \\ & f_{ij} = \sum_t f^t_{i,j}, \quad \text{for all } (i,j) \end{array} \tag{13.2}$$

Using the terminology from Chapter 4, we know this problem (13.2) is a linearly constrained, convex optimization problem. So it is easy to solve, at least through a centralized algorithm. And there are many specialized algorithms to further speed up the computation.

But IP does *not* allow end-to-end tunneling, so it cannot keep track of the f^n_{ij}. This is a design choice driven by the simplicity of network management. IP also does not allow dynamic routing adaptive to link loads, that is the job of TCP. What actually happens is that the most popular intra-AS routing protocol, OSPF, solves the above problem *indirectly* through link weight optimization. This is illustrated in Figure 13.9 and summarized below.

- *Link weight computation.* A centralized management server collects or estimates the source-destination traffic, *i.e.*, all the $D(v,n)$ in (13.2) every 12 hours or so. This timescale is not dynamic at the same timescale as traffic fluctuation. Then the server computes a set of link weights, one for each link in the network, and uploads this information to all the routers. Each router has a global view of the topology, not just a neighborhood local view as in distance vector routing.

- *Use link weights to split traffic.* Given the weights computed by the centralized management server, each router constructs many paths to each of the possible destinations. In OSPF, each router constructs just the shortest paths, under the given link weights, and splits the incoming traffic equally among all the shortest paths.
- *Forward the packets.* As each packet arrives at the router, the next hop is decided based just on the destination IP address, and the traffic splitting decided in the step above. It does not matter what is the source and what routers the packet has traversed so far. This is called *destination-based and hop-by-hop forwarding.*

Comparing link state routing like OSPF with distance vector routing like RIP, we see that link state routing passes detailed messages about each local topology, while distance vector routing passes coarser messages about the global topology. There are also different tradeoffs among communication overhead and local computation between link state and distance vector protocols.

Is it easy to turn the knob of link weight, and hope the right weights will indirectly induce a traffic distribution $\{f_{ij}^t\}$ solving (13.2)? The answer is no. Picking the link weights for OSPF in order to induce a solution to (13.2) is an NP-hard problem.

But you do *not* have to use OSPF. There are other members of this family of link state routing, *e.g.*, PEFT, where link weights are used to define the weight for all the paths to be used: not just the shortest paths, but also the longer paths which are used exponentially less. If link weights are used in PEFT fashion, it turns out computing the right link weights becomes a computationally tractable problem. Similar to mechanism design, this is a case in the principle of "design for optimizability": designing the network protocol so that its operation can be readily optimized.

13.4.2 Inter-AS routing: BGP

Since the Internet is a network of networks, we need all ASs to cooperate with each other so that one AS's customers can reach customers of another AS. Inter-AS routing glues the entire Internet together, but it is messy, because the best path (defined by some distance or weight metric) across the ASs is often *not* chosen due to policy and economic concerns.

Consider a small scale inter-AS connectivity graph in Figure 13.10. Each AS has a number called ASN, just like each host and each interface of a router has an IP address.

BGP governs the routing across the ASs (eBGP session), and moves inter-AS packets within an AS (iBGP session). Picking the next hop AS in BGP is *almost* like picking the next hop router in RIP, as each AS passes a list of *BGP attributes* to neighbor ASs. Part of the attributes is called AS-PATH, listing the ASs that this AS needs to pass in order to reach a certain range of IP addresses.

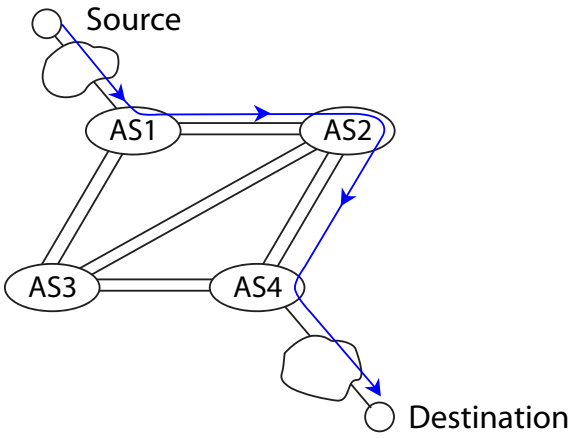


Figure 13.10 An example of BGP routing a session across multiple ASs in the same tier. Each node is an AS, and each link is a peering relationship manifested through some physical connectivity between border routers. Which AS to pick as the next hop depends on not just performance metrics but also policies based on economic and security concerns.

Here comes the messy but critical part in eBGP: each AS can have a sequence of *filters* that reprioritize the choices of neighboring ASs to go to in order to reach a certain range of IP addresses. On top of that list of filters is a *local preference policy*: AS 1 might decide that it simply does not want to go through AS 3, so any AS-PATH containing AS 3 is put to the bottom of the priority list. This decision might be due to security or economic concern: AS 3 may be perceived by AS 1 as non-secure, or AS 1 may not want to send too much traffic along to AS 3 in case it tips the traffic volume balance between them and results in a new payment contract between these two peering ASs.

Usually there are multiple border routers connecting two ASs. Which one should we use? iBGP uses a simple rule: pick the border router that has the minimum cost to reach (from where the packet is positioned when it enters the AS). This means each AS wants to get rid of the packet as soon as possible. This is called *hot potato* routing.

There is a lot of detail of BGP beyond the above gist, and there has been an interesting model called the *stable path problem* that crystallizes the BGP stability issues.

We conclude this chapter on the fundamentals of the Internet by mentioning that there are actually many other types of routing in communication networks. There are specialized routing protocols in wireless mesh networks and optical networks. There is also much work on routing with a guarantee on quality of service such as delivery time. Some of these routing protocols are centralized whereas others are distributed. Some have centralized control planes for param-

eter optimization, yet distributed data planes that forward the actual packets. Some of these protocols are static, whereas others are dynamic as a function of the link loads on a fast timescale.

And there have been years of work in both research community and industry in implementing **multicast routing**. We have only been looking at unicast routing: from one sender to one receiver. As the Internet is used increasingly for content distribution and video entertainment delivery, often there are many receivers at the same time. Creating a unicast routing session for each of the receivers is often inefficient. But multicasting at the IP layer turns out to be hard to manage, and has often been replaced by an *architectural alternative*: application layer multicasting in an overlay network. We will pick this up in Chapter 15 on P2P.

Further Reading

There is a whole library of computer networking and Internet textbooks, and thousands of papers on all kinds of aspects of routing.

1. The following book provides a highly readable and concise overview on Internet routing protocols:

[Hui99] C. Huitema, *Routing in the Internet*, 2nd Ed., Prentice Hall, 1999.

2. One of the standard textbooks on computer networking is

[PD12] L. L. Peterson and B. Davie, *Computer Networks: A Systems Approach*, Morgan Kauffman, 2012.

3. A recent research article on link weight optimization in link state rouing is

[XCR12] D. Xu, M. Chiang, and J. Rexford, "Link state routing protocol can achieve optimal traffic engineering," *IEEE/ACM Transactions on Networking*, vol. 19, no. 6, pp. 1717-1730, November 2011.

4. We did not have the time to cover BGP in detail. The following is a relatively recent survey article on BGP:

[CR05] M. Caesar and J. Rexford, "BGP routing policies in ISP networks," *IEEE Network Magazine*, 2005.

5. The following book provides a comprehensive survey of layer 2 switching and layer 3 routing protocols. And it is a rare example of a book that is devoted to network protocols and yet can maintain a sense of humor; see Chapter 18 for example.

[Per99] R. Perlman, *Interconnections: Bridges, routers, switches, and internetworking protocols*, 2nd Ed., Addison-Wesley, 1999.

Problems

13.1 *Packet switching* ⋆

(a) *Statistical multiplexing*

Suppose you have a 10Mbps link shared by many users. Each user of the link generates 1Mbps of data 10% of the time, and is idle 90% of the time.

If we use a circuit switched network, and the bandwidth allocation is equal, how many users can the link support? Call this number N. Now consider a packet switched network. Say we have M users in total, and we want the probability of a user being denied service to be less than 1%. Write down the expression that must be solved in the form of $f(M, N) < 0.01$. Solve this numerically for M.
(Hint: Use the binomial CDF.)

(b) *Resource pooling*

We will consider modeling a shared resource and see what happens when both the demand for the resource and ability to fulfill requests increases. Suppose we have m servers. When a request comes in, a server answers the request. If all servers are busy, the request is dropped. The following **Erlang formula** gives the probability of a request being denied, given m servers and E units of traffic:

$$P(E, m) = \frac{\frac{E^m}{m!}}{\sum_{i=0}^{m} \frac{E^i}{i!}}.$$

Calculate $P(3, 2)$. Now calculate $P(6, 4)$. What do you observe? In general, $P(wx, wy) < P(x, y), \quad \forall w > 1$. This is one way to quantify the notion of resource pooling's benefits.

13.2 *RIP* ⋆⋆

Consider the network shown in Figure 13.2.

(a) Run an example of RIP on this network to find the minimum paths between all nodes. Show the routing tables at each time step.

(b) Now the link between A and C fails, resulting in a cost of ∞ for both directions of transmission. B and C immediately detect the link failure and update their own routing tables based on the information they already have from their one-hop neighbors. Write down the routing tables for four iterations after the link failure. You only need to show the routing tables that change. What is happening to the paths to A?

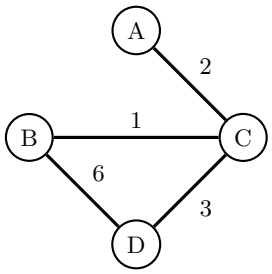


Figure 13.11 A network for a RIP example.

(c) Propose a solution to the problem found in (b).

13.3 *Ford-Fulkerson algorithm and the max flow problem* ⋆⋆⋆

You are in the Engineering library studying for your final exam, which will take place in 1 hour. You suddenly realize you did not attend a key lecture on power control. Your kind TA offers to send you a video of the lecture. Unfortunately, she lives off in the Graduate College, which is somewhere way off-campus (you do not even know where).

Since you want to get the video as quickly as possible, you decide to split it into many pieces before sending it over the Princeton network. The Princeton network has pipe capacities given in Figure 13.12. How much bandwidth should you send over each pipe so that you maximize your total received bandwidth?

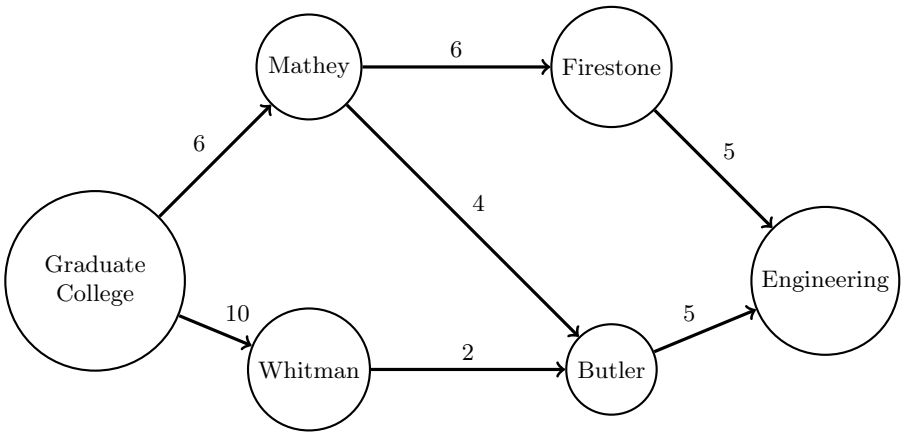


Figure 13.12 A simplified Princeton campus network, with the six nodes abbreviated as G, M, B, W, F, and E.

We will walk through a step-by-step approach for solving this *maximum flow problem*. A useful operation will be generating a *residual graph*, given a flow. For

each link on the given flow's path, we draw a *backward link* with the amount of flow, leaving a forward link with the remaining capacity of the link. An example is shown in Figure 13.3.

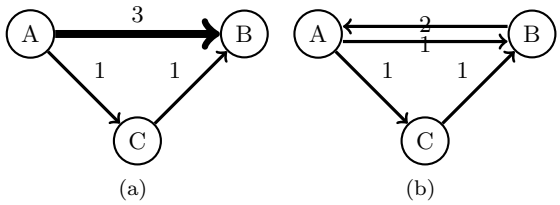


Figure 13.13 Example of drawing the residual graph. We decide to push 2 units of flow from A to B in the left graph, which gives us the residual graph on the right.

(a) Allocate 4 Mbps to the path G-M-B-E. Draw the residual graph.

(b) Allocate 2 Mbps to the path G-W-B-M-F-E on the residual graph from (a). Draw the new residual graph.

(c) Allocate 2 Mbps to the path G-M-F-E on the residual graph from (b). Draw the new residual graph.

(d) There are no paths remaining on the residual graph from (c), so the algorithm terminates. The bandwidth allocation is given by the net bandwidth from steps (a),(b),(c). Draw the graph with the final capacity allocation on each of the links.

This classic algorithm is called the **Ford-Fulkerson** algorithm.

13.4 *Dynamic alternative routing* ⋆⋆

We did not get a chance to talk about routing in circuit switched networks. A major brand there is **DAR**: Dynamic Alternative Routing, which was adopted by British Telecom in 1996 for their networks.

Suppose that, instead of having fixed paths to route packet from source to destination, we want to have the routes change dynamically in response to network conditions. One simple and effective scheme is **dynamic alternative routing**. We will consider this protocol in the case of a fully-connected graph.

We want the routing to adapt dynamically to the link utilization and select a new path if the current one is too busy. Each possible session (source-destination pair) has an associated backup node. When a session is initiated, it first tries to send its traffic along the direct link. If the direct link fails because it is full, the session tries to use the 2-hop path with the backup node. If the backup path fails too because it is busy, the session fails and selects a new backup node

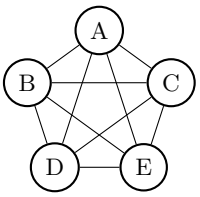


Figure 13.14 An example topology to illustrate DAR.

from the network. Clearly, the backup node should not be the same node as the destination of the session.

One possible problem with this scheme is if many sessions end up using 2-hop paths. Then we are not being very efficient, since we are using double the capacity compared to a 1-hop path. We would like 1-hop sessions to be serviced. Therefore, dynamic alternative routing reserves a fraction t_l of each link l for direct sessions (sessions that use the direct link between the source and destination, as opposed to the backup path). We call this parameter $0 < t_l < 1$ the **trunk reservation coefficient**. Each link l in the network has an overall capacity of c_l Mbps and is bidirectional. The non-reserved capacity, $c_l - t_l c_l$, may be used for either direct or indirect sessions.

(a) Suppose we have the network shown in Figure 13.14. Let $c_l = 10$ Mbps, $t_l = 0.1$, $\forall l$. The backup nodes are initialized as in Table 13.1.

Session	Backup node
(B,C)	A
(C,B)	A
(A,C)	E

Table 13.1 Backup nodes for a DARP example.

Link	Unreserved capacity used [session]	Reserved capacity used [session]
(A,B)		
(A,C)		
(A,E)		
(B,C)	9 Mpbs [9 × (B,C)]	1 [1 × (B,C)]
(C,E)		

Table 13.2 The table of link utilizations to be filled out in the DAR example.

The following events occur in sequence.
1. 10 parallel sessions of (B,C) begin.
2. 10 parallel sessions of (C,B) begin.
3. 10 parallel sessions of (A,C) begin.

Assume the sessions last a long time and each session consumes 1 Mbps. Fill in Table 13.2 after the above sequence of events has occurred. Remember that sessions and links are different. One row has been filled out for you as an example.

(b) Repeat (a) without the trunk reservation scheme.

(c) What is efficiency of link utilization, $\frac{\text{number of sessions}}{\text{total network capacity used}}$, under (a) and (b)?

(For more details, see R. J. Gibbens, F. P. Kelly, and P. B. Key, "Dynamic alternative routing," *Routing in Communication Networks*, Prentice Hall, 1995.)

13.5 *Spanning tree* ⋆⋆⋆

Routing in an interconnected set of **local area networks** (LANs), like the one in a corporation or campus, is much easier than over the entire Internet globally connecting many different layers 1 and 2 technologies. The connections among LANs are called **bridges**, or **switches**. Each with multiple ports, with one port connecting to one LAN. We could configure the LAN IDs on each port of a bridge, but a more scalable and automated way is for each bridge to listen to each packet that arrives on a port, and copy the source address in that packet's header to a database of hosts reachable from that port.

This learning bridge protocol works well, except when there are loops in the graph of LANs and bridges. So we need to build a tree that connects all the LANs without forming any cycles, the so-called **spanning tree**. Then there is only one way to forward a packet from one device to another (unless you traverse the same link multiple times). We will later see building multiple trees for multicast overlay in P2P networks in Chapter 15.

If each link has a weight, say, the distance of the link, then finding the smallest weight spanning tree is the well-studied graph-theoretic problem of **minimum spanning tree**. If you can add extra intermediate nodes to shorten the total distance in the tree, it is called the **Steiner tree** problem. It turns out that some neurological networks in animals follow the principle of Steiner tree construction.

In this homework problem, we tackle a simpler and still important problem of *distributedly* discovering a spanning tree. This protocol was invented by Perlman in 1985. Like link state routing protocols, the spanning tree protocol we see below is an example of achieving a globally consistent view of the topology through only local interactions that eventually propagate throughout the network.

Consider a set of local area segments (each with some devices attached to it) and bridges depicted in Figure 13.15. Clearly there are cycles in the given graph:

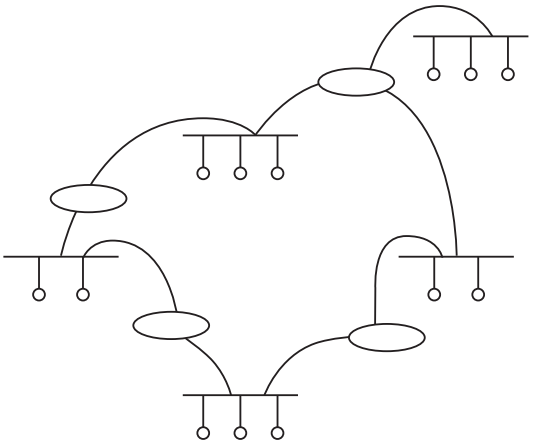


Figure 13.15 An example of local area networks connected by bridges. There are two types of nodes in this graph: each local area network (a line with its hosts represented as small circles) is a node, and each bridge (an oval) is also a node. The links in this graph connect bridges with local area networks. If all the briges are used, the graph becomes cyclic. Distributed spanning tree protocols discover an acyclic subgraph that includes all the local area networks but not all the links.

we can go from one segment through other segments and back to itself without going through a link twice. We want to determine a cycle-free way to provide connectivity among all the segments. One way to arrive at a consistent spanning tree is to have the bridge with the smallest ID number as the root of the tree, and each of the other bridges reach this root bridge through the smallest-hop-count path. That is easy, at least for such small networks, if we have a global view. But how to do that distributedly, with message passing only between neighbors? How do the nodes even agree on which bridge is the root of the tree?

One possibility is to ask each bridge to announce the following message, consisting of three fields, during each time slot:

- ID of the bridge believed to be the root ID.
- Number of hops to reach that root bridge from this bridge.
- ID of this bridge.

Now, here come the questions:

(a) Initially, each bridge only has local information about itself. What are the messages from the bridges in Figure 13.15?

(b) Upon receiving a message, each bridge selects the root bridge and discover the way to reach it based on the following ordered list of criteria:

1. Lower ID number of a bridge wins and becomes he new root bridge as believed by this bridge.

2. If there are multiple paths to reach the same root bridge, the path with the smallest hop count wins.
3. If there are multiple equal-hop-count paths to reach the same root bridge, the path sent to this bridge from a bridge with a smaller ID number wins.

Each bridge then updates the root bridge field of the message following rule number 1 above, and increase the hop count by 1. Then it sends the new message to its neighbors, except of course those neighbors that have a shorter path toward the same root bridge.

Write down the evolution of the messages for the bridges in Figure 13.15. Does it converge to a spanning tree?

(c) Even after convergence, the root bridge keeps sending the message once every regular period. Why is that? Consider what happens when a bridge fails.

14 Why doesn't the Internet collapse under congestion?

14.1 A Short Answer

14.2 Principles of distributed congestion control

When demand exceeds supply, we have congestion. If the supply is fixed, we must reduce demand to alleviate congestion. When demand comes from different nodes in a network, we need to coordinate it in a distributed way. As the demand for capacity in the Internet exceeds the supply every now and then, **congestion control** becomes essential.

This was realized in October 1986, when the Internet had its first congestion collapse. It took place over a short, 3-hop connection between Lawrence Berkeley Lab and UC Berkeley. The normal throughput was 32 kbps (that is right, kbps, not the Mbps numbers we hear these days). That kind of dial-up modem speed was low enough, but during the congestion event, it dropped all the way down to 40 bps, by almost a factor of 1000.

The main reason was clear as we saw from the last chapter on routing: when users send so many bits per second that their collective load on a link exceeds the capacity of that link, these packets are stored in a buffer and they wait in the queue to be transmitted. But when that waiting becomes too long, more incoming packets accumulate in the buffer until the buffer overflows and packets get dropped. This is illustrated in Figure 14.1.

These dropped packets never reach the destination, so the intended receiver never sends an acknowledgement (an ACK packet) back to the sender, as it should do in the **connection-oriented**, end-to-end control in TCP. Internet design evolution considered different divisions of labor between layers 3 and layer 4, eventually settling on a connection-oriented layer 4 and connectionless layer 3 as the standard configuration. According to TCP, the sender needs to resend the unacknowledged packets. This leads to a vicious cycle, a *positive feedback loop* that feeds on itself: as the same set of senders that caused congestion in the first place keeps resending the dropped packets, congestion persists. Packets keep getting dropped at the congested link, resent from the source, dropped at the congestion link... Senders need to rethink how they can avoid congestion in the first place, and they need to back off when congestion happens. We need to turn the positive feedback loop into a *negative* feedback loop.

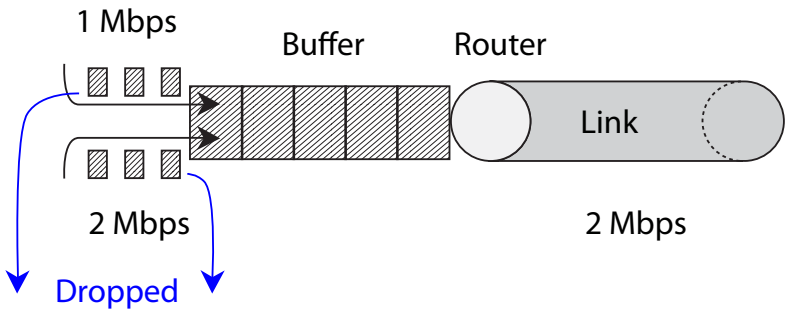


Figure 14.1 An illustration of congestion at one end of a link. Two sessions arrive at the buffer with an aggregate demand of 3 Mbps, but there is only a supply of 2 Mbps in the outgoing link. The buffer is filled up and packets start to get dropped. Which packets get dropped depends on the details of the queue management protocols.

That was what Van Jacobson proposed in the first congestion control mechanism in 1988, called **TCP Tahoe**. It has been studied extensively since then, and improved significantly several times. But most of the essential ideas in congestion control for the Internet were in TCP Tahoe already:

- *End-to-end control via negative feedback.* We can imagine congestion control within the network where, hop by hop, routers decide for the end hosts at what rates they should send the packets. That is actually what another protocol, called Asynchronous Transmission Mode (**ATM**), does to one type of its traffic, the Arbitrary Bit Rate traffic. But TCP congestion control adopts the alternative approach of *intelligent edge network* and *dumb core network*. The rate at which a sender sends packets is decided by the sender itself. But the network provides hints through some feedback information to the senders. Such feedback information can be inferred from the *presence* and *timing* of acknowledgement packets, transmitted from the receiver back to the sender acknowledging the in-order receipt of each packet.
- *Sliding-window-based control.* If a sender must wait for the acknowledgement of a sent packet before it is allowed to send another packet, it can be quite slow. So we pipeline by providing a bigger allowance. Each sender maintains a sliding window called the **congestion window** (`cwnd`). If the window size is 5, that means up to 5 packets can be sent before the sender has to pause, and wait for acknowledgement packets to come back from the receiver. For each new acknowledgement packet received by the sender, the window is slid one packet forward and enables the sending of a new packet, hence the name "sliding window". This way of implementing a restriction

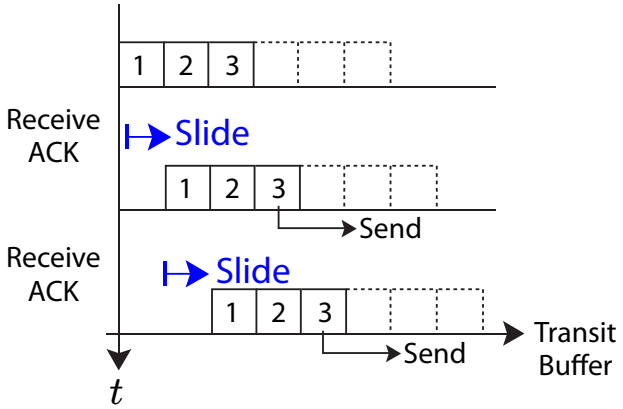


Figure 14.2 An illustration of a sliding window of size 3. When three packets are outstanding, *i.e.*, have not been acknowledged, transmission has to pause. As each acknowledgement is received, the window is slid by one packet, allowing a new packet to be transmitted.

on transmission rate introduces the so-called *self-clocking property* driven by the acknowledgement packets. A picture illustrating the sliding-window operation is shown in Figure 14.2.

- *Additive increase and multiplicative decrease.* We will not have the time to discuss the details of how the `cwnd` value is initialized as a new TCP connection is established, during the so-called **slow start** phase. We focus on the **congestion avoidance** phase instead. If there is no congestion, `cwnd` should be allowed to grow, to efficiently utilize link capacities. *Increasing* the `cwnd` value is different from *sliding* the window under the same given `cwnd` value: `cwnd` becomes larger in addition to getting slid forward. And in TCP, when `cwnd` grows, it grows *linearly*: `cwnd` is increased by 1/`cwnd` upon receiving each acknowledgement. That means over 1 round trip time, `cwnd` grows by 1 if all ACKs are properly received. This operation is shown in the space-time graph in Figure 14.3. But if there is congestion, `cwnd` should be reduced so as to alleviate congestion. And TCP says when `cwnd` is cut, it is cut *multiplicatively*: `cwnd` next time is, say, half of its current value. Increasing `cwnd` additively and decreasing it multiplicatively means that the control of packet injection into the network is conservative. It would have been much more aggressive if it were the other way around: multiplicative increase and additive decrease.
- *Infer congestion by packet loss or delay.* But how do you know if there is congestion? If you are an iPhone running a TCP connection, you really have no idea what the network topology looks like, what path your packets are taking, which other end hosts share links with you, and which links

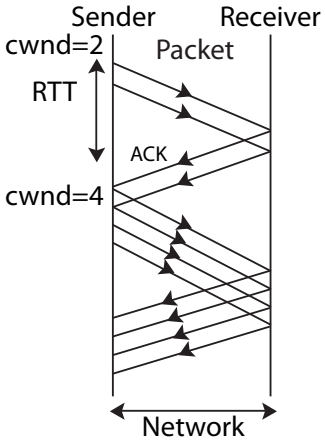


Figure 14.3 The space-time diagram of TCP packets being sent and acknowledged. The horizontal distance between the two vertical lines represents the spatial distance between the sender and the receiver. The vertical axis represents time. As two acknowledgements are received by the sender, the congestion window is not only slid, but also increased by 1.

along the path are congested. You only have a local and noisy view, and yet you have to make an educated guess: is your connection experiencing congestion somewhere in the network or not? The early versions of TCP congestion control made an important assumption: if there is a packet loss, there is congestion. This sounds reasonable enough, but sometimes packet loss is caused by a bad channel, like in wireless links, rather than congestion. In addition, often it is a little too late to react to congestion by the time packets are already getting dropped. The first problem has been tackled by many proposals of TCP for wireless. The second problem is largely solved by using packet delay as the congestion feedback signal. Instead of a binary definition of congestion or no congestion, delay value implies the *degree* of congestion.

- *Estimate packet loss and delay by timers.* Assuming that you agree packet loss or delay implies congestion, how can you tell if a packet is lost and how do you calculate delay? TCP uses two common sense approximations. (1) If the sender waits for a long time and the acknowledgement does not come back, probably the packet is lost. How long is a "long time?" Say this timeout timer is 3 times the normal **round trip time** (RTT) between the sender and the receiver. And what is the "normal" RTT? The sender timestamps each packet, and can tell the RTT of that packet once the acknowledgement is received at a later time. This is how the sender calculates delay for each packet. Then it can calculate a moving-averaged RTT. (2) Each packet sent has a sequence number, and if the sender hears from the receiver

that several, say three, later packets (numbered 10, 11, and 12) have been received but this particular packet 9 is still missing, that probably means packet 9 is lost. Packet 9 may have traversed a different path with a longer RTT (as discussed in IP routing in the last chapter), but if as many as three later packets already arrived, chances are that packet 9 is not just late but lost.

As mentioned in the last chapter, TCP/IP is the "thin waist" of the Internet layered protocol stack. It glues the functional modules below it, like the physical and link layers, to those above it, like the application layer. There are alternatives to TCP in this thin waist, such as the connectionless UDP that does not maintain an end-to-end feedback control. As part of that thin waist, the above five elements of congestion control design in TCP lead to a great success. The reason the Internet has not collapsed, despite the incredible and unstoppable surge of demand, is partially attributable to its congestion control capability.

Starting with **TCP Tahoe** in 1988 and its slightly modified variant **TCP Reno** in 1990, TCP congestion control had gone through twenty years of improvement. For example, **TCP Vegas** in 1995 shifted from a loss-based congestion signal to a delay-based congestion signal. **FAST TCP** in 2002 stabilized congestion control to achieve high utilization of link capacity. **CUBIC** in 2005 combined loss- and delay-based congestion signals, and is now the default TCP in the Linux kernel. There have also been many other variants of TCP congestion control proposed over the past two decades.

If you think about it, for end-to-end congestion control without any message passing from the network, an end host (like a laptop) really has very little to work with. Estimates of packet loss and calculations of packet delay are pretty much the only two pieces of information it can obtain through time-stamping and numbering the packets.

14.2.1 Loss-based congestion inference

For loss-based congestion control like TCP Reno, a major TCP variant especially for the Windows operating system, the main operations are:

- If all the w outstanding packets are received at the receiver properly (*i.e.*, in time and not out-of-order more than twice), increase `cwnd` window size by 1 each RTT, *e.g.*, from w to $w + 1$.
- Otherwise, decrease it by cutting in half, *e.g.*, from w to $w/2$.

There are also other subtle features like Fast Retransmit and Fast Recovery that we will not have time to get into.

Let us look at an example. For simplicity, let RTT = 1 unit, and assume it is a constant. Actually, RTT is about 50ms across the USA and varies as congestion condition changes. Initialize `cwnd` $\leftarrow 5$. Suppose all packets are successfully

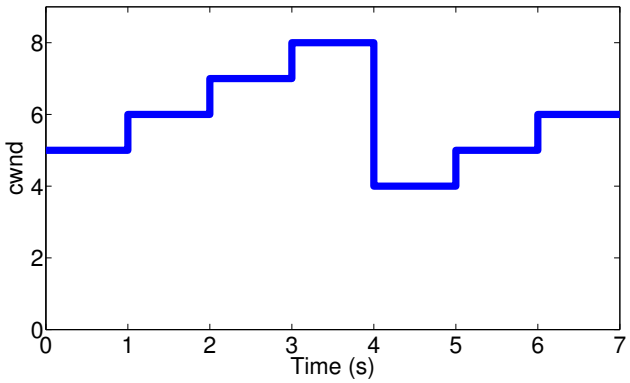


Figure 14.4 Zoomed-in view of `cwnd` evolution over time for TCP Reno, with RTT=1 unit of time.

received and acknowledged (ACK) during each RTT, except at $t = 4$, when a packet loss occurs.

At $t = 0$, `cwnd`=5, so the sender sends 5 packets and pauses.

At $t = 1$, the sender has received 5 ACKs, so it slides the congestion window by 5 packets and increases `cwnd` by 1. It sends 6 packets.

At $t = 2$, the sender has received 6 ACKs, so it sends 7 packets.

At $t = 3$, the sender has received 7 ACKs, so it sends 8 packets.

At $t = 4$, the sender detects a lost packet. It halves `cwnd` to 4, and sends 4 packets.

At $t = 5$, the sender has received 4 ACKs, so it sends 5 packets.

At $t = 6$, the sender has received 5 ACKs, so it sends 6 packets...

Figure 14.4 shows these values of `cwnd` over time. When there was no packet loss ($t = 0, 1, 2, 3$), `cwnd` grew linearly. When the packet loss occurred ($t = 4$), `cwnd` decreased sharply, then began growing linearly again ($t = 5, 6$).

Zooming out, Figure 14.5(a) shows a typical evolution of TCP Reno's `cwnd` over time. The y-axis is the congestion window size. If you divide that by RTT and multiply it by packet size, you get the actual transmission rate in bps.

14.2.2 Delay-based congestion inference

Now we turn to delay-based congestion control like TCP Vegas. We first have to appreciate that the total RTT is mostly composed of both **propagation delay**, the time it takes to just go through the links, and **queuing delay**, the time a packet spends waiting in the queue due to congestion. The heavier the congestion, the longer the wait. So the sender needs to estimate RTT_{min}, the minimum RTT that tells the sender what the delay value should be if there is (almost) no congestion.

Then, upon receiving each acknowledgement, the sender looks at the difference between w/RTT_{min} and w/RTT_{now}. It is the difference between the transmission rate (in packets per second) without much congestion delay and that with the current congestion delay.

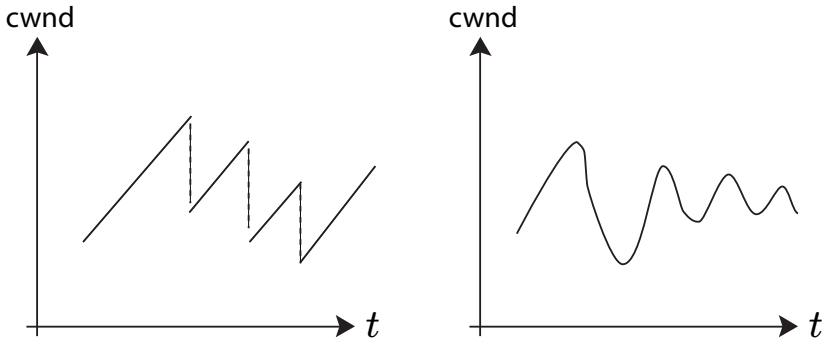


Figure 14.5 Typical evolution of `cwnd` values in TCP Reno and TCP Vegas. TCP Reno uses loss as the congestion signal whereas TCP Vegas uses delay as the congestion signal. The zigzags between overshooting and under-utilizing capacity tends to be smaller in Vegas if the parameters are properly tuned.

- If this difference is smaller than a prescribed threshold, say 3, that means there is little congestion, and `cwnd` is increased by 1.
- If the difference is larger than the threshold, that means there is some congestion, and `cwnd` is decreased by 1.
- If the difference is exactly equal to the threshold, `cwnd` stays the same.
- If all sources stop adjusting their `cwnd` values, an equilibrium is reached.

We can compare this congestion control with the power control in Chapter 1: at the equilibrium everyone stops changing its variable simultaneously. We would like to know what exactly is the resource allocation at such an equilibrium, and if it can be reached through some simple, iterative algorithm.

Figure 14.5(b) shows a typical evolution of TCP Vegas' `cwnd` over time. You can see that the zigzag between a rate that is too aggressive (leading to congestion) and an overly conservative a rate (leading to under-utilization of link capacities) can be reduced, as compared to TCP Reno. Using delay as a *continuous* signal of congestion is better than using only loss as a *binary* signal, and we will see several arguments for this observation in the next section.

14.3 A Long Answer

Whether distributedly like TCP or through a centralized command system, any protocol trying to control congestion in a network must consider this fundamental issue: each link l's fixed capacity c_l is shared by multiple sessions, and each of these end-to-end sessions traverses multiple links. We assume each source i has one session and uses a single path routing. So, "flows," "sessions," and "sources"

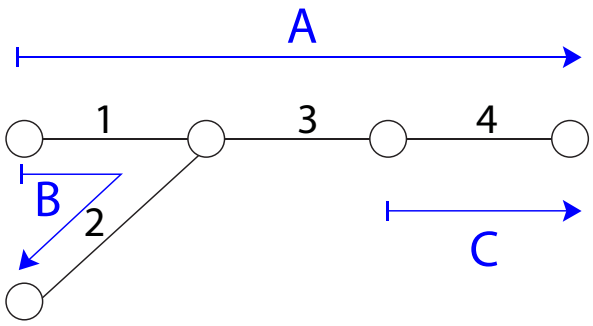


Figure 14.6 A simple network with 4 links and 3 sessions. Sessions A and B share link 1, and sessions A and C share link 4. Constrained by the fixed capacities on the four links, it is not trivial to design a distributed algorithm that allocates the capacities in an efficient and fair way among the three competing sessions.

are interchangeable terms here. Each link l is shared by a set of sessions $S(l)$, and each session i uses a set of links $L(i)$ along its path decided by IP routing.

Consider the simple example in Figure 14.6. As we will see in Chapter 19, it is often accompanied by control signaling that traverses other paths too. Sometimes, the contents of one session also reside at different locations, *e.g.*, advertisements on a webpage need to be downloaded from a different server than the actual content of the webpage. We ignore these factors here.

In this graph, session A originating from node 1 traverses links 1, 3 and 4. And link 1 is shared by sessions A and B. Even when link 3 is not fully utilized, we cannot just increase session A's rate since the bottleneck link for that session may be link 1.

How can we allocate each link's capacity so that the sessions collectively use as much capacity as they can without causing congestion, and their competition is fairly coordinated? A capacity allocation must first be feasible under the link capacity constraints, and then, also be efficient and fair.

In Figure 14.6, consider each link's capacity as 1 Mbps. One feasible solution that satisfies all four links' capacity constraints, thus a feasible solution, is $[0.5, 0.5, 0.5]$ for the three sessions A, B, and C. In this equal distribution of end-to-end rates, the efficiency of link capacity utilization is 3 Mbps across all the links. For the same capacity utilization, another feasible solution is $[1, 0, 0]$, which starves sessions B and C, and probably is not viewed as a fair allocation. It turns out a standard notion of fairness, called proportional fairness, would give the allocation $[1/3, 2/3, 2/3]$ to the three competing sessions. This may make intuitive sense to some people, as session A traverses two links that are potential bottlenecks.

Now we need to write down the problem statement more precisely. We will call this optimization the basic **Network Utility Maximization** (NUM) prob-

lem. It is a networked version of the social welfare problem in Chapter 11. We will soon present a distributed solution to this problem in this section, leaving the derivation steps to Advanced Material. Then we will show that TCP Reno and TCP Vegas actually can be reverse-engineered as solutions to specific NUM problems.

14.3.1 Formulating NUM problem

We need to address two issues in modeling congestion control: how to measure efficiency and fairness, and how to capture capacity constraint?

How do we measure efficiency and fairness? We use the utility functions introduced in Chapter 11. We then sum up each individual TCP session's utility to the end user. We model utility as a function of the end-to-end transmission rate of a TCP session here, since we are only adjusting these rates and we assume the application's performance only depends on this rate. Fairness may also be captured by some of these utility functions, like the α-fair utility functions.

How do we represent the link capacity constraint? On each link l, there's a limited capacity c_l in bps. The load must be smaller than c_l. There are several ways to express the load on a link in terms of the variable transmission rates at the sources and the given routing decisions.

We can write the load on link l as the sum of source rates x_i across those sources using this link: $\sum_{i \in S(l)} x_i$. Or, we can use R_{li} as a binary-valued indicator, so that $R_{li} = 1$ if source i's session traverses link l, and $R_{li} = 0$ otherwise. Then the load on link l is simply $\sum_i R_{li} x_i$. In this notation, you can readily see that the constraints

$$\sum_i R_{li} x_i \leq c_l, \ \ \forall l, \tag{14.1}$$

are equivalent to the following linear inequalities in a matrix notation:

$$\mathbf{R}\mathbf{x} \leq \mathbf{c}, \tag{14.2}$$

where $\leq$ between two vectors means component-wise $\leq$ between the corresponding entries of the vectors.

For example, in the network topology in Figure 14.6, the link capacity constraint in matrix form becomes

$$\begin{pmatrix} 1 & 1 & 0 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \\ 1 & 0 & 1 \end{pmatrix} \begin{pmatrix} x_A \\ x_B \\ x_C \end{pmatrix} \leq \begin{pmatrix} c_1 \\ c_2 \\ c_3 \\ c_4 \end{pmatrix}.$$

14.3.2 Distributed algorithm solving NUM

Now we have completely specified the link capacity allocation problem that prescribes what congestion control should be solving:

$$\begin{array}{ll} \text{maximize} & \sum_i U_i(x_i) \\ \text{subject to} & \mathbf{Rx} \leq \mathbf{c} \\ \text{variables} & x_i \geq 0, \ \ \forall i. \end{array} \tag{14.3}$$

We refer to this problem as the basic NUM problem. (14.3) is easy to solve for several reasons:

- It is a convex optimization problem, as defined in Chapter 4. More precisely, this time it is maximizing a concave utility function (the sum of all sessions' utilities) rather than minimizing a convex cost function. Therefore, it enjoys all the benefits of being convex optimization: a locally optimal solution is also globally optimal, the duality gap is zero (under some technical condition satisfied here), and it can be solved very efficiently in a centralized computer.
- It is also decomposable. **Decomposition** here refers to breaking up one optimization problem into many smaller ones, somehow coordinated so that solving them will be equivalent to solving the original one. Why would we be interested in having many problems instead of just one? Because such a decomposition leads to a distributed algorithm. Each of these smaller problems is much easier to solve, often locally at each node in a network. And if their coordination can be done without explicit message passing, we have a truly distributed way to solve the problem.

Postponing the derivation of decomposition to Advanced Material, we have the following solution to (14.3), consisting of source actions and router actions.

At each of the discrete time slots $[t]$, the source of each session simply decides its transmission rate from its demand function, with respect to the current price along its path. This path price q_i is the sum of link prices p_l along all the links this session traverses: $q_i = \sum_{l \in L(i)} p_l$.

$$x_i[t] = D_i(q_i[t]) = U^{'-1}(q_i[t]). \tag{14.4}$$

Of course, in a sliding window based implementation, the source adjusts its window size `cwnd` rather than x_i directly. The path price serves as the *congestion feedback signal* from the network. We hope it can be obtained without explicit message passing in actual implementations.

At the same time, the router on each link l updates the "price" on that link:

$$p_l[t] = \{p_l[t-1] + \beta\,(y_l[t] - c_l)\}^+, \tag{14.5}$$

where y_l is the total load on link l: $y_l[t] = \sum_{i \in S(l)} x_i[t]$. And $\{\ldots\}^+$ simply says that if the expression inside the bracket takes on negative value, then just return 0. In this case, it means that link price is never allowed to be negative. This is

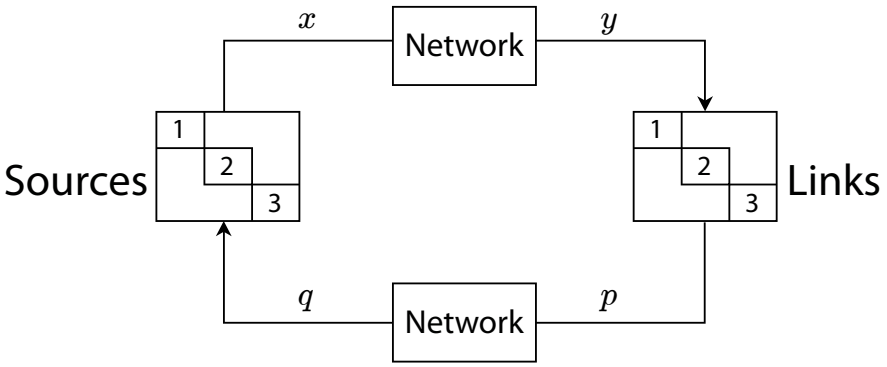


Figure 14.7 The feedback control loop in the distributed solution of NUM. Each source autonomously adapts its window size (or, transmission rate x_i) based on the path congestion price feedback q_i, while each link autonomously adapts its congestion price p_l based on its own load y_l.

an *interpretation* in the language of pricing, not actual money changing hands between network entities like in Chapters 11 and 12. Parameter $\beta \geq 0$ is the **stepsize** that controls the tradeoff between a convergence guarantee and the convergence speed. Think about playing golf: if you hit the ball too hard, even in the right direction, it will fly by the hole, and you will have to hit backward again. Stepsize is like the magnitude of your force. If it is sufficiently small, the above algorithm is guaranteed to converge, and converge to the right solution to (14.3). If it is too small, the guaranteed convergence becomes too slow. In real systems, tuning this parameter is not easy.

The feedback loop in the pair of equations (14.4,14.5) is illustrated in Figure (14.7) . It makes sense from an economic stanpoint. If at time t, there is more load than there is capacity on link l, then price p_l will go up according to (14.5), and the price for all paths containing link l will rise, in the next timeslot $t+1$. Higher price will reduce demand according to (14.4), and x_i will drop at all sources that use link l, helping to restore the balance between demand and supply on link l. This pricing signal balances the elastic demand and the fixed supply of capacities. What is interesting is that it carries out this task distributedly through a network consisting of many links and sessions. Mathematically, these link prices turn out to be the variables in a "mirror image" of the basic NUM problem: the variables in the Lagrange dual problem of NUM.

The above algorithm not only solves the basic NUM (for proper β), but solves it in a very nice way: fully distributed and intuitively motivated.

- As clearly shown in (14.5), each link only needs to measure its own total load.

It does not need to know any other link's condition, nor even the load coming from each of the sources using it.

- As clearly shown in (14.4), each source only needs to know the total price along the path it is using. It does not need to know any other path or source's condition, nor even the price per link along the path that it is using. If the path price q_i can be measured locally at each source i without explicit message passing, this would be a completely distributed solution. That is the case for using packet losses as the price in TCP Reno, and packet delays as the prices in TCP Vegas.

14.3.3 Reverse engineering

At this point, it might feel like the first section of this chapter and the current section are somehow disconnected. Are the TCP congestion control protocols implemented in the real world related to the distributed solution of the basic NUM problem? Roughly a decade after the first TCP congestion control protocol was invented, researchers reverse engineered these protocols and showed that they actually can be interpreted, approximately, as solutions to NUM. If you describe a protocol to me, I can tell you what is the utility function being implicitly maximized and what are the price variables.

For example, it turns out that TCP Reno implicitly maximizes arctan utilities, with packet losses as the price. And TCP Vegas implicitly maximizes logarithmic utilities, with packet delays as the price. In Advanced Material, we will present the derivation in the case of TCP Reno, and a similar derivation is in a homework problem for TCP Vegas.

Reverse engineering presents a peculiar viewpoint: give me the solution and I will tell you what is the problem being solved by this solution. You might wonder why I would care about the problem if I already have the solution? Well, discovering the underlying problem being solved provides a rigorous understanding on why the solution works, when it might not work, and how to make it work better. It also leads to new designs: forward engineering based on insights from reverse engineering.

Here is an example of the implications on the properties of TCP derived from reverse engineering. Since TCP Reno implicitly solves NUM with arctan utility, we know that the equilibrium packet loss rate, *i.e.*, the optimal dual variables, cannot depend on parameters that do not even show up in NUM. In particular, if we double the buffer size, it will not help reduce the equilibrium packet loss rate, since buffer size does not appear in the NUM problem (14.3). Intuitively, what happens is that increasing buffer size simply postpones the onset of congestion and packet loss.

Here is another example: since TCP Vegas is reverse engineered as a solution to log utility maximization, it leads to a proportionally fair allocation of link capacities. Of course, that does not guarantee TCP Vegas will converge at all. But if it does converge, we obtain proportional fairness.

In addition to the mentality of reverse engineering, we have introduced two important themes in this chapter, themes that run beyond just TCP congestion control in the Internet:

- *Feedback signal can be generated and used in a network for distributed coordination.* Selfish interests of the users may be aligned with pricing signals to achieve a global welfare maximization. In some cases, the pricing signals do not even require explicit message passing, an additional benefit not commonly found in general.
- *A network protocol can be analyzed and designed as a control law.* Properties of Internet protocols can be analyzed through the trajectories of the corresponding control law. This might sound straightforward once it has been stated, but it was an innovative angle when first developed in the late 1990s, and opened the door to thinking about network protocols not just as bottom-up, trial-and-error solutions, but also as the output of a first-principled, top-down design methodology.

14.4 Examples

Consider the network shown in Figure 14.6 again. The routing matrix, where the rows are the links and the columns are the sources, is

$$\mathbf{R} = \begin{bmatrix} 1 & 1 & 0 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \\ 1 & 0 & 1 \end{bmatrix}.$$

Assume the capacity on all links is 1 Mbps and the utility function is a log function for all sources, *i.e.*, $U_i(x_i) = \log x_i$. Our job is to find the sending rate of each source: x_A, x_B, x_C. The NUM problem is formulated as:

$$\begin{aligned} \text{maximize} \quad & \log(x_A) + \log(x_B) + \log(x_C) \\ \text{subject to} \quad & \begin{bmatrix} 1 & 1 & 0 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \\ 1 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_A \\ x_B \\ x_C \end{bmatrix} \leq \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}, \\ & x_i \geq 0, \ \forall\, i. \end{aligned} \tag{14.6}$$

Recall that the source rates and the link prices converge to the distributed solution to the NUM problem by the following iterative updates:

$$x_i[t] = U'^{-1}(q_i[t]) = \frac{1}{q_i[t]} \tag{14.7}$$

$$p_l[t] = \{p_l[t-1] + \beta(y_l[t] - c_l)\}^+, \tag{14.8}$$

where q_i is the path price seen by source i (we step over the time index here), and y_l is the total load on link l:

$$q_i[t] = \sum_{l \in L(i)} p_l[t-1]$$
$$y_l[t] = \sum_{i \in S(l)} x_i[t].$$

Let us initialize the source rates to 0 and the link costs to 1, *i.e.*, $x_A[0] = x_B[0] = x_C[0] = 0$ and $p_1[0] = p_2[0] = p_3[0] = p_4[0] = 1$. Let stepsize $\beta = 1$.

At $t = 1$, we first update the source rates. Since

$$q_A[1] = p_1[0] + p_3[0] + p_4[0] = 1 + 1 + 1 = 3$$
$$q_B[1] = p_1[0] + p_2[0] = 1 + 1 = 2$$
$$q_C[1] = p_4[0] = 1,$$

we have, in Mbps,

$$x_A[1] = \frac{1}{q_A[1]} = 0.333$$
$$x_B[1] = \frac{1}{q_B[1]} = 0.5$$
$$x_C[1] = \frac{1}{q_C[1]} = 1.$$

We then update the link prices. Since the link loads are, in Mbps,

$$y_1[1] = x_A[1] + x_B[1] = 0.333 + 0.5 = 0.833$$
$$y_2[1] = x_B[1] = 0.5$$
$$y_3[1] = x_A[1] = 0.333$$
$$y_4[1] = x_A[1] + x_C[1] = 0.333 + 1 = 1.33,$$

we have

$$p_1[1] = [p_1[0] + y_1[1] - c]^+ = [1 + 0.833 - 1]^+ = 0.833$$
$$p_2[1] = [p_2[0] + y_2[1] - c]^+ = [1 + 0.5 - 1]^+ = 0.5$$
$$p_3[1] = [p_3[0] + y_3[1] - c]^+ = [1 + 0.333 - 1]^+ = 0.333$$
$$p_4[1] = [p_4[0] + y_4[1] - c]^+ = [1 + 1.33 - 1]^+ = 1.33.$$

At $t = 2$, we update the source rates. Since

$$q_A[2] = p_1[1] + p_3[1] + p_4[1] = 0.833 + 0.333 + 1.33 = 2.5$$
$$q_B[2] = p_1[1] + p_2[1] = 0.833 + 0.5 = 1.33$$
$$q_C[2] = p_4[1] = 1.33,$$

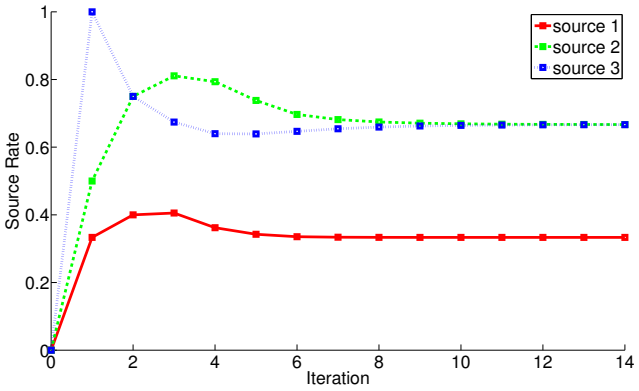


Figure 14.8 Source rates converge over time in a congestion control example.

we have, in Mbps,

$$x_A[2] = \frac{1}{q_A[2]} = 0.4$$
$$x_B[2] = \frac{1}{q_B[2]} = 0.75$$
$$x_C[2] = \frac{1}{q_C[2]} = 0.75.$$

We then update the link prices. Since the link loads are, in Mbps,

$$y_1[2] = x_A[2] + x_B[2] = 0.4 + 0.75 = 1.15$$
$$y_2[2] = x_B[2] = 0.75$$
$$y_3[2] = x_A[2] = 0.4$$
$$y_4[2] = x_A[2] + x_C[2] = 0.4 + 0.75 = 1.15,$$

we have

$$p_1[2] = [p_1[1] + y_1[2] - c]^+ = [0.833 + 1.15 - 1]^+ = 0.983$$
$$p_2[2] = [p_2[1] + y_2[2] - c]^+ = [0.5 + 0.75 - 1]^+ = 0.25$$
$$p_3[2] = [p_3[1] + y_3[2] - c]^+ = [0.333 + 0.4 - 1]^+ = 0$$
$$p_4[2] = [p_4[1] + y_4[2] - c]^+ = [1.33 + 1.15 - 1]^+ = 1.48.$$

These iterations continue. We plot their evolution over time in Figures 14.8 and 14.9.

We see that an equilibrium is reached after about 10 iterations. At this point, the source rates are:

$$x_A^* = 0.33 \text{ Mbps}$$
$$x_B^* = 0.67 \text{ Mpbs}$$
$$x_C^* = 0.67 \text{ Mbps}.$$

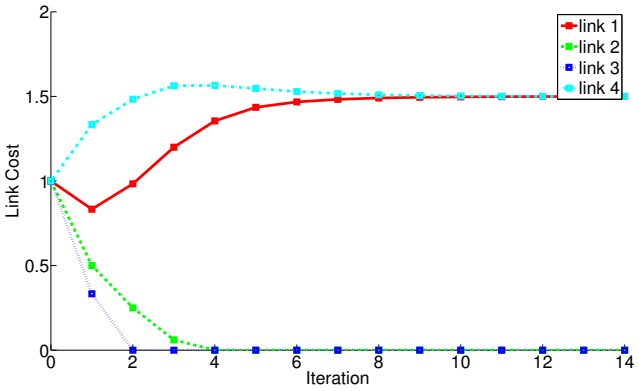


Figure 14.9 Link prices converge over time in a congestion control example.

It makes sense that, for proportional fairness, the session that takes up more network resources is given a lower rate: session 1 traverses twice as many bottleneck links and receives half as much the allocated rate.

As a sanity check, let us make sure that the equilibrium values satisfy the constraints in (14.6). Obviously, all the x_i^* are non-negative. To check the link capacity constraint, we see that

$$\begin{bmatrix} 1 & 1 & 0 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \\ 1 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_A \\ x_B \\ x_C \end{bmatrix} = \begin{bmatrix} 1 & 1 & 0 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \\ 1 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0.33 \\ 0.67 \\ 0.67 \end{bmatrix} = \begin{bmatrix} 1 \\ 0.67 \\ 0.33 \\ 1 \end{bmatrix} \leq \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}.$$

At equilibrium, the link prices are:

$$\begin{aligned} p_1^* &= 1.5 \\ p_2^* &= 0 \\ p_3^* &= 0 \\ p_4^* &= 1.5. \end{aligned}$$

It makes sense that links 2 and 3 have 0 price at equilibrium, since their capacity is not fully utilized, as constrained by the way the sessions are routed. Conversely, links 1 and 4 have strictly positive prices at equilibrium, and by the **complementary slackness** property in Advanced Material, we know that their link capacities must be fully utilized, *i.e.*, links 1 and 4 are the bottleneckes.

14.5 Advanced Material

14.5.1 Decomposition of NUM

In this subsection, we derive the solution (14.4, 14.5) to the basic NUM problem (14.3). The objective function is already decoupled across the sessions indexed

by i: each term U_i in the sum utility only depends on the rate x_i for that session i. So we just need to decouple the constraints. It is precisely this set of linear capacity constraints that couples the sessions together through the given routing matrix $\mathbf{R}$.

The decomposition method we will use is called **dual decomposition**, since it actually solves the **Lagrange dual problem** of NUM. Given any optimization problem, we can derive a "mirror" problem called the dual problem. Sometimes the dual problem's optimized objective function value equals that of the original primal problem. And at all times, it provides a performance bound to that of the original problem. The dual problem can sometimes be solved much faster, and in our case, solved in a distributed way.

The first step in deriving the Lagrange dual problem is to write down the **Lagrangian**: the sum of the original objective function and a weighted sum of the constraints ($c_l - \sum_{i \in S(l)} x_i \geq 0$). The positive weights are called **Lagrange multipliers p**, interpreted as the link prices:

$$L(\mathbf{x}, \mathbf{p}) = \sum_i U_i(x_i) + \sum_l p_l \left(c_l - \sum_{i \in S(l)} x_i \right).$$

The intuition is that we change a constrained optimization to a much easier, *unconstrained* one, by moving the constraints up to augment the objective function. The hope is that if we set the right weights $\mathbf{p}$, we can still get the original problem's solution.

Next, we group everything related to the variables $\mathbf{x}$ together, in order to try to extract some structure in the Lagrangian:

$$L(\mathbf{x}, \mathbf{p}) = \sum_i U_i(x_i) - \sum_l \sum_{i \in S(l)} p_l x_i + \sum_l c_l p_l.$$

Now suddenly something almost magical happens: we can rewrite the double summation above by reversing the order of summation: first sum over l for a given i and then sum over all i, which allows us to rewrite the part of L involving the rate variables $\mathbf{x}$ as follows:

$$L(\mathbf{x}, \mathbf{p}) = \sum_i \left[U_i(x_i) - \left(\sum_{l \in L(i)} p_l \right) x_i \right] + \sum_l c_l p_l.$$

For example, for the network in Figure 14.6, we have $L(x_A, x_B, x_C, p_1, p_2, p_3, p_4) =$

$$U_A(x_A) - (p_1 + p_3 + p_4)x_A + U_B(x_B) - (p_1 + p_2)x_B + U_C(x_C) - p_4 x_C + c_1 p_1 + c_2 p_2 + c_3 p_3 + c_4 p_4.$$

We denote the path price (the sum of prices along the links used by session i) as q_i, so $q_i = \sum_{l \in L(i)} p_l$. Then we have a decomposed Lagrangian: the maximization over $\mathbf{x}$ can be independently carried out by each source i (see the square bracket within $\sum_i$ below):

$$L(\mathbf{x}, \mathbf{p}) = \sum_i \left[U_i(x_i) - q_i x_i \right] + \sum_l c_l p_l.$$

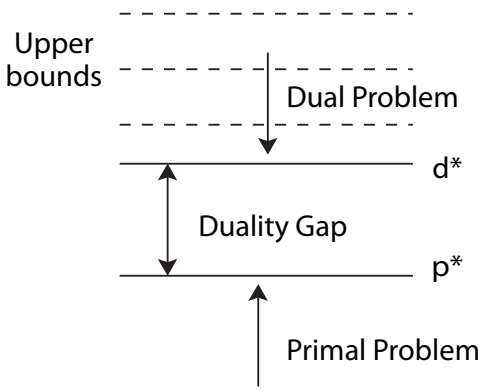


Figure 14.10 Suppose we have a maximization problem, which we will call the primal problem, with an optimized objective function's value p^*. There is a corresponding Lagrange dual problem, which is a minimization problem, with an optimized objective function's value d^*. Any feasible solution in the dual problem generates an upper bound on the primal problem's p^*. That is called weak duality. The tightest bound is d^*, which may still have a gap from p^*. If there is no gap, as is the case when the primal problem is convex optimization and satisifes some technical conditions, we say the strong duality property holds.

Suppose now we maximize the Lagrangian over the original variables $\mathbf{x}$. This was our plan in the first place: to turn a constrained optimization into an unconstrained one. Of course the maximizer and the maximized L value depend on what Lagrange multipliers $\mathbf{p}$ we used. So, we have to denote the resulting value as a function of $\mathbf{p}$:

$$g(\mathbf{p}) = \max_{\mathbf{x}} L(\mathbf{x}, \mathbf{p}).$$

This function $g(\mathbf{p})$ is called the Lagrange dual function.

It turns out that no matter what $\mathbf{p}$ we use (as long as they are non-negative), $g(\mathbf{p})$ is always a performance bound. It is an upper bound on the maximum $U^* = \sum_i U_i(x_i^*)$ of the original NUM problem. This is easy to see. Consider the maximizer of the NUM problem $\mathbf{x}^*$. It must be a feasible vector, and $\mathbf{p}$ is non-negative. So the Lagrangian L must be larger than U^* when $\mathbf{x} = \mathbf{x}^*$. Since the Lagrange dual function g is the largest Lagrangian over all $\mathbf{x}$, it must also be larger than U^*. This is called the **weak duality** property, which actually holds for all optimization problems.

How about we tighten this bound $g(\mathbf{p})$, by picking the best $\mathbf{p}$? We call the resulting problem the Lagrange dual problem, and give the name Lagrange dual variables to $\mathbf{p}$ now:

$$\text{minimize}_{\mathbf{p}} \ \ g(\mathbf{p}),$$

As illustrated in Figure 14.10, if this tightening generates the exact answer to the original optimization, we say the optimal **duality gap** is zero, and the property of **strong duality** holds. Together with some technical conditions, the original problem being a convex optimization problem is a sufficient condition for strong duality to hold. This is another reason why convex optimization is easy.

Applying the above **dual decomposition** method of breaking up one problem into many smaller problems to NUM, we see that the first step is maximizing over $\mathbf{x}$ for a given $\mathbf{p}$. This is nothing but net utility maximization we saw in Chapter 11, selfishly and distributedly carried out at each source now:

$$x_i^*(\mathbf{p}) = \text{argmax}[U_i(x_i) - q_i x_i].$$

We obtain exactly (14.4).

The second step, minimizing the Lagrange dual problem's objective function $g(\mathbf{p}) = L(\mathbf{x}^*(\mathbf{p}), \mathbf{p})$, over $\mathbf{p}$, can be carried out by the gradient method. Go down along the direction of the negative gradient with a stepsize β, as illustrated in Figure 14.11:

$$\mathbf{p}[t+1] = \mathbf{p}[t] - \beta\,(\text{ Gradient of } g(\mathbf{p}) \text{ at } \mathbf{p}[t])\,.$$

It turns out that for a linearly constrained, concave maximization problems like NUM, the constraint function itself $c_l - \sum_{i \in S(l)} x_i = c_l - y_l$ is the gradient for each p_l. So all we need to do is to multiply the gradient with a stepsize β (and then make sure it is never negative):

$$p_l[t] = \left\{ p_l[t-1] - \beta \left(c_l - \sum_{i \in S(l)} x_i^*(\mathbf{p}) \right) \right\}^+ ,$$

which is exactly (14.5).

Since strong duality holds for NUM, solving the Lagrange dual problem is equivalent to solving the original problem. This concludes the derivation of (14.4,14.5) as a distributed solution algorithm to (14.3).

These optimized primal variables (the rate vector $\mathbf{x}^*$) and dual variables (the link price vector $\mathbf{p}^*$) also satisfy other useful properties, including the following **complementary slackness** property. If primal constraint l is slack: $\sum_{i \in l} x_i^* < c_l$, the corresponding optimal dual variable p_l^* (the optimal link congestion price) must be 0, *i.e.*, the non-negativity constraint in the dual problem is not slack. Conversely, if the optimal link congestion price $p_l^* > 0$ for some link l, we must have $\sum_{i \in l} x_i^* = c_l$, *i.e.*, link l is a bottleneck link at equilibrium.

14.5.2 Reverse engineering

We mentioned earlier that if you give me a TCP congestion control protocol, I can return to you a NUM problem implicitly solved by it, with the utility function completely specified, where the source rates (or window sizes) are the variables,

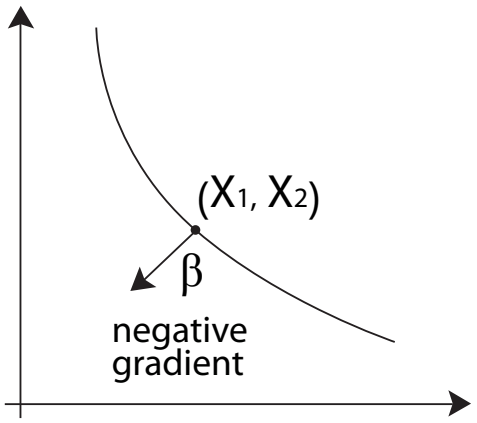


Figure 14.11 Suppose we want to minimize a function of two variables shown here. The gradient algorithm moves from the current point along the direction of the negative gradient of the function, with a certain stepsize β. Sometimes, the gradient can be computed distributively.

and the pricing signals are the Lagrange dual variables. We now illustrate the main steps in this reverse engineering approach for TCP Reno.

The first step in the derivation is to write down the evolution of `cwnd` size w as specified by the given protocol. Each time an in-order acknowledgement is received at the source, the window size grows by $1/w$ above its current size. Therefore, if every packet is properly received, the window size increases by 1 after one RTT. But each time a packet loss is detected, the window size is halved. Therefore, the net change to the window size $w[t]$ is:

$$x[t](1-q[t])\frac{1}{w[t]} - x[t]q[t]\frac{w[t]}{2}, \tag{14.9}$$

where we omitted the subscript i for notational simplicity.

Now let RTT be d and assume it is constant (even though it obviously varies in time depending on the congestion condition). Since $x = w/d$, (14.9) leads to the following difference equation:

$$x[t+1] = x[t] + \frac{1-q[t]}{d^2} - \frac{1}{2}q[t]x^2[t].$$

By definition of *equilibrium*, x does not change anymore at an equilibrium, which means $\frac{1-q}{d^2} = \frac{1}{2}qx^2$. This equilibrium condition gives us an equation connecting q with x:

$$q = \frac{2}{x^2d^2+2}.$$

From the demand function definition, we know $U_i'(x_i) = q_i$. So, if we integrate

the above expression in x, we recover the utility function:

$$U(x) = \frac{\sqrt{2}}{d}\arctan(\sqrt{1/2}x_i d). \tag{14.10}$$

In summary, TCP Reno's equilibrium solves NUM with arctan utility, with the help of packet loss as the Lagrange variables. We can also verify that complementary slackness is satisfied: if a primal constraint is slack, *i.e.*, the demand is strictly less than the capacity on a link at equilibrium, there will be neither loss nor queuing delay. Conversely, if a dual constraint is slack, *i.e.*, there is loss or queuing delay on a link, its capacity must be fully utilized.

Now, the packet loss rate q along a path is not actually equal to the *sum* of loss rates on the links along the path. It is $1 - \prod_l (1 - p_l)$, *i.e.*, 1 minus the probability that no packet is lost on any link. But when the loss rate is small, $\sum_{l \in L(i)} p_l \approx q_i$ holds pretty well as an approximation. More seriously, TCP Reno might not converge. Its equilibrium behavior may be desirable, but an equilibrium may never be reached.

We have made quite a few other assumptions implicitly along the way:

- focusing only on equilibrium behavior,
- ignoring the actual queuing dynamics inside the queues,
- forgetting about the propagation delay it takes for packets and acknowledgements to travel through the network,
- assuming that there is a fixed set of sessions sharing the network capacities, each going on forever.

Many of these assumptions have been taken away and stronger results obtained over the years. What is somewhat surprising is that, even with some of these assumptions, the theory prediction from the NUM analysis works quite well when compared to actual TCP operations.

Further, the optimization model of congestion control has lead to forward engineering of new TCP variants that are provably stable. **Stability** here means that the trajectory of a protocol's variables converges to the desired equilibrium, such as the solution to a NUM with a properly chosen utility function. In fact, some of these variants have been demonstrated in real-life experiments and then commercialized, including FAST TCP for long-distance and large-volume transmissions, and CUBIC, the default TCP in the Linux kernel.

Further Reading

Congestion control models and design have been an active research area in networking for more than 20 years.

1. A standard reference book on TCP/IP is:
[Ste94] W. R. Stevens, *TCP/IP Illustrated, Vol. 1: The Protocols*, Addison

Wesley, 1994.

2. The control dynamic system viewpoint, the optimization model, and the pricing interpretation of TCP were pioneered by Kelly in the late 1990s:
[KMT98] F. P. Kelly, A. Maulloo, and D. Tan, "Rate control in communication networks: Shadow prices, proportional fairness, and stability," *Journal of the Operational Research Society*, vol. 49, pp. 237-252, 1998.

3. Reverse engineering TCP protocols into NUM models has been summarized in the following article:
[Low03] S. H. Low, "A duality model of TCP and queue management system," *IEEE/ACM Transactions on Networking*, vol. 11, no. 4, pp. 525-536, 2003.

4. The following monograph summarized the major results in congestion control modeling up to 2004, including stochastic session arrivals and departures:
[Sri04] R. Srikant, *The Mathematics of Internet Congestion Control*, Berkhauser, 2004.

5. Generalizing the modeling approach we saw in this chapter to "layering as optimization decomposition" was surveyed in the following article:
[Chi+07] M. Chiang, S. H. Low, A. R. Calderbank, and J. C. Doyle, "Layering as optimization decomposition: A mathematical theory of network architecture," *Proceedings of the IEEE*, vol. 95, no. 1, pp. 255-312, 2007.

Problems

14.1 *A numerical example of NUM* ⋆⋆

Suppose we have the network shown in Figure 14.1, with links labeled and two sessions as shown.

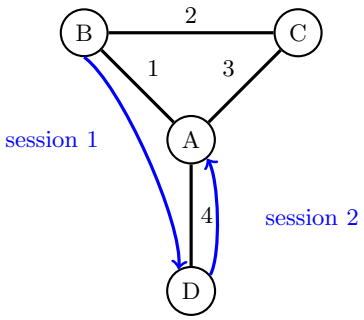


Figure 14.12 A small network for a numerical example of NUM.

(a) Write down the routing matrix $\mathbf{A}$.

(b) Run a simulation for 10 time steps to solve the NUM using the link price and source rate updates in (14.4) and (14.5). The utility function is a logarithmic function of the source rate. Initialize the link prices to 1, and run the source rate update step first. Set the step size $\beta = 1$. Plot the source rates over time and the link prices over time. What are the equilibrium values?

(c) Change the step size β and observe the impact on convergence of the algorithm.

14.2 *TCP slow start* ⋆

We learned about the primary mode of operation of TCP Reno, where *cwnd* increases by 1 for each RTT. Equivalently, *cwnd* increases by $\frac{1}{cwnd}$ for each ACK received. This operational mode is called *congestion avoidance*.

However, this results in linear increase of *cwnd* over time. At the beginning of a TCP connection, we would like to quickly ramp up *cwnd* before transitioning to congestion avoidance mode. Most TCP protocols have a (somewhat confusingly named) *slow start* phase that accomplishes exactly this. In this mode, *cwnd* increases by 1 for each ACK received.

(a) If we plot *cwnd* versus *time*, instead of having a linear increase as in congestion avoidance, what is the rate of increase in slow start?

(b) Draw a space-time diagram for the slow start phase, for 4 RTTs. Assume *cwnd* starts at 1.

14.3 *TCP Reno congestion window* ⋆⋆

Recall that the congestion window length changes with time as follows during TCP Reno's congestion avoidance phase:

- If an ACK is received, then increase *cwnd* by $\frac{1}{cwnd}$.
- If congestion is detected, then decrease *cwnd* by $\frac{cwnd}{2}$.

Suppose the probability of failed transmission is p; the probability of a successful transmission is then $1 - p$. The transmission rate $x = \frac{cwnd}{RTT}$.

(a) Write down the equation for the expected change of *cwnd* per time step.

(b) At equilibrium, the expected change is 0. Using (a), show that $x_r = \frac{1}{RTT}\sqrt{\frac{2(1-p)}{p}}$.

14.4 *TCP Vegas* ⋆⋆

TCP Vegas attempts to anticipate congestion by estimating delay. In this scheme, the congestion window size is changed based on the timings of the ACKs it has received. Specifically, we have:

$$cwnd[t+1] = \begin{cases} cwnd[t]+1 & \text{if } \frac{cwnd[t]}{d} - \frac{cwnd[t]}{D[t]} < \beta \\ cwnd[t]-1 & \text{if } \frac{cwnd[t]}{d} - \frac{cwnd[t]}{D} > \beta \\ cwnd[t] & \text{otherwise,} \end{cases}$$

where d is the minimum RTT observed historically, $D[t]$ is the RTT observed at time t, and β is a parameter. So, $\frac{cwnd}{d}$ is the expected rate and $\frac{cwnd}{D}$ is the observed rate, so $cwnd$ decreases (increases) if the expected rate is greater (smaller) than the actual rate by β.

Suppose $D[t] = \begin{cases} t & \text{if } t \leq 4 \\ 4 & \text{otherwise} \end{cases}$, and $cwnd[1] = 4, \alpha = 3$ and $d = 1$. Plot the evolution of $cwnd$ versus time and D versus time for 10 time steps.

14.5 *Reverse engineering TCP Vegas* ⋆⋆⋆

It can be shown that TCP Vegas approximately solves the following weighted logarithmic utility maximization problem:

$$\begin{aligned} \text{maximize} \quad & \sum_i \beta_i d_i \log(x_i) \\ \text{subject to} \quad & \sum_{i \in S(l)} x_i \leq c_l \ \ \forall l \\ \text{variables} \quad & x_i \geq 0, \ \forall i, \end{aligned} \tag{14.11}$$

where the links update their prices as follows:

$$p_l[t+1] = \{p_l[t] + \gamma_l \left(y_l[t] - c_l\right)\}^+, \tag{14.12}$$

and, as before,

$$\begin{aligned} L(i) &= \text{set of links used by session } i, \\ S(l) &= \text{set of sessions present on link } l, \\ y_l[t] &= \sum_{i \in S(l)} x_i[t], \\ q_i[t] &= \sum_{l \in L(i)} p_l[t]. \end{aligned} \tag{14.13}$$

We will show this through several steps.

(a) Define the total backlog on link l from all sessions as $b_l[t]$. Then each link

updates its backlog at each time step by $b_l[t+1] = \{b_l[t] + \beta(y_l[t] - c_l)\}^+$. Show that if we define $p_l[t] = \frac{b_l[t]}{c_l}$, the links update their prices as in (14.12). What is γ_l?

(b) If a network is trying to solve (14.11), what should the source update rule be? Recall that $x_i[t] = U_i'^{-1}(q_i[t])$.

(c) Recall that the session rates in TCP Vegas are updated by:

$$cwnd_s[t+1] = \begin{cases} cwnd_i[t] + \frac{1}{D_i[t]} & \text{if } \frac{cwnd_i[t]}{d_i} - \frac{cwnd_i[t]}{D_i[t]} < \beta \\ cwnd_i[t] - \frac{1}{D_i[t]} & \text{if } \frac{cwnd_i[t]}{d_i} - \frac{cwnd_i[t]}{D_i[t]} > \beta \\ cwnd_i[t] & \text{otherwise} \end{cases}$$

We also know that the backlog on link l from session i is $\frac{x_i[t]}{c_l} b_l[t]$. The congestion window size, $cwnd_i$, is the sum of the total backlog in the path of i and the bandwidth-delay product, *i.e.*, $cwnd_i[t] = \sum_{l \in L(i)} \frac{x_i[t]}{c_l} b_l[t] + d_i x_i[t]$. Show that the source rate update rule matches the answer to part (b).

15 How can Skype and BitTorrent be free?

We just went through some of the key concepts behind the TCP/IP thin-waist of the Internet protocol stack. We will now go through five more chapters on technology networks, focusing on two major trends: massive amounts of content distribution and the prevalent adoption of mobile wireless technologies.

Scaling up the distribution of content, including video content, can be carried out either through the help of peers or large data centers. These two approaches, P2P and cloud, are described in this and the next chapters, respectively. In particular, P2P illustrates a key principle behind the success of the Internet: under-specify protocols governing the operation of a network so that an overlay network can be readily built on top of it for future applications unforeseen by today's experts. It also illustrates the importance of backward compatibility, incremental deployability, and incentive alignment in the evolution of the Internet.

15.1 A Short Answer

Skype allows phone calls between IP-based devices (like laptops, tablets, smart phones) or between IP devices and normal phones. It is free for IP to IP calls. How could that be? Part of the answer is that it uses a peer-to-peer (**P2P**) protocol riding on top of IP networks.

P2P started becoming popular around 1999. For example, Kazaa and Gnutella were widely-used P2P file and music sharing systems back in the late 1990s. However, incentives were not properly designed in those first generation P2P systems; there were a lot of *free riders* who did not contribute nearly as much as they consumed.

Skype started in 2001 from Kazaa, and was acquired by eBay for $2.6 billion in 2006 and then by Microsoft for $8 billion in 2011. As of 2010, there were 663 million Skype users worldwide, and on any given day, there were, on average, 567 million minutes of Skype calls. Skype is also popular as it encrypts messages with a 256 bit AES key. Some think that it is therefore completely secure, but actually it is not. Oppressive regimes have been able to tap into Skype conversations or filter them.

BitTorrent started in 2001 as well, and is heavily used for file sharing including

movie sharing. Like Skype, it is free and uses P2P technologies. At one point, P2P was more than half of Internet traffic, and BitTorrent alone in the mid 2000s was 30% of Internet traffic. P2P sharing of multimedia content is still very popular today, with 100 million users just in BitTorrent.

P2P showcases a major success of the evolution of the Internet: make the basic design simple and allow overlay constructions. The architecture of the Internet focuses on providing simple, ubiquitous, stable, economical connectivities, and leaves the rest of the innovations to overlays to be constructed in the future for unforeseeable applications. Different types of applications, unicast as well as multicast, have been built using P2P overlays, including file sharing, video streaming, and on-demand multimedia distribution.

Both Skype and BitTorrent are free (of course the Internet connection from your device may not be free):

- Skype is free in part because it leverages peer capability to locate each other and establish connections. P2P is used for signaling in Skype.
- BitTorrent is free in part because it leverages peer uplink capacities to send chunks of files to each other, without deploying many media servers. (And it is free in part because the content shared sometimes does not incur royalty fees). P2P is used for sharing bites of content in BitTorrent.

Both Skype and BitTorrent are *scalable*. They illustrate a positive side of the "networking effect:" where each additional node in the network contributes to many other nodes. We can therefore add many more nodes as the network scales up without creating a bottleneck. Of course this assumes the nodes can effectively contribute, and that requires some smart engineering design. This **P2P law** is a refinement of our intuition about the networking effect codified in the **Metcalfe's law**: the benefit of joining a network grows as the *square* of the number of nodes. Metcalfe's law takes a generic assumption that each node is basically connected to all the other nodes, or at least the number of neighbors per node grows as a linear function of the network size. In contrast, P2P law does *not* require that, and shows that the benefit of scalability can be achieved even when each node has only a small number of neighbors at any given time, as long as these are carefully chosen.

Skype's operational details is a commercial secret. BitTorrent is much more transparent, with papers written by the founder explaining its operation. So our treatment of Skype P2P connection management will be thinner than that of BitTorrent's P2P content sharing.

15.1.1 Skype basics

To understand how the technology behind Skype works, we need to understand two major topics: voice over IP (**VoIP**) and P2P. We postpone the discussion of VoIP to Chapter 17 together with multimedia networking in general. This chapter's focus is on P2P.

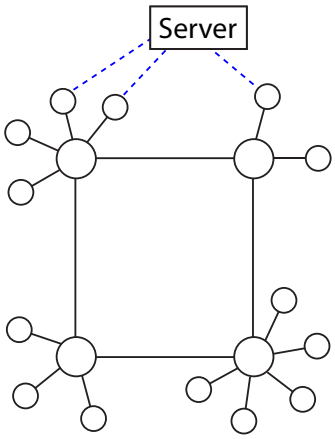


Figure 15.1 A typical topology of Skype. There is a mesh of super nodes (the bigger circles) and a shallow tree of ordinary nodes (smaller circles) rooted at each super node. There is also an authentication server (the rectangle) that each node exchanges control messages with first.

Phone calls are intrinsically P2P: a peer calls another peer (as opposed to a server). What is interesting is that Skype uses P2P to discover peers and to traverse firewalls (software and hardware that blocks incoming data connections). As shown in Figure 15.1, Skype's central directory allows a caller to discover the IP address of the callee and then establish an Internet connection. These directories are replicated and distributed in **super nodes** (SN).

The problem is that sometimes both the caller and the callee are behind firewalls, with a NAT box (see Chapter 13) in between. So the actual IP address is not known to the caller. Those outside of a firewall cannot initiate a call into the firewall.

What happens then is that super nodes have *public* IP addresses, serving as anchors to be reached by anyone and collectively acting as a network of publicly visible relays. The caller first initiates a connection with an SN, and the callee initiates a connection with another SN. Once a connection is established, two way communication can happen. The caller then calls her SN, who calls the callee's SN, who then calls the callee. Once a connection between the caller and the callee is established through these two SNs, they can also mutually agree to use just a single SN that they both can connect to, thus shortening the communication path.

15.1.2 BitTorrent basics

BitTorrent uses P2P for resource sharing: sharing upload capacities of each peer and the content stored in each peer, so that (file and multimedia) content sharing

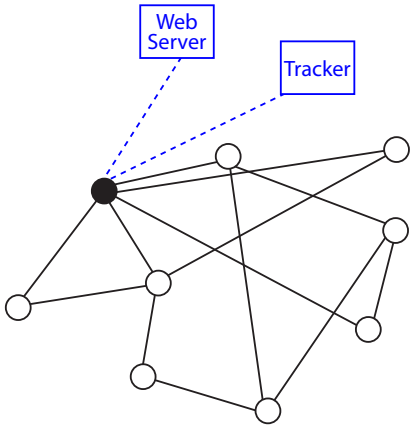


Figure 15.2 A typical topology of BitTorrent. There are actually three topologies: (a) a graph of physical connections among peers and routers, (b) a graph of overlay neighbor relationships among peers, and (c) a graph of peering relationships among peers. (c) is an overlay on (b), which is in turn an overlay on (a). This figure shows graph (c). It changes regularly depending on the list of peers provided by the tracker to, say, peer A (in black), as well as the subset of those peers chosen by peer A.

can scale itself. It is designed primarily for **multicasting**: many users all demand the same file. With P2P, they share what they have with each other.

In BitTorrent, each file is divided into small pieces called *chunks*, typically 256 kB, so that pieces of a file can be shared simultaneously. Each peer polls a centralized directory called the *Tracker*, which tells a peer a set of 50 (or so) peers with chunks of the file it needs. Then the peer picks 5 peers to exchange file chunks. This set of 5 peering neighbors is refreshed at the beginning of every time slot, based in part on how much a neighbor is helping this peer and in part on randomization.

As shown in Figure 15.3, each individual chunk traverses a tree of peers, although the overall peering relationship is a general graph that evolves in time. A **tree** is an undirected graph with only one path from one node to any other node, and there are no cycles in a tree. We usually draw a tree with the root node on top and the leaf nodes on the bottom.

We see that the control plane for signaling is somewhat centralized in both Skype and BitTorrent, but the data plane for the actual data transmission is distributed, indeed peer to peer. This is in sharp contrast to the traditional **client-server** architecture, where each of the receivers request data from a centralized server and do not help each other.

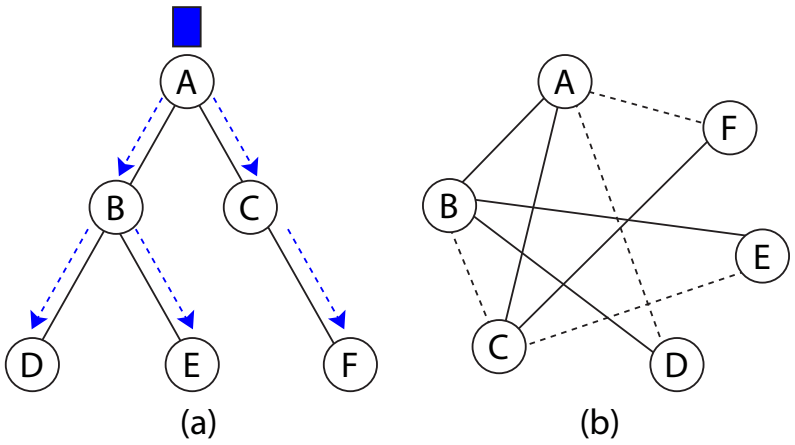


Figure 15.3 (a) Each chunk traverses a tree (with the chunk represented by the rectangle and the data transmission in dotted lines), even though (b) the peering relationships form a general graph (where the solid lines represent the current peering relationships and dotted lines represent possible peering relationships in the next timeslot).

15.2 A Long Answer

Before we go into some details of the smart ideas behind Skype and BitTorrent, we highlight two interesting observations:

- P2P is an **overlay network**, as illustrated in Figure 15.4. Given a graph with a node set V and a link set E, $G = (V, E)$, which we call the underlay, if we select a subset of the nodes in V and call that the new node set $\tilde{V}$, and we take some of the *paths* connecting nodes in $\tilde{V}$ as *links* and call that the new link sete $\tilde{E}$, we have an overlay graph $\tilde{G} = (\tilde{V}, \tilde{E})$. The Internet itself can be considered as an overlay on top of the PSTN, wireless, and other networks, and online social networks are an overlay on top of the Internet too. The idea of overlay is as powerful as that of layering in giving rise to the success of the Internet. It is evolvable: as long as TCP/IP provides the basic service of addressing, connectivity, and application interfaces, people can build overlay networks on top of existing ones. For example, multicasting could have been carried out in the network layer through **IP multicast**. And there are indeed protocols for that. But other than within a Local Area Network (see the homework problem in Chpater 13) and IPTV for channelized content (see Chapter 17), IP multicast is rarely used. The management of IP multicast tends to be too complicated, and P2P offers an alternative, overlay-based approach with less overhead.
- P2P is about *scalability*, and in BitTorrent's case, scalability in *multicasting*. If you consume, you also need to contribute. This upside of the networking

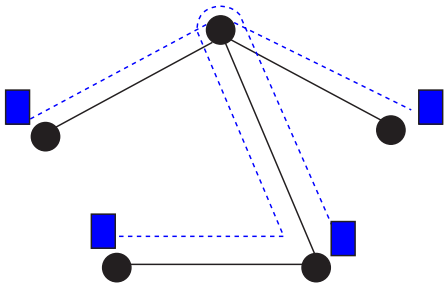


Figure 15.4 Building an overlay network of 4 nodes (the dark circles) on top of an underlay network of 5 nodes (the light circles). In the overlay graph (b), the nodes are a subset of the nodes in the underlay graph (a), and each link represents a path in the underlay graph. The overlay graph in this example is simply two parallel lines.

effect is the opposite of the wireless network interference problem, where one user's signal is other users' interference. Of course, even in BitTorrent, there is the problem of free rider: what if a user only consumes but does not contribute? We will look at BitTorrent's solution next. We will also see in the next chapter another way to provide scalability to the server-client architecture using small switches to build large ones as the data center scales up.

15.2.1 More on Skype

There are two types of nodes in Skype: super nodes (SNs) and ordinary hosts. A SN must have a public IP address, so that it can help traverse NATs and firewalls. Preferably, it should also have abundant resources, including CPU, memory, and capacities on ingress and egress links. An ordinary host must connect to a super node. Some of your desktops may actually be super nodes on Skype networks.

SNs are useful because they act as relay nodes to bypass the firewall blocking of calls as explained in the last section, and turn Skype into one of the most successful distributed systems overlaying the Internet.

Skype uses an overlay P2P network with two tiers: a mesh of super nodes and a shallow tree rooted at each super node. This two tier structure is mainly for the purpose of firewall traversal in Skype, although we will encounter it again in Advanced Material for performance optimization in P2P file sharing.

When a Skype client, whether on an ordinary host or a super node, wants to initiate a call, it must first authenticate itself with the Skype login server, which stores all the usernames and passwords. If the call is between an IP device and a PSTN phone, additional servers and procedures are required (and it is not free anymore).

Each ordinary host maintains and updates a Host Cache, which contains the IP addresses and port numbers of super nodes. During login, a host advertises its presence to other hosts, determines if it is behind a firewall, and discovers which super node to connect to and which public IP address to use. Compared to BitTorrent, the P2P topology is much more stable in Skype after login is finished.

Once logged in, a user search can be carried out through Skype's global index database. A TCP connection needs to be established for signaling between the caller and callee, and a UDP (or a TCP, if the firewall needs to be traversed) connection established for the actual voice traffic. For conferencing, more steps are needed for connection establishment.

15.2.2 More on BitTorrent

The first smart idea in BitTorrent file sharing is to use *smaller granularity* than the whole file. In this way, each chunk can traverse different trees, and the transmission can be *spatially pipelined*. The advantage of *multi-tree transmission* is similar to the advantage of multi-path routing in packet switching, which divides a given message into smaller granularity (called packets), and lets them go through possibly different paths. In fact, the richness of the tree topology compared to the path topology, and the heavy usage of multiple trees make P2P tree selection more robust than IP routing: one can pick peers without too much optimization and still achieve very high overall efficiency for the network. We will see more of this in Advanced Material.

When we discuss content distribution networks in Chapter 17, we will see that they are similar to deploying peers with large upload and storage capacities. Indeed, the term "peer" in BitTorrent refers to the fact that the node is both a sender and a receiver of content, and when it acts as a sender, it is a (small) server. In content distribution networks, deciding which content to place on which servers is a key design step. In P2P, this content placement is optimized through the strategy of *rarest chunk first*. When a peer looks at the bitmap and chooses which chunks to download, it should start with the chunks that very few peers have. By equalizing the availability of chunks, this strategy mitigates the problem where most of the peers have most of the chunks, but all must wait for the few rare chunks.

Yet another smart idea in BitTorrent is its peering construction method. The first step is for the Tracker to suggest a set of 50 or so potential peers to a new peer. These potential "friends" are recommended based on the content they have and other performance-driven factors like the distance to the new peer. They are also driven by peer churns: which peers are still sending "I am here" messages to the Tracker. A list of 50 provides a larger degree of freedom than actually used by each peer.

The second step is to let the new peer pick, at each time, her actual "friends".

These are the peers to exchange chunks with. Usually 5 peers are picked, and the upload bandwidth is evenly distributed among these 5 in the next timeslot:

- 4 of them are the top 4 peers in terms of the amount of content received from them by this node in the last time slot. This is called the **tit-for-tat** strategy.
- The remaining peer is selected at random from the set of 50.

The first feature mitigates the free rider problem, where a node could contribute but decides not to. The second feature avoids unfairness to those nodes with little upload capacity. Randomization is also generally a good idea to avoid getting trapped in a locally optimal solution. This is similar to Google's pagerank calculation in Chapter 3: 85% topology driven and 15% randomization.

Now we can summarize the BitTorrent operation, knowing why each step is designed as it is:

1. A new peer A receives a .torrent file from one of the BitTorrent web servers, including the name, size and number of chunks of a particular file, together with the IP address and port number of the corresponding Tracker.
2. It then registers with the right Tracker. It will also periodically send keep-alive messages to the Tracker.
3. The Tracker sends to peer A a list of potential peers.
4. Peer A selects a subset and establishes connections with these 5 peers.
5. They exchange bitmaps to indicate which chunks of the content they each have.
6. With chunks selected, they start exchanging chunks among themselves, starting with the rarest chunks.
7. Every now and then, each peer updates its peer list.

15.3 Examples

15.3.1 Back-of-the-envelope bounds

To illustrate the P2P networking effect, and how P2P changes the scalability property of file distribution, we run a simple back-of-the-envelope calculation:

First consider N clients requesting a file of size F bits from a server with upload capacity u_s bps. Each of these clients has a download capacity of d_i bps. This is illustrated in Figure 15.5.

- The server needs to send out NF bits, so it takes at least NF/u_s seconds.
- All the clients need to receive the file, including the slowest one with a download capacity of d_{min}, and that takes at least F/d_{min} seconds.

So the total download time is the larger of the two numbers above:

$$T = \max\left\{\frac{F}{d_{min}}, \frac{NF}{u_s}\right\}. \tag{15.1}$$

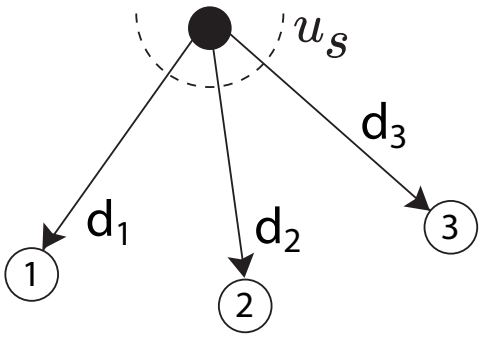


Figure 15.5 A typical server-client star topology. The upload speed of the server is u_s bps, and the download speeds of the clients are $\{d_i\}$ bps.

This could be fine, if we can increase u_s as N becomes larger. But scaling the upload capacity of a server becomes technologically and financially difficult as N becomes very large. So the alternative is to deploy more servers. Well, these N clients can also become servers themselves, and we call them peers. A hybrid peer and server deployment is what actually happens, but it is the P2P part of the network that scales itself as the number of peers increases.

Suppose each peer i has an upload capacity u_i, in addition to a download capacity d_i as before. These upload capacities may be much smaller than download capacities, because the traditional design assumes that the Internet traffic is primarily unidirectional. With user-generated content on the rise and P2P protocols heavily used, this assumption is no longer valid. In many cases, $\{u_i\}$ are quite large, at least for some of the peers. Peers with larger upload capacities can help distribute the files by sitting closer to the root of the multicast distribution tree in Figure 15.3.

Suppose these distribution trees can be perfectly designed to fully utilize all the upload capacities. Then we can say that for the total number of bits to be shared: NF, the total upload bandwidth available to the whole network is $u_s + \sum_{i=1}^{N} u_i$. So the time it takes is $NF/(u_s + \sum_{i=1}^{N} u_i)$ seconds.

Of course, the server still needs to send out each bit at least once to some peer, taking F/u_s seconds, and the slowest peer still needs to receive each bit, taking F/d_{min} seconds. Therefore, the time it takes to distribute the file throughout the network is now:

$$T = \max\left\{\frac{F}{u_s}, \frac{F}{d_{min}}, \frac{NF}{(u_s + \sum_{i=1}^{N} u_i)}\right\}. \tag{15.2}$$

In comparing (15.1) with (15.2), the key point is that, among the terms in

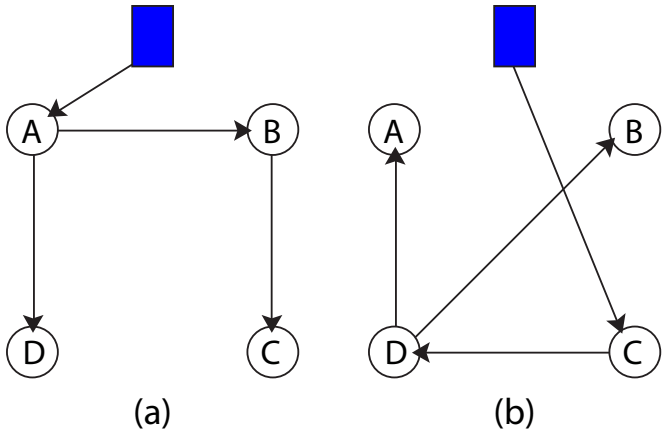


Figure 15.6 Two peering relationship trees. If only one of them is used, it is impossible to avoid wasting some nodes' uplink capacities: the tree on the left wastes C's and D's uplink capacities, while the tree on the right wastes A's and B's. This problem can be mitigated if we use both tree (a) and tree (b) for the same multicast session.

(15.2), only the third term has a numerator that scales with N, the number of peers, but that is divided by the summation of u_i over all N nodes, so T itself no longer scales with N. The network performance scales itself with the network size.

15.3.2 Constructing trees

The above back-of-the-envelope calculation assumes that all the peer upload capacities can be fully utilized. That is hard to do, and sometimes downright impossible, especially when you only have one distribution tree. As shown in Figure 15.6: in Tree 1, peers C and D's upload capacities are not used. In Tree 2, peers A and B's upload capacities are not used.

How about we use both trees at the same time? This is called the **multi-tree** construction of peering relationships. That helps, but it is still not clear what is the *best* way to construct all the trees needed to utilize the upload capacities. The basic idea, however, is clear: those peers with a lot of leftover upload capacities should be placed higher up in the newly constructed trees. Exactly how to do that involves solving a difficult combinatorial optimization problem; embedding even one tree in a general graph is hard, let alone multiple trees. That is the subject in Advanced Material.

But first, here is a special case that is easy to solve. Assume that the download capacities of peers are not the bottlenecks, *e.g.*, d_i's are large enough. Now we

want to prove

$$T = \max\left\{\frac{F}{u_s}, \frac{NF}{u_s + \sum_{i=1}^{N} u_i}\right\}.$$

To show this, we need to construct a multi-tree, *i.e.*, a set of multicast trees that collectively achieve the desired rates among N peers. Clearly, it suffices to show that the maximum broadcast rate of the multi-tree is

$$r_{max} = \min\left\{u_s, \frac{u_s + \sum_{i=1}^{N} u_i}{N}\right\}.$$

To see this, we reason through two cases.

Case 1: If $u_s \leq (u_s + \sum_{i=1}^{N} u_i)/N$, then the maximum broadcast rate of $r_{max} = u_s$ should be supported. The server upload capacity is too small. So we consider a multi-tree that consists of N trees, such that each i-th tree is two-hop, *e.g.*, the server takes peer i as its child and peer i takes the other $N-1$ peers as its children. Collectively these trees should deplete the upload capacity of the server. Furthermore, trees with more capable peers near the root should stream at a higher rate. Let each tree i carry a rate proportional to u_i:

$$r_i = \left(\frac{u_i}{\sum_{j=1}^{N} u_j}\right) u_s, \quad i = 1, \ldots, N,$$

as illustrated in Figure 15.7.

This rate assignment is possible because the total upload required for the server is within its capacity:

$$\sum_{i=1}^{N} r_i = u_s.$$

So is the total upload capacity required for peer i:

$$(N-1)r_i = (N-1)\frac{u_i}{\sum_{j=1}^{N} u_j} u_s \leq u_i,$$

since $Nu_s \leq u_s + \sum_{j=1}^{N} u_j$ by the assumption of this case, which implies that

$$Nu_s\left(\frac{u_i}{\sum_j u_j}\right) \leq \frac{u_s u_i}{\sum_j u_j} + u_i,$$

which further implies that

$$(N-1)\frac{u_i}{\sum_{j=1}^{N} u_j} u_s \leq u_i.$$

Now each peer receives a data stream directly from the server and also receives $N-1$ additional data streams from the other $N-1$ peers. So the aggregate broadcast rate at which *any* peer i receives is

$$r_{max} = r_i + \sum_{j \neq i} r_j = \sum_{i=1}^{N} r_i = u_s.$$

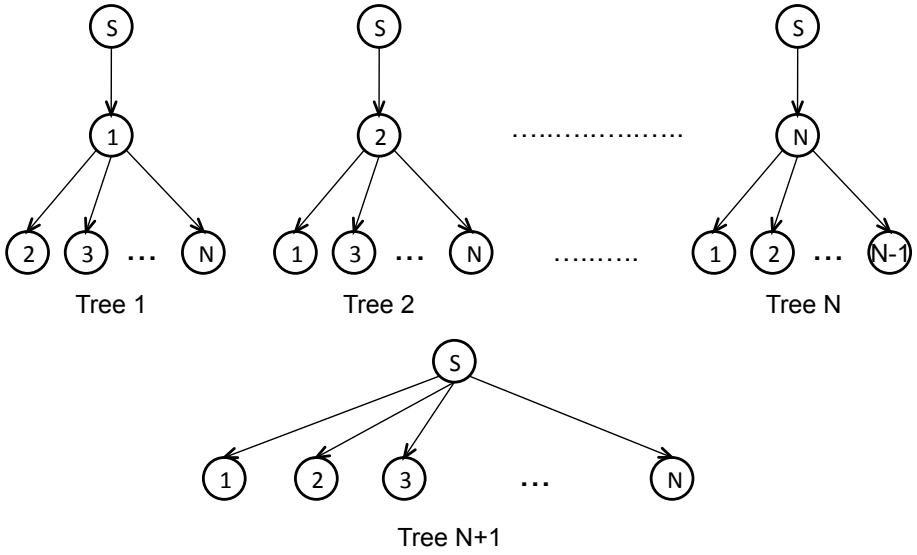


Figure 15.7 Multi-tree construction to maximize the multicast rate among N peers. In case 1, the server upload capacity is relatively small, and we use trees 1 to N. In case 2, the server upload capacity is sufficiently large, and we use trees 1 to $N+1$, with a different assignment of rates on the first N trees. In both cases, we only need shallow trees for this simple problem, and the multi-tree depth is 2.

Hence, it takes F/u_s time to transfer the whole file.

Case 2: If $u_s > (u_s + \sum_{i=1}^{N} u_i)/N$, then we need to show that the maximum broadcast rate of $r_{max} = (u_s + \sum_{i=1}^{N} u_i)/N$ can be supported. In this case, the server upload capacity is large enough for a different set of trees, including one tree where the server directly connects to all the peers, (a server-client tree), so as to fully utilize its upload capacity.

Consider a multi-tree that consists of $N+1$ trees, such that the i-th tree is two-hop and carries a rate of

$$r_i = \frac{u_i}{N-1},$$

i.e., equal distribution of each peer's uplink capacity among the other peers. And the $(N+1)$-th tree is one-hop directly from the server, which carries a rate of

$$r_{N+1} = \frac{u_s - \frac{\sum_{i=1}^{N} u_i}{N-1}}{N}.$$

This is the leftover uplink capacity from the server (after sustaining the first N trees) evenly distributed among all the N peers.

On i-th tree, for $i = 1, 2, \ldots, N$, the server has peer i as its child and peer i has the other $N-1$ peers as its children. In contrast, on the $(N+1)$-th tree, the server has all peers as its direct children. The tree construction is shown in Figure 15.7. This is possible because the total upload capacity required for peer

i is exactly

$$(N-1)r_i = u_i,$$

and the total upload capacity required for the server is exactly u_s:

$$\sum_{i=1}^{N} r_i + N \cdot r_{N+1} = \sum_{i=1}^{N} \frac{u_i}{N-1} + N\frac{u_s - \frac{\sum_{i=1}^{N} u_i}{N-1}}{N} = u_s.$$

Of course, the above two equalities are true by the way we design the rates on these $N+1$ trees.

Now each peer receives two data streams directly from the server and also receives $N-1$ additional data streams from the other $N-1$ peers. So the aggregate broadcast rate at which any peer i receives is

$$r_i + r_{N+1} + \sum_{j \neq i} r_j = \frac{u_i}{N-1} + \frac{u_s - \frac{\sum_{i=1}^{N} u_i}{N-1}}{N} + \sum_{j \neq i} \frac{u_j}{N-1} = \frac{u_s + \sum_{i=1}^{N} u_i}{N}.$$

Hence, it takes $NF/(u_s + \sum_{i=1}^{N} u_i)$ time to transfer the whole file.

Combining the two cases above produces our desired results.

15.4 Advanced Material

15.4.1 P2P streaming capacity

In the example above, we have assumed many ideal conditions in the above calculation. Peering relationship construction in a general, large-scale network is much more challenging. Structured P2P overlay carefully designs the peering relationships based on criteria such as throughput, latency, and robustness. Some of these topologies are inspired by what we saw in Chapters 9 and 10, *e.g.*, the Watts Strogatz graph. This leads us to a graph-theoretic optimization problem.

Consider the following problem: given a directed graph with a source node and a set of receiver nodes, how do we embed a set of trees spanning the receivers and determine the amount of flow in each tree, such that the sum of flows over these trees is maximized? Constraints of this problem include an upper bound on the amount of flow from each node to its children, the maximum degree of a node allowed in each tree, and other topological constraints on the given graph. This is the general problem of P2P streaming capacity computation.

What is the P2P streaming capacity and what is an optimal peering configuration to achieve the capacity? Here, "capacity" is defined as the largest rate that can be achieved for all receivers in a multicast session, with a given source, a set of receivers, and possibly a set of helper (non-receiver relay) nodes.

There are in fact at least sixteen formulations of this question, depending on whether there is a single P2P session or multiple concurrent sessions, whether the given topology is a full mesh graph or an arbitrary graph, whether the number

of peers a node can have is bounded or not, and whether there are helper nodes or not. In each formulation, computing P2P streaming capacity requires (1) the determination of how to embed an optimal set of multicast trees, and (2) what should the rate be in each tree.

We outline a family of algorithms that can compute or approximate the P2P streaming capacity and the associated multicast trees. In general this problem is intractable; it is difficult to find polynomial time algorithms that solve the problem exactly. The algorithm we summarize below can solve, in polynomial time, seven of the sixteen formulations arbitrarily accurately, and eight other formulations to some constant factor approximations.

We will be reformulating the optimization to turn the combinatorial problems into linear programs with an exponential number of variables. The algorithms combine a primal-dual update outer loop (similar to what we saw in Chapter 14) with an inner loop of "smallest price tree construction" (similar to what we just saw in the last section), driven by the update of Lagrange dual variables in the outer loop. Graph-theoretic solutions to various cases of the smallest price tree problem can then be leveraged, although that is beyond the scope here. Our focus will be on formulating this problem of embedding multiple trees in a graph, and the generalization of congestion control's primal-dual solution to a more complicated case where each inner loop is not just a simple rate or price update equation.

Consider a multicast streaming session. It originates from one source, and is distributed to a given set of receivers. For example, in video conferencing, there are multiple participants; each may initiate a session and distribute her video to others, and each participant can subscribe to others' videos. In an IPTV network, different channels may originate from different servers, with different sets of subscribers. Denote by s the original source, by R the set of receivers, and by H the set of helpers. We say that the session has *rate* r bps if all the receivers in this session receive the streaming packets at a rate of r or above.

As illustrated in Figure 15.8, we consider a P2P network as a graph $G = (V, E)$, where each node $v \in V$ represents a peer, and each edge $e = (u, v) \in E$ represents a *neighboring* relationship between vertices (u, v). A peer may be the source, or a receiver, or a helper that serves only as a relay. A helper does not need to get all packets but only the ones that it relays.

- This graph is an overlay on top of the given underlay graph representing the *physical connections* among users. The underlay graph may constrain the design of *peering* relationships: if two nodes u and v are not physically connected by some path of reasonable length, they cannot be neighbors in the overlay graph, and do not stand a chance to become peers either.
- Neighbors do not have to become peers. Neighboring relationship is given, while peering relationship is to be designed as part of the P2P streaming capacity computation.
- The graph G may or may not be full mesh. Typically, full mesh is only possible

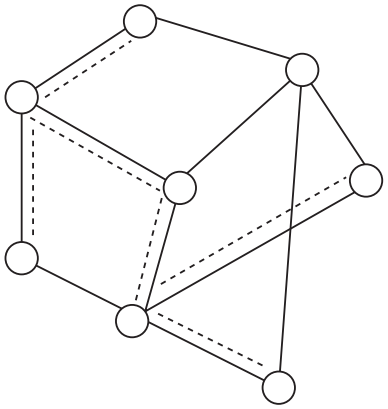


Figure 15.8 An overlay network where each node represents a peer, and each link (in solid line) represents a neighboring relationship. The job of computing P2P streaming capacity is to construct multiple trees so that their rates add up to the largest possible for a multicast session. A particular tree is shown in dotted lines, which represents peering relationships designed under the constraint of the given neighboring relationships.

in a small network with a small number of users, while a large network has a sparse topology.

Consider one chunck of a given stream. It starts from the source s, and traverses over all nodes in R, and some nodes in H. The traversed paths form a **Steiner tree** in the overlay graph $G(V, E)$, a tree that spans all the nodes in only a given *subset* of V. In this case, the subset is R. You can view this as a generalization of the spanning tree homework problem in Chapter 13.

Different packets may traverse different trees. We call the superposition of all the trees belonging to the same session a *multi-tree*. For each tree t, denote y_t as the sub-stream rate supported by this tree. Here, t is not a time index, but a complicated data structure: a tree.

The use of a P2P protocol imposes certain constraints on sub-trees. The most frequently encountered one is the node degree constraint. For example, in BitTorrent, although a node may have 50 neighbors in G, it can upload to at most 5 of them as peers. This gives an outgoing degree bound for each node and constrains the construction of the trees. We can examine the case of degree bound for each node *per* tree. Let $m_{v,t}$ be the number of outgoing edges of node v in tree t, and the bound be M_v. This gives an inequality constraint on allowed trees: $m_{v,t} \leq M_v, \forall t$. The more general case of degree bound for each node across *all* the trees is even harder.

We denote by T the set of all allowed sub-trees. Obviously, the multicast rate

r is the sum of all the rates on the trees:

$$r = \sum_{t \in T} y_t. \tag{15.3}$$

We will make the following assumptions for streaming applications: there is a static set of stationary users and all desired chunks of packets are available at each node. We also assume that data rate bottlenecks only appear at user uplinks. This assumption is widely adopted because in today's Internet access links are the bottlenecks rather than backbone links, and uplink capacity is several times smaller than downlink capacity in typical access networks. Denote by C_v the uplink capacity of node v. We have the following bound on the total uplink rate U_v for each node v:

$$U_v = \sum_{t \in T} m_{v,t} y_t \leq C_v.$$

A rate is called *achievable* if there is a multi-tree in which all trees satisfy the topology constraint ($t \in T$) and transmission rates satisfy the uplink capacity constraint ($U_v \leq C_v$). We define **P2P streaming capacity** as the largest achievable rate.

15.4.2 A combinatorial optimization

Now we can represent the single-session streaming capacity problem as the following optimization problem:

$$\begin{array}{ll} \text{maximize} & \sum_{t \in T} y_t \\ \text{subject to} & \sum_{t \in T} m_{v,t} y_t \leq C_v \ \ \forall v \in V \\ \text{variables} & y_t \geq 0, \ \ \forall t \in T. \end{array}$$

This representation of the problem is deceptively simple: the difficulty lies in searching through all combinations of trees t in the set of allowed trees T. For those trees not selected in the optimizer, their rates y_t are simply 0. Compared to all the other optimization problems in previous chapters, this one is much more difficult. It has a combinatorial component and is not convex optimization. It has coupling across trees and cannot be readily decomposed.

Still we can try to derive the Lagrange dual problem. From Lagrange duality, solving the above problem is equivalent to solving its dual problem, and an optimizer of the dual problem readily leads to an optimizer of the primal algorithm. The dual problem associates a non-negative variable p_v interpreted as the price with each node. The Lagrange dual problem turns out to be as follows:

$$\begin{array}{ll} \text{minimize} & \sum_{v \in V} C_v p_v \\ \text{subject to} & \sum_{v \in V} m_{v,t} p_v \geq 1, \ \ \forall t \in T \\ \text{variables} & p_v \geq 0, \ \ \forall v \in V. \end{array}$$

We can interpret the dual problem similar to the dual congestion control problem in Chapter 14: p_v is the (per unit flow) price for any edge outgoing from v.

If node v uploads with full capacity, the incurred cost is $p_v C_v$. There are $m_{v,t}$ connections outgoing from node v in tree t, and thus the *total tree price* for tree t is simply $\sum_{v \in V} m_{v,t} p_v$. Therefore, the dual problem is to minimize the total full capacity tree cost given the tree price is at least 1, and the minimization is over all possible price vectors $\mathbf{p}$. This is a generalization of link (and path) prices used in solving the Network Utility Maximization problem for distributed capacity allocation and TCP congestion control.

In general, the number of trees we need to search when computing the right multi-tree grows exponentially with the size of the network. This dimensionality increase is the consequence of turning a difficult graph-theoretic, discrete problem into a continuous optimization problem. Hence, the primal problem has too many variables and its dual problem has too many constraints, neither of which is suitable for direct solution. However, the above representations turn out to be very useful to allow a primal-dual update outer loop, which converts the combinatorial problem of multi-tree construction into a much simpler problem of "smallest price tree" construction.

15.4.3 Garg-Konemann iterations

We adopt the **Garg-Konemann** technique, where flows are augmented in the primal solution and dual variables are updated iteratively. The algorithm constructs peering multi-trees that achieve an objective function value within $(1+\epsilon)$-factor of the optimum.

For a given tree t and prices $\mathbf{p}$, let $Q(t, \mathbf{p})$ denote the left-hand-side of constraint (15.4.2), which we call the price of tree t. A set of prices $\mathbf{p}$ is a feasible solution for the Lagrange dual problem if and only if

$$\min_{t \in T} Q(t, \mathbf{p}) \geq 1.$$

The algorithm works as follows. Start with initial weights $p_v = \frac{\delta}{C_v}$ for all $v \in V$. Parameter δ depends on the accuracy target ϵ. Repeat the following steps until the dual objective function value becomes greater than 1:

1. Compute a tree t^* for which $Q(t, \mathbf{p})$ is minimum. We call t^* a **smallest price tree**.
2. Send the maximum flow on this tree t^* such that the uplink capacity of at least one internal node (neither the root nor the leaf nodes of the tree) is saturated. Let $I(t)$ be the set of internal nodes in tree t. The flow sent on this tree can only be as large as

$$y = \min_{v \in I(t^*)} \frac{C_v}{m_{v,t^*}}. \tag{15.4}$$

3. Update the prices p_v as

$$p_v \leftarrow p_v \left(1 + \frac{\beta m_{v,t^*} y}{C_v}\right), \ \forall v \in I(t^*),$$

where the stepsize β depends on ϵ.

4. Increment the flow Y sent so far by y.

The optimality gap can be estimated by computing the ratio of the primal and dual objective function values in each step of the iteration above, which can be terminated after the desired proximity to optimality is achieved. When the iteration terminates, primal capacity constraints on each uplink may be violated, because we were working with the original (and not the residual) uplink capacities at each stage. To remedy this, we can scale down the flows uniformly so that uplink capacity constraints are satisfied.

For any given target accuracy $\epsilon > 0$, the algorithm above computes a solution with an objective function value within $(1 + \epsilon)$-factor of the optimum, for appropriately chosen algorithmic parameters. It runs in polynomial time in the network size and $\frac{1}{\beta}$:

$$O\left(\frac{N \log N}{\beta^2} T_{spt}\right),$$

where N is the number of peers, and T_{spt} is the time to compute a smallest price tree.

The core issue now lies with the inner loop of smallest price tree computation: can this be accomplished in polynomial time for a given price vector? This graph-theoretic problem is much more tractable than the original problem of searching for a multi-tree that maximizes the achievable rate. However, when the given graph G is not full mesh, or when there are degree bounds on nodes in each tree, or when there are helper nodes, computing a smallest price tree is still difficult.

Moreover, how to approach the P2P streaming capacity with distributed and low complexity algorithms is another challenge, even when the capacity can be easily computed with a centralized optimization of peering multi-trees. Part of this challenge is the control signaling and part of the solution is spatial hierarchy.

Control signaling in P2P can rely on either a centralized tracker, or on broadcasting, the so-called **query flooding**, so that each peer has a local copy of the topology and network states. In between these two options is hierarchical overlay. It turns out a two-level hierarchy is often used in both theory and practice, and for both control signaling and the actual data sharing. For example, research papers on P2P have proposed a hierarchical architecture with a separation of the peers into groups of peers clustered by geographic proximity. Across the clusters, super peers, *e.g.*, servers or peers with high uplink capacities, form a shallow tree and communicate with one another. These super peers also serve as source nodes for peers inside each cluster and follow a densely connected mesh network. Since the number of super peers is much smaller than that of peers, careful optimization can be performed to ensure the core achieves close-to-optimal performance. Inside each cluster, peers are organized into streaming trees rooted at super peers that are close to them. These trees are usually "shallow": they take the form of

one-hop or two-hop multicast trees. They also follow the intuition that peers with low capacity should be placed close to leaf nodes of a multicast tree.

Further Reading

There have been many measurement, modeling, and design papers written about P2P systems since the early 2000s.

1. The founder of BitTorrent, Bram Cohen, wrote the following widely cited paper explaining some of the design choices in BitTorrent's incentive mechanisms:
[Coh03] B. Cohen, "Incentives build robustness in Bit Torrent," *Proceedings of Workshop on Economics of Peer-to-Peer Systems*, 2003.

2. The following paper provided a comprehensive survey of P2P-based IP telephony systems, including reverse engineering some of the Skype details:
[SS06] A. B. Salman and H. Schulzrinne, "An analysis of the Skype peer-to-peer Internet telephony protocol," *Proceedings of IEEE Infocom*, 2006.

3. The following is a seminal paper on modeling P2P streaming applications:
[KLR07] R. Kumar, Y. Liu, and K. Ross, "Stochastic fluid theory for P2P streaming systems," *Proceedings of IEEE Infocom*, 2007.

4. A large-scale peering topology measurement project can be found at:
[WLZ08] C. Wu, B. Li, and S. Zhao, "Exploring large-scale peer-to-peer live streaming topologies," *ACM Transactions on Multimedia*, vol. 4, no. 3, 2008.

5. The approach of P2P streaming capacity computation was developed in the following paper:
[Sen+11] S. Sengupta, S. Liu, M. Chen, M. Chiang, J. Li, and P. A. Chou, "P2P streaming capacity," *IEEE Transactions on Information Theory*, vol. 57, no. 8, pp. 5072-5087, 2011.

Problems

15.1 *Embedding trees* $\star$

Consider a network of one server and $N = 3$ peers, given that (1) $u_s = 2$, $u_1 = 3$, $u_2 = 2$, $u_1 = 1$ (all in Mbps), (2) all nodes have unlimited download capacity, and (3) each peer in a multicast tree can upload to any number of peers.

(a) Find the maximum multicast rate r_{max}.

(b) Draw a multi-tree that achieves this maximum.

(c) Find the rate r_i of the i-th multicast tree, for all $i = 1, \ldots, T$, where T is the number of multicast trees from part (a).

Now consider a network with one server and $N = 3$ peers, with $u_s = 3$, $u_1 = 3$, $u_2 = 2$, $u_1 = 1$ (in Mbps). We now constrain that each peer in a multicast tree can only upload to at most one peer, so as to limit the overhead in maintaining the states of the peers.

(d) Draw the resulting multi-tree that achieves the maximum multicast rate.

(e) Compute the per-tree rates r_i.

15.2 *Delay components in P2P streaming* ⋆⋆

Consider a P2P streaming tree that consists of N nodes. Suppose the streaming tree is balanced (*e.g.*, the depths of leaf nodes never differ by more than 1), and the tree fanout is M (*e.g.*, every internal node has M children). Every node has the same upload capacity C, and every link connecting two nodes has the same latency L. Let B be the chunk size used in the streaming system.

(a) Suppose the streaming delay consists of two components: node (transmission) delay, and link (propagation) delay, what is the maximum streaming delay over all nodes in this tree?

(b) Let $N = 1000$, $C = 1$Mbps, $B = 20$KB, $L = 50$ms, and $M = 5$. Which delay component is more significant?

(c) Suppose the *less* significant delay component can be ignored, what is the optimal fanout M to choose in order to minimize delay?

15.3 *Stable marriage matching* ⋆⋆⋆

The **stable marriage matching** problem is a long-studied problem and has many applications, from matching medical students to hospitals to analyzing voting systems and auctions. Figure 15.9 illustrates a matching in a bipartite graph.

Suppose a set of partners can be split into two equal-sized subsets A and B. Given each element $a \in A$ has a strict ranking of potential partners $b \in B$ (and vice versa). The ranking can be determined in terms of bandwidth, latency etc. A stable matching assigns each $a \in A$ to some $b \in B$ and each $b \in B$ to some $a \in A$, such that there does not exist a pair $(a, b) \in A \times B$ with a preferring b to the $b' \in B$ that a is currently assigned to, and b preferring a to the $a' \in A$ that b is currently assigned to.

A standard solution to solve the stable marriage matching problem is the **Gale-Shapley algorithm**. It follows a simple, iterative procedure. A man (a

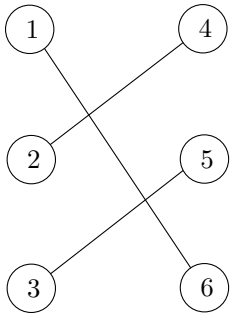


Figure 15.9 An example of a matching on a bipartite graph, where nodes 1, 2 and 3 are assigned to nodes 6, 4 and 5, respectively.

node in set A), or a woman (a node in set B), can be unengaged, engaged, or married. At each round, an unengaged man proposes to the most preferred woman among those that he has not yet proposed in previous rounds, and each woman chooses to engage to the suitor whom she prefers most and rejects the rest. As the iteration continues, engagements may be broken. When there is no more unengaged man, the iteration stops, and the engaged pairs at that point are married.

(a) Run the algorithm to find a stable matching between the two sets $A = \{a, b, c\}$ and $B = \{d, e, f\}$ with the following rankings:

$$
\begin{aligned}
a &: d > e > f, \\
b &: d > e > f, \\
c &: d > f > e, \\
d &: c > b > a, \\
e &: c > a > b, \\
f &: a > c > b.
\end{aligned}
$$

(b) Argue that at the conclusion of the algorithm, one of the stable marriage matchings must have been found.

15.4 *BitTorrent as a game* ⋆⋆⋆

BitTorrent's upload-incentive mechanism can be analyzed with game theory. Let there be N peers indexed as $1, 2, \ldots N$, and c_i, u_i and d_i be peer i's upload capacity, upload speed, and download speed respectively (all in Mbps). The speeds u_i and d_i can vary with time). Furthermore, we assume peers to have unlimited download capacities.

Each peer i can directly control its u_i (constrained by $u_i \leq c_i$) but not its d_i, and its aim is to (1) maximize d_i, (2) while minimizing u_i. There is a tradeoff between the two objectives: if peer i makes u_i small, other peers will realize that

peer i is selfish, and refuse to upload to it, resulting in a small d_i. BitTorrent's peer selection mechanism aims to enforce this tradeoff so as to make u_i large and to encourage uploads.

Now consider the following set of rules, which are a simplified version of BitTorrent's peer selection mechanism:

(i) Peers take turns to update u_i in the ascending order of i and then wrap around.

(ii) After peer i updates its u_i, all other peers see this change, and choose to upload to the top n_u (an integral parameter) peers j in terms of u_j values (and break ties by choosing randomly). The upload speeds are shared evenly among the n_u peers.

(iii) Peer i chooses u_i by anticipating the d_i it receives according to rule (ii): u_i is chosen to maximize the expected d_i. If multiple u_i values result in the same d_i, choose the smallest one, plus a small constant ϵ.

Here comes your two tasks in this homework problem. Let there be $N = 4$ peers with each peer uploading to $n_u = 2$ peers, and set $\epsilon = 0.1$.

(a) Suppose $c_1 = 1$, $c_2 = c_3 = c_4 = 2$. Initially it is peer 1's turn to update u_1 with $u_2 = u_3 = u_4 = 1.1$. Then we have the following line of reasoning:

(1) Regardless of the value of u_1, no peer will upload to peer 1 ($d_1 = 0$) because $0 \le u_1 \le c_1 < u_2, u_3, u_4$, so peer 1 sets $u_1 = 0 + \epsilon = 0.1$ by rule (iii);

(2) In the next time slot, it is peer 2's turn to update u_2, which becomes $u_1 + \epsilon$ because u_2 needs to be the third largest, *i.e.*, greater than u_1, so that peers 3 and 4 will upload to it.

Continue this line of reasoning to show that the u_i values never converge to fixed values.

(b) Suppose $c_1 = c_2 = c_3 = c_4 = 2$. Show that setting $u_1 = u_2 = u_3 = u_4 = 2$ constitutes a Nash equilibrium.

(Hint: show that if it is peer i's turn to set u_i, setting u_i to be any value other than $c_i = 2$ will not improve d_i.)

(For more detail, see D. Qiu and R. Srikant, "Modeling and performance analysis of BitTorrent-like peer-to-peer networks," *Proc. ACM Sigcomm*, 2004.)

15.5 *Private BitTorrent games* ⋆⋆⋆

We have been discussing the public BitTorrent. There are also many private torrents that create their own rules of rewarding seeders and encouraging uploads beyond the simple tit-for-tat in BitTorrent. More than 800 such private communities were found in 2009. In this homework problem, we explore one possible rule of proportional reward, again through the modeling language of game theory and utility functions.

Let d_i and u_i be the actual download and upload volumns, meausred in bytes,

for peer i in a fixed population of peers. A standard incentive mechanism in private BitTorrents is the following ratio incentive:

$$d_i \leq f(u_i),$$

i.e., the download volumn cannot be bigger than some function of the upload volumn, and an affine parameterization of this function is

$$f(u_i) = \frac{u_i}{\theta} + \Delta.$$

Here, similar to leacky bucket admission control that we will see in Chapter 17, θ is the upload-download ratio targeted, and Δ is the slack: the amount of data a peer can download outside of the ratio rule.

Each peer's utility function V_i can be parameterized by

$$V_i(d_i, u_i) = B(d_i) - C(u_i) + \beta(f(u_i) - d_i),$$

as long as $f(u_i) - d_i \geq 0$. It becomes $-\infty$ (it will be evicted out of the community) otherwise. Here, B and C are some utility and cost functions.

Suppose the strategy for peer i is the two-tuple of target download and target upload (per unit time): (δ_i, σ_i), over the strategy spaces of $\delta_i \in [0, D]$ and $\sigma_i \in [0, U]$.

(a) How can we express the actual upload and download amounts $\{u_i, d_i\}$ as functions of the strategies $\{\delta_i, \sigma_i\}$ chosen by all the peers? This would be very difficult, but if we make an assumption that all the downloads add up to be exactly the same as the sum of all the uploads, then there is a closed-form answer. What is that answer?

(b) Now we want the Nash equilibrium to be efficient: each peer chooses the target download and upload to be just D and U. Can you prove that $(\delta_i, \sigma_i) = (D, U)$, $\forall i$, is indeed a Nash equilibrium if $f(u) > u$ and $f'(u) > C'(u)/\beta$?

(c) As a corollary to part (b), show that if the ratio incentive parameters (θ, Δ) are such that $u/\theta + \Delta > u$ and $\beta > \theta C'(u)$ for all $u \in [0, U]$, then using ratio incentive implies that $(\delta_i, \sigma_i) = (D, U)$, $\forall i$, is the unique Nash equilibrium.

(For more details, see Z. Liu, P. Dhungel, D. Wu, C. Zhang, and K. W. Ross, "Understanding and improving incentives in private P2P communities," *Proc. IEEE International Conference on Distributed Computing Systems*, 2010.)

16 What's inside the cloud of iCloud?

16.1 A Short Answer

In June 2011, Apple announced its iCloud service. Part of the eye-catching features is its digital rights management of music content. The other part is its ability to essentially carry your entire computer hard drive with you anywhere and stream music to any device.

Cloud is more than just storage. For example, in the same month, Google introduced ChromeBook, a "cloud laptop" that is basically just a web browser with Internet connectivity, and all the processing, storage, and software are somewhere in Google servers that you access remotely.

These new services and electronics intensify the trends that started with web-based emails (*e.g.*, Gmail), software (*e.g.*, Microsoft Office 365), and documents (*e.g.*, Google Docs and Dropbox), where consumers use the network as their computers, the ultimate version of online computing.

In the enterprise market, many application providers and corporations have also shifted to cloud services, running their applications and software in rented and shared resources in **data centers**, rather than building their own server farms. Data centers are facilities hosting many servers and connecting them via many switches. Large data centers today can typically be over 300,000 square feet, house half a million servers, and cost hundreds of millions of dollars to build.

There are three major cloud providers: Amazon's EC2, Microsoft's Azure, and Google's AppEngine. A pioneering player in cloud services is actually Amazon, even though to most consumers Amazon stands for an online retail store. In Amazon's S3 cloud service today, you can rent slices of the cloud for $0.12 per GB per month and $0.10 per GB of data transfer.

For many years, it has been a goal in the computing and networking industries that one day users could readily rent resources inside the network (the "cloud" on a typical network illustration), in a way that makes economic sense for all the parties involved. That day is today. Thanks to both technological advances and new business cases, cloud services are taking off and evolving fast.

Many features of cloud services are not new, some are in fact decades-old. Several related terms have been used in the media somewhat confusingly too: cloud computing, utility computing, clustered computing, software as a service,

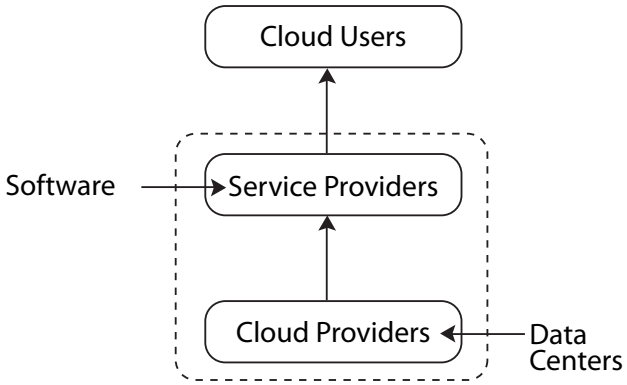


Figure 16.1 Three segments of the cloud service industry. Cloud providers operate data centers that house the hardware, including interconnected processors, storage and network capacity. Service providers run cloud services through their software. Cloud users include both consumer and enterprise customers for cloud services provided collectively by the service providers and cloud providers.

etc. To clarify the terminology, we refer to the graph in Figure 16.1. There are three key components of the "food chain:"

Cloud providers build and manage the hardware platform, consisting of computing resources (servers), networking resources (switches), and storage resources (memory devices) organized inside data centers. There is a network within each data center where the nodes are servers and switches, and each data center in turn becomes a node in the entire Internet.

Service providers offer software and applications that run in data centers, and interface with users. For example, an iPad application developer may use the computing and storage resources in Amazon's EC2 cloud to deliver its services. Sometimes, the service provider is the same as the cloud provider. For example, the iCloud music storage and streaming service from Apple runs on Apple's own data centers.

Cloud users are consumers and enterprises that use services running in data centers. Users can get the content they want (*e.g.*, documents, books, music, video) or software (*e.g.*, Office software, an iPhone application, or in the cloud laptop's case, pretty much every software you need) *on demand*, anytime, anywhere, and on any device with an Internet connection.

16.1.1 What features uniquely define cloud services?

To make the overall cloud service food chain work, we need all of the following ingredients:

1. *Large-scale computing and storage systems*, often leveraging *virtualization* techniques in sharing a given hardware resource among many processes as if they each had a slice of a dedicated and isolated resource.
2. *Networking* within a data center, across the data centers, and to the end users (often with a wireless hop like WiFi or 4G). This networking dimension naturally will be the main focus of this chapter, especially networking within a data center.
3. *Software* that provides Graphic User Interface, Digital Rights Management, security and privacy, billing and charging, etc.

If I open up my home desktop's CPU and hard drive to renters, does that constitute a cloud service? Probably not. So, what are the defining characteristics of a cloud service? The keyword is *on demand*, along two dimensions: time and scale.

- On demand in timing: a cloud service allows its users to change their requests of resources on a short notice, and possibly only for a short period of time.
- On demand in scale: a cloud service allows its users to start at a very small minimum level of resource request (*e.g.*, 1.7 GB of RAM and 160GB of memory on Amazon's EC2 today), and yet can go to really large scale (*e.g.*, Target, the second largest retail store chain in the US, runs its web and inventory control in a rented cloud).

16.1.2 Why do people hesitate with cloud services?

Cloud services face many challenges, even though they are increasingly outweighed by the benefits. Let us briefly bring them up before moving on.

Similar to the pros-cons comparison between packet switching and circuit switching, once you are in a shared facility, performance guarantee is compromised and so are security and privacy. If you ride a bus instead of a taxi, you pay less but you might not have a seat and you will be seen by the fellow bus riders. That is the price you pay to enjoy the benefits of cloud services. There are many technologies that mitigate cloud's downsides for various market segments, but riding a bus will never be exactly the same as taking a taxi.

As illustrated by the Amazon cloud outage in April 2011, availability of service in the first place is one of the top performance concerns. Main root causes for unavailability are network misconfigurations, firmware bugs, and faulty components. The best way to enhance availability is *redundancy*: spread your traffic across multiple cloud providers (assuming it is easy enough to split and merge this traffic), and across different reliability zones in each of the providers.

16.1.3 Why do people like cloud services?

Why does it make sense to provide and to use cloud services? The answers are similar to many other wholesale rental businesses, such as libraries and rental car

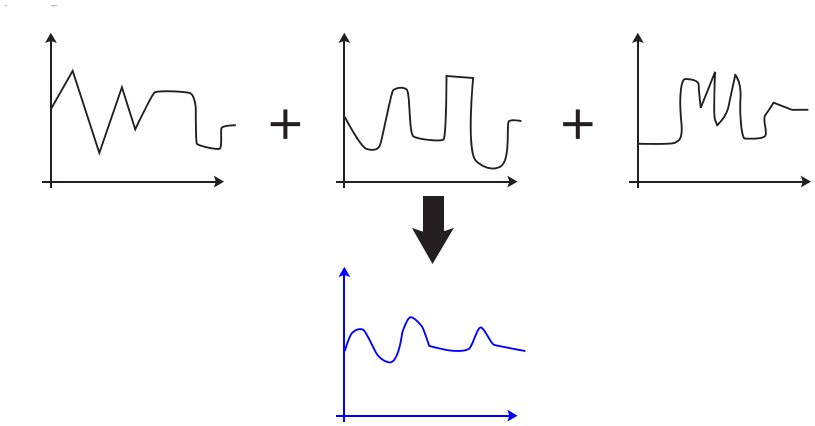


Figure 16.2 Statistical multiplexing smoothes out burstiness of individual users. Suppose there are three users with their transmission rates over time charted above. Their aggregate transmission rate, shown in the lower graph, is much smoother. Cloud providers benefit from such burstiness reduction, as long as the users' demands do not peak together.

companies. We summarize the arguments below and will go through an example in a homework problem.

To the cloud users, the key benefit is *resource pooling*. The cost of building and provisioning resources is now shared by many other users. This is called the "CapEx to OpEx conversion": instead of spending money in capital expenditure to build out dedicated facilities, a user pays rent as part of its operational expenditure to share facilities. This is similar to going from circuit switching to packet switching in the design of the Internet. The risk of miscalculating resource need shifts to cloud providers, a significant advantage if the resource demand varies a lot or is just hard to predict. But why would cloud *providers* be interested?

A main reason is the *economy-of-scale* advantages on both the supply and demand sides of the business. On the supply side, a cloud provider can procure the servers, switches, labor, land, and electricity at significantly discounted price because of its large scale and bargaining power. Even when compared to a medium size data center with thousands of servers, a large scale data center with a hundred thousand servers can often achieve a factor 5-7 advantage in cost per GB of data stored or processed.

On the demand side, scale helps again, through *statistical multiplexing*. Fluctuations of demand for each user are absorbed into a large pool of users, as shown in Figure 16.2. This is the same principle behind ISPs oversubscribing at each aggregation point of their access networks: aggregating many bursty users reduces the burstiness. Of course, the overall pool may still exhibit time-of-day peak-valley patterns. The *average* utilization of servers in a data center is often

below 20% today. These peak-valley patterns can be further smoothed out by time dependent pricing as discussed in Chapter 12.

Cloud is all about *scale*. Today's large scale data centers are indeed huge, so big that electricity and cooling costs sometimes represent more than half of the total cost. If iPhone is one of the smallest computers we use, each data center is one of the largest. We have made an important assumption, that it is actually *feasible* to scale up a data center. Otherwise, we would have to truncate all the benefits associated with scaling up. But as we saw in Chapter 10, scale can also be a *disadvantage* when each (reasonably priced) network element can only have a small number of high performance ports. Unless you have the right network topology, building a 100,000-server data center can be much more expensive, in unit price of capacity (or bandwidth in this field's common terminology), than building a 10,000-server data center. This echoes Chapter 10: (high throughput) connectivity per node does *not* scale up beyond a certain point in either technology or human networks. Yet we want (high throughput) connectivity for the whole network to keep scaling up. That is the subject of the next section: how to achieve the advantages of scale for a network without suffering the limitation of scale per node.

16.2 A Long Answer

16.2.1 Building a big network from small switches

We need a network within a data center. Many applications hosted in a data center require transfer of data and control packets across the servers at different locations in that big building. A natural, but inferior solution, is to build a tree like Figure 16.3(a), where the leaf nodes are the servers, and the other nodes are the routers. The low level links are often 1 Gbps Ethernet, and upper level ones 10 Gbps Ethernet links. The top-of-the-tree switches are big ones, each with many 10 Gbps links. It is expensive to build these big switches. As the number of leaf nodes increases to 100,000 and more, it becomes technologically impossible to build the root switch. A high end switch today can only support 1280 servers.

So we need to start *over-subscribing* as we climb up the tree. Sometimes the **oversubscription ratio** runs as high as 1:200 in a large data center. What if all the leaf node servers want to fully utilize their port bandwidths to communicate with other servers at the *same* time? Then you have a 200-factor congestion. The whole point of resource pooling is defeated as we fragment the resources: idle servers cannot be utilized because the capacity between them cannot be used in an oversubscribed tree. No matter how you cut it, a tree is not scalable. Many trees would have been better, as in P2P in Chapter 15, but we cannot swap the leaf nodes upstream in multi-trees here, because the leaf nodes are servers and the upstream nodes are switches.

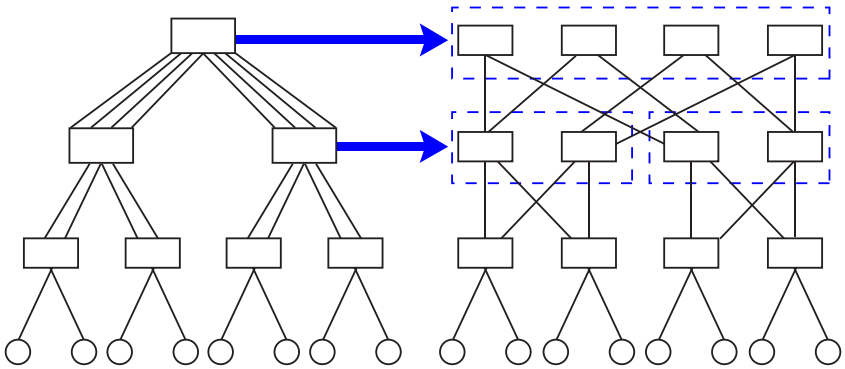


Figure 16.3 From tree to fat tree. (a) shows a tree supporting 8 servers (the leaf nodes represented by circles, with 4 switches with two inputs and two outputs, 2 switches with four inputs and four outputs, and 1 switch with eight inputs and eight outputs. It is expensive to build the larger switches. When the root switch is too big, it becomes impossible to build one, and oversubscription has to take place as we go up the tree. In contrast, in (b), 2 small switches (each with two inputs and two outputs) collectively serve the role of the 4 by 4 switch, and 4 small switches collectively serve the role of the 8 by 8 switch.

Is it still possible to build a large network with small switches, just like building a reliable network out of unreliable components?

The answer is yes, if you are smart enough with the network topology, and go from a tree to a **multi-stage switched network**. Instead of building a large scale network by scaling *up* the number of ports per router and hitting the economic and technology ceilings, we scale *out* the topology and can do that as much as we want. We use many small switches to make a large switch, with the same number of links per switch at each level.

This branch of networking has long been studied in the old days of circuit switching, when it became impossible to build a large enough single switch to handle all phone calls. Then **interconnection networks** were studied for multicore processors and parallel computation. Now the study of interconnection networks has revived in the context of data center networking for cloud services. The key message here is that *connectivity itself is a resource* that we can build and need to build carefully.

There are many ways to quantify how good an interconnection topology is. We focus on throughput rather than latency.

- The worst case pairwise end-to-end capacity (*e.g.*, from one leaf node to another in a tree) is one possibility.
- **Bisection bandwidth** is another: similar to social graph partitioning in Chapter 8, the capacity on all the links between two equal-sized halves of the network is called a **bisection cut**, and the worst case of that over

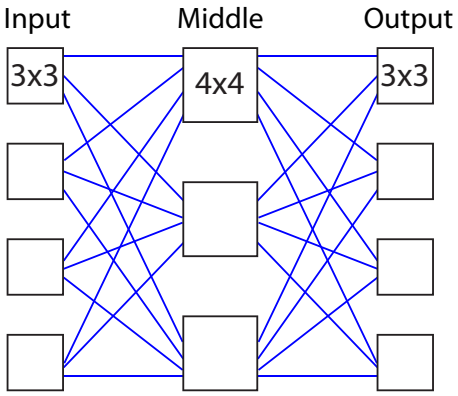


Figure 16.4 An example of a Clos network. This is a 3-stage, (3, 3, 4) Clos network, supporting 12 inputs and 12 outputs using only 3 by 3 and 4 by 4 switches. There are 4 of these 3 by 3 switches on the input stage, 4 of them on the output stage, and 3 of these 4 by 4 switches in the middle stage. Each input switch is connected to each of the middle switches. Each output switch is also connected to each of the middle switches.

all possible combinations of halves of the network is called bisection bandwidth.

- A third metric is a classic one often used in circuit switching: a network is called **nonblocking** if any pair of (unused) input and (unused) output can be connected as each traffic session (a switching request) arrives and is switched. It is called **rearrangeably nonblocking** if some existing pairs' connection needs to be rearranged in order to achieve the nonblocking property.

In 1953, the most famous interconnection network topology, called the **Clos network**, was invented. An example is shown in Figure 16.4. The general definition of a Clos network is as follows. Each Clos network is specified by three integers: (n, m, r). Each input switch is $n \times m$, and there are r of them. Symmetrically, each output switch is $m \times n$, and there are also r of them. The middle stage switches of course must be $r \times r$, and there are m of them. Each of the input and output switches is connected to each of the middle stage switches. This is a rich connectivity, using only small switches to support rn input-output pairs of ports, as illustrated in Figure 16.4.

Assuming a centralized controller is picking the switching patterns, it can be readily seen that if $m \geq 2n - 1$, then a Clos network is nonblocking. Consider a new switching request arriving at a particular input port of an input switch A, to be switched to an output port of an output switch B. In the worst case, each of the other $n - 1$ ports on A is already occupied, each of the other $n - 1$

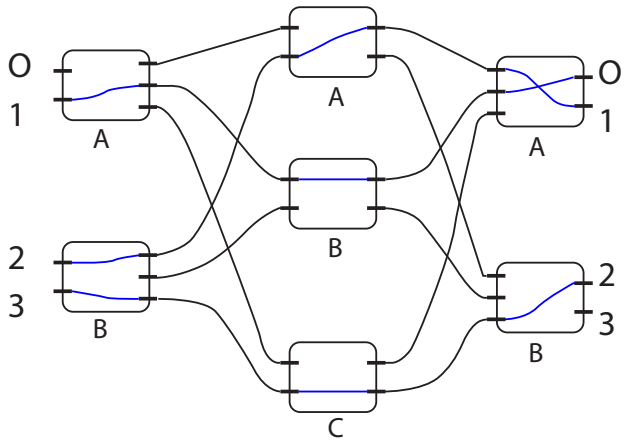


Figure 16.5 An example illustrating that when $m \geq 2n - 1$, the Clos network is nonblocking. The key point is that with a sufficiently large m, there are enough middle stage switches even under the worst case traffic pattern and wiring. In this example, $n = 2$, $m = 3$, and $r = 2$. Wiring within each switch, which determines the switching pattern, is shown in blue. Wiring among the switches, which is fixed a priori, is shown in black. The traffic pattern is that input port 1 needs to be connected to output port 0, input port 2 to output port 1, input port 3 to output port 0, and input port 0 to output port 3. At this point, three input-output port pairs have already been connected as shown. We need to connect input port 0 to output port 3. The other input port sharing the same input stage switch A connects through middle stage switch B. The other output port sharing the same output stage switch B connects through middle stage switch C. Had there not been another middle stage switch, A in this case, there would have been no way to connect input port 0 and output port 3 without rearranging existing connections. But with the presence of switch A, the Clos network becomes non-blocking.

ports on B is already occupied, and most importantly, these $2n - 2$ connections each goes through a *different* middle stage switch. This means we need to have an additional middle stage switch in such a worst case scenario. That means if there are $m \geq 2n - 1$ middle switches, we have the non-blocking property. This is illustrated in Figure 16.5.

It is also interesting to see that r does not factor into the non-blocking condition. But of course, if r is too big, the middle stage switches will have large port counts. Bigger r means there are more input and output stage switches, and larger middle stage switches, which can be recursively built from 3-stage Clos networks using only small switches.

It takes a little longer to show that if $m \geq n$, then a Clos network is rearrangeably nonblocking. We will go through this proof in a homework problem.

A Clos network can have its input part and output part folded. A folded Clos network is often called a **fat tree**. Not to be confused by this terminology, a fat tree enjoys scalability beyond what a tree can. It can achieve the maximum

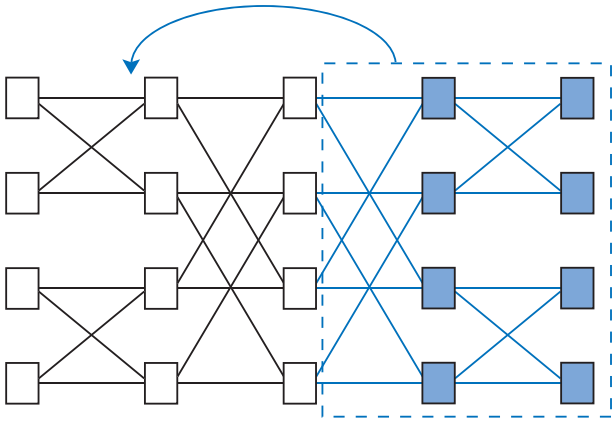


Figure 16.6 From Clos network to fat tree. Due to symmetry around the middle stage of this 5-stage Clos network, the right two stages can be folded to the left two stages. We call a folded Clos network a fat tree. Now each link is bidirectional in the fat tree.

bisection bandwidth without having to oversubscribe traffic as we go up the tree levels. This is shown in Figure 16.6. Fat trees have been implemented in many data centers as the standard way to scale up.

There are several alternatives to Clos networks, such as hybercube, mesh, bufferfly, *etc.* A variant called VL2 builds small trees while trees are still scalable, and then a Clos network among the roots of these trees, as illustrated in Figure 16.7.

Given a topology, we still need to run routing, congestion control, and scheduling of traffic on it. Some of these topics will be touched upon in Advanced Material. When the connectivity in Clos network is sufficiently rich, even simple, randomized routing can load balance the traffic well enough to achieve near optimal bisection bandwidth. In some proprietary systems such as Infiniband, deterministic routing, an even simpler approach, is used.

16.2.2 Comparing network topologies: Internet, data center, and P2P

Before we move on to illustrative examples of data center topologies, we pause to reflect upon three ways of drawing and sizing topologies, one for each key type of wireline technology network. We also explore the root causes behind these different design choices.

1. *The Internet backbone*: Overprovision link *capacities* and then carefully run IP routing, possibly multipath routing. Since routing is not responsive to link load in real time, that job of congestion control is given to the transport layer, and end hosts react to varying loads on a fast timescale by TCP.
2. *Data center networks*: Overprovision *connectivity* by increasing the number

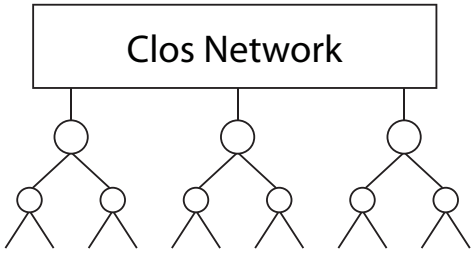


Figure 16.7 The "Virtual Layer 2" (VL2) design leverages spatial hierarchy in data centers. Many servers are connected by trees, and then the trees are connected to each other through a Clos network.

of paths available, and then run massive amounts of multipath routing, either carefully or randomly. Why not do this for the Internet backbone too? Because overprovision connectivity is even more expensive than overprovision capacity in the Internet's spatial scale, unless you overlay, like in P2P.

3. *P2P multicast overlay network*: Overprovision *connectivity* rather than capacity, by increasing both the number of paths *and* the number of source nodes, and then run massive amounts of multi-tree construction by picking not just routing paths but *source-destination relationships*, either carefully or randomly. More than just creating a richly connected topology and then picking many paths, this creates many concurrent multicast trees.

The progression of the above three designs can be summarized as follows: (1) Fix a topology, make pipes fatter and use the pipes intelligently. (2) Enrich the topology by increasing connectivity. (3) Create many topologies to choose from at the same time.

In the end, (2) and (3) can get close to their bisection bandwidth limit and peer upload capacity limit, respectively, but (1) cannot get to full utilization of backbone bandwidth. Furthermore, if there is enough overprovisioning of connectivity, you even get to choose among the connections randomly (like in VL2 and BitTorrent) and be pretty close to the limit. Overprovisioning connectivity pays off better than overprovisioning capacity, if you can *afford* it.

Choice of network design also depends on the dominant cost drivers. In the Internet backbone, digging trenches is the dominant cost for the ISPs. And links are long haul, covering thousands of miles, constrained by the presence of fibers and population. It is very expensive to create connectivity. In a data center, the network inside a large building is a relatively small fraction of the cost,

compared to the server, electricity and cooling costs. So overprovisioning connectivity makes economic sense. P2P is an overlay network, so connectivity is a logical rather than physical concept and even cheaper to overprovision. P2P connectivity can also be dynamically managed through control signals without digging any trenches.

In addition to cost structures, traffic demand's predictability and flexibility are other root causes for these fundamentally different choices in network design. In the Internet, traffic matrices are relatively predictable. Traffic matrix fluctuation on the Internet is over time rather than space, thus can be mitigated by either capacity overprovisioning or time-dependent pricing. In data centers, traffic demands are quite volatile and not yet well understood, another reason to overprovision connectivity. In P2P, you have the option of *changing* the traffic matrix, by picking different peers. So, leveraging that flexibility gives the biggest "bang for the buck."

16.3 Examples

16.3.1 Expanding and folding a Clos network

In this example, we demonstrate how to expand a Clos network from 3 stages to 5 stages, and then rearrange for symmetry before folding into a fat tree.

As illustrated in Figure 16.8, we follow a sequence of five steps:

- Step 1: We start with a particular 3-stage Clos network where $n = 2, m = 2, r = 4$. We would like to replace the middle stage larger switches with small, 2 by 2 switches.
- Step 2: We construct a new 3-stage Clos network where $n = 2, m = 2, r = 2$. Each of these Clos networks can act as a 4 by 4 switch.
- Step 3: We now replace each center stage switch in step (1) with the new 3-stage Clos network in step (2). This recursive operation expands the original 3-stage Clos network into a 5-stage one. There are more switches, but they are all small ones (2 by 2) now.
- Step 4: We conform to the standard input stage connection pattern by appropriately rearranging the positions of switches in stage 2 and stage 4.
- Step 5: Finally, we can fold the 5-stage Clos into a 3-stage fat tree, each link being bidirectional now.

16.3.2 Max flow and min cut

Bisection bandwidth is a special case of the cut size of a network. There is a celebrated result connecting the sizes of cuts in a graph with the the maximum amount of flow that can be routed through the network.

Consider a directed graph with edge capacities illustrated in Figure 16.9(a). We want to find out the maximum flow from source s to destination t. Figure 16.9(b)

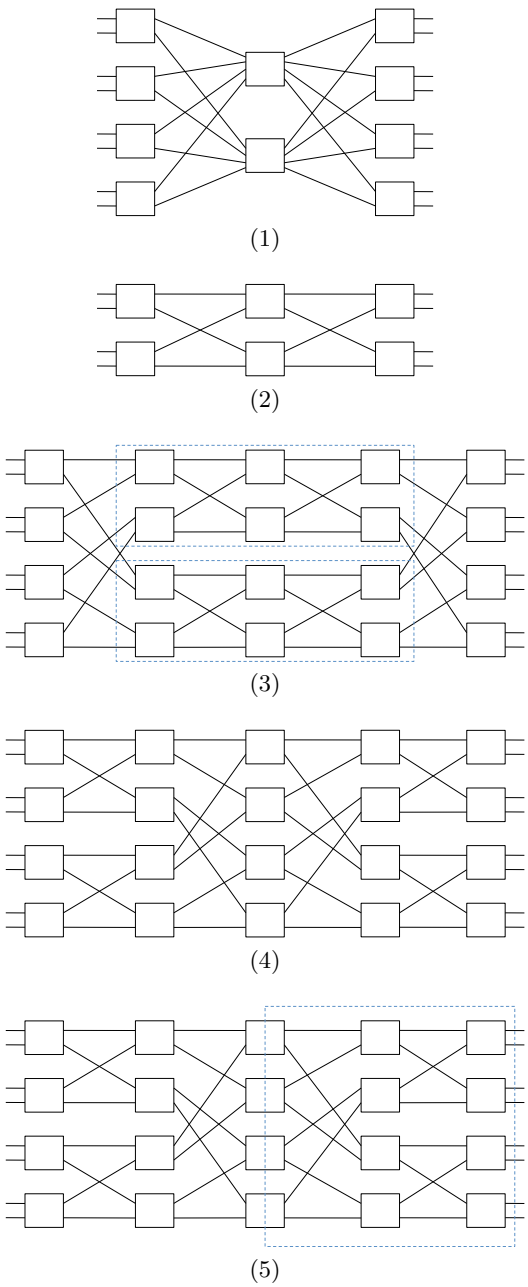


Figure 16.8 Convert a 3-stage (2,2,4) Clos network into a 3-stage fat-tree with only 2 by 2 switches. Each of the middle stage switches in the original Clos network is replaced by a 3-stage (2,2,2) Clos network. Then the switches are rearranged in stages 2 and 4 to create symmetry around the middle stage. Finally fold the network into a more compact one with bidirectional links.

gives a solution of maximum flow computed using the Ford Fulkerson algorithm that we went through in a homework problem in Chapter 13.

In general, a **cut** (S, T) of the network $G = (V, E)$ is defined with respect to a given source-destination pair s, t. It is a partition of the node set V into two subsets: S and $T = V - S$ (nodes in set V other than those in set S), such that $s \in S$ and $t \in T$. The capacity of the cut (S, T), or its **cut size**, is the sum of the capacities on the links from S to T. A *minimum cut* of a network is a cut whose capacity is the minimum over all cuts of the network. Figure 16.9(c) shows a minimum cut in the network.

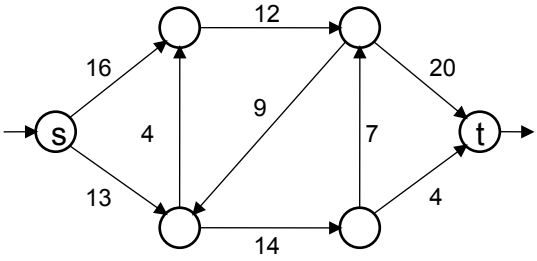
The **max-flow min-cut theorem** states that the maximum amount of flow passing from the source to the destination is equal to the minimum cut of the network with respect to that source-destination pair. In this example, both numbers equal 23 units. The largest possible is clearly 24 units since that is the capacity of the cut that leaves the destination node as T and all the other nodes in S. But that is not the minimum cut here. Instead, the min cut divides the network into the left 4 nodes and the right 2 nodes as shown.

16.4 Advanced Material

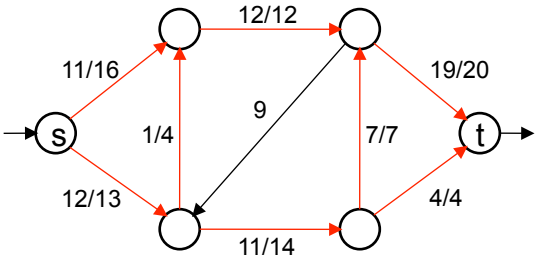
Data centers host a multitude of services, which range from financial, security, websearch, and data mining, to email, gaming, content distribution, and social networking. These services live over varying time scales and belong to different traffic classes with diverse Quality of Service (QoS) needs. For instance, interactive services such as gaming, video streaming, and voice-over-IP are delay-sensitive, whereas other services such as large file transfers and data mining applications are throughput-sensitive. In addition, for efficient sharing of workload, components of a single application can be distributed, processed, and assembled on multiple servers that are located at different data centers. All these result in interesting traffic patterns, both within a data center and over the backbone network that interconnects multiple data centers over a large geographical span. We focus on traffic management within a data center here, especially on four degrees of freedom: topology, placement, routing, and scheduling.

Scalable data center topology. Topology of the connected servers is a key design issue as discussed in this chapter. Figure 16.10 shows some typical data center topologies by interconnecting switches. Most data centers, *e.g.*, tree, VL2, fat tree and DCell, follow a 3-tier architecture. At the bottom is the access tier, where each server connects to one (or two) access switches. Each access switch connects to one (or two) switches at the aggregation tier, and finally, each aggregation switch connects to multiple switches at the core tier. In BCube, servers are assumed to have multiple input and output ports, so that they can be part of the network infrastructure and forward packets on behalf of other servers.

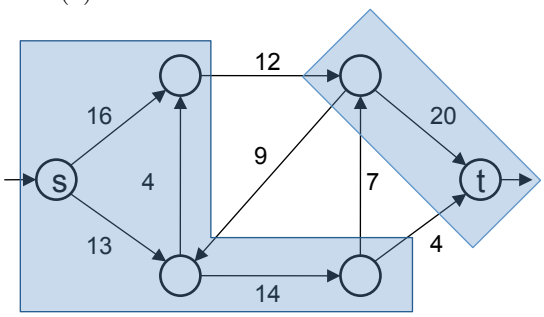
Localizing traffic by flexible VM placement. Cloud customers usually rent multiple machine instances with different capabilities as needed and pay at per-



(a) Network topology and capacity.



(b) The maximum flow from s to t is 23.



(c) The minimum cut $c(S,T)$, where $s \in S$ and $t \in T$, is $12 + 7 + 4 = 23$.

Figure 16.9 Maximum flow equals minimum cut. In the middle graph, a/b means that a units of capacity, out of a total of b units available, is used on a link. Algorithms such as the Ford Fulkerson algorithm can compute the maximum flow from source s to destination t. That must be equal to the minimum size of the cut where s belongs to one side of the cut and t the other.

machine hour billing rate. Virtualization based data centers are becoming the mainstream hosting platform for a wide spectrum of application mixtures. The virtualization technique provides flexible and cost-effective resource sharing in data centers. For example, both Amazon EC2 and GoGrid use Xen virtualization to support multiple **Virtual Machine** (VM) instances on a single physical server. An application job usually subscribes a handful of VMs placed at different hosts that communicate with each other, with different amounts of resource requirements for CPU and memory.

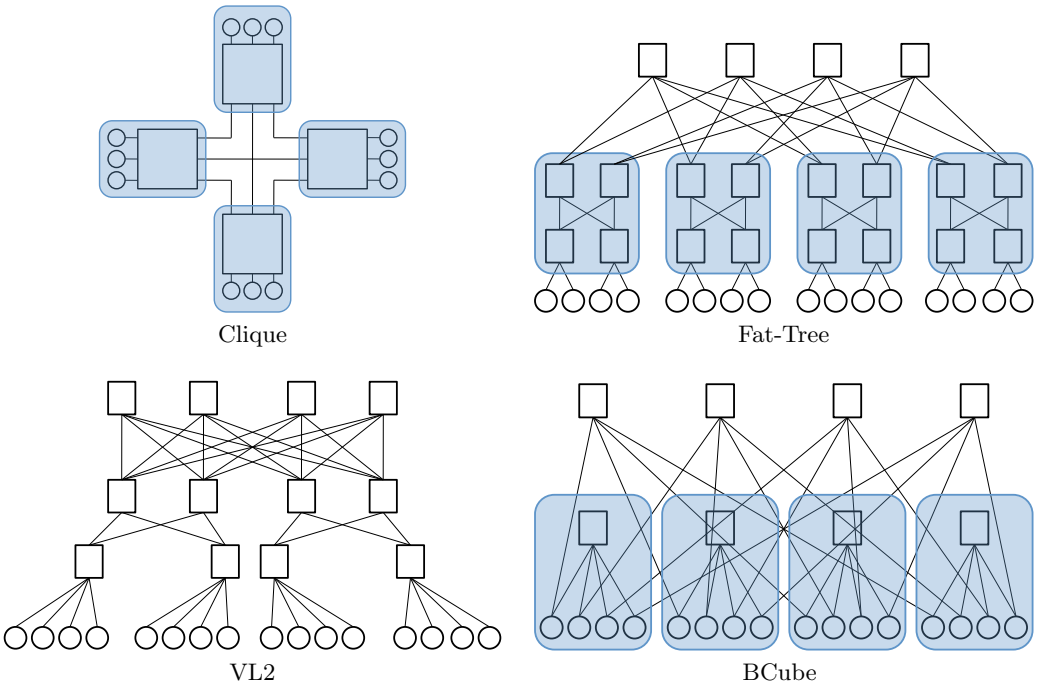


Figure 16.10 There are many topologies possible to scale up a data center. We have seen fat-tree as a folded Clos network, VL2 as trees combined with a Clos network, and there are other options like cliques and cubes. Circles are servers and squares are switches.

A number of proposals have been made to improve the agility inside a data center, *i.e.*, any server can be dynamically assigned to any host anywhere in the data center, while maintaining proper security and performance isolation between services. Maximizing the network bisection bandwidth could be viewed as a global optimization problem — servers from all applications must be placed to ensure the sum of their traffic does not saturate any link.

Route selection by exploiting multipath capability. Data centers rely on the path multiplicity to achieve scaling of host connectivity. Data center topologies often take the form of multi-rooted spanning trees with one or multiple paths between hosts. Route selection can then be congestion-adaptive to bandwidth availability between different parts of the network.

Scheduling is yet another degree of freedom in managing traffic in a data center. The basic problem of job scheduling is essential to a wide variety of systems, from an operating system of a smart phone to an interconnection network on a factory floor. We assume for now that there is a centralized computer to collect all the inputs and computes the output. In Chapter 18, we will encounter the challenge of distributed scheduling in WiFi.

- The input to the scheduling problem is a list of jobs, each with some attributes: the job size (or more generally, the amount of resource required for each type of resources), a strict deadline or a cost of exceeding the deadline, and a quality of service expected, *e.g.*, the minimum throughput. Sometimes there are also dependencies among the jobs, which can be visualized as a graph where each node is a job and each directional link is a dependence between two jobs.
- The output is a schedule: which job occupies which parts of each resource during each time slot.
- Some criteria to judge how good a schedule is, *e.g.*, the distribution of response times (the time it takes to start serving a job), the distribution of job completion times, the efficiency of using all the resources, and the fairness of allocating the resources.

Scheduling has been extensively researched in queuing theory, dynamic programming, algorithm theory, graph theory, etc. There are too many variants to list in this brief summary. Some major scheduling policies include the following, with self-explanatory names:

- *First come first serve*: whenever there is a available resource, the first job to arrive claims it.
- *Smallest job first*: When there are multiple jobs waiting to be scheduled, the smaller jobs get served first. This helps reduce response times, if we just count the number of jobs without weighting them by the job sizes.
- *First to finish first*: This helps reduce completion times by the count of the number of jobs.
- *Longest queue first*: If we group the jobs by the type of resource they request, and each group has its queue to hold jobs waiting to be scheduled. This scheduling policy helps avoid long queues building up as more jobs arrive at the queues.

Joint optimization: Topology, VM placement, routing, and job scheduling are four of the degrees of freedom in managing traffic inside a data center. Optimizing on any one of these alone can be quite inefficient:

- Ill-designed network topology limits the bisectional bandwidth and the path diversity between communicating nodes.
- Suboptimal VM placement introduces unnecessary cross traffic.
- Oblivious routing even in well-designed topologies can under-utilize the network resource by several factors.
- Inefficient scheduling may be constrained by routes available among the servers, which in turn reduces the efficiency of routing.

Having joint control over all the "knobs" provides an opportunity to fully utilize the data center resources. For example, the operators have control over

both where to place the VMs that meet the resource demand, and how to route the traffic between VMs.

Further Reading

The subject of building large networks from small switches is both old and new. Its root goes back to the telephone network design and its current driving application is one of the "hottest" buzz words in networking industry today.

1. The classic paper by Clos in 1953 started the field of switched network design:
[Clo53] C. Clos, "A study of non-blocking switching networks," *Bell System Technical Journal*, vol. 32, no. 2, pp. 406-424, 1953.

2. The following is a standard graduate level textbook on interconnection networking:
[DT04] W. J. Dally and B. P. Towles, *Principles and Practices of Interconnection Networks*, Morgan Kaufmann, 2004.

3. A down-to-earth introduction to key elements in cloud computing, with much detail on the practical side, is the following book:
[Sos11] B. Sosinsky, *Cloud Computing Bible*, Wiley, 2011.

4. The following is a readily accessible, general overviews of cloud research problems:
[GHMP09] A. Greenberg, J. Hamilton, D. A. Maltz, and P. Patel, "The cost of a cloud: Research problems in data center networks," *ACM Sigcomm Computer Communication Review*, vol. 39, no. 1, pp. 68-73, 2009.

5. Here is a comprehensive book on the mathematical foundation of distributed computation:
[BT97] D. P. Bertsekas and J. N. Tsitsiklis, *Parallel and Distributed Computation: Numerical Methods*, Athena Scientific, 1997.

Problems

16.1 *To cloud or not to cloud* ⋆

The Bumbershoot Corporation's biology research center produces 600 GB of new data for every wet lab experiment. Assume the data generated can be easily parallelized, with a negligible overhead.

(a) Suppose the Amazon Web Services (AWS) sells CPU hours at the price of \$0.10/hr per Elastic Compute Cloud (EC2) instance, where each instance

takes 2 hours to process 1 GB of the experimental data. The data transfer fee is \$0.15/GB. What is the price Bumbershoot will need to pay for processing the experiment using the EC2 service?

(b) Suppose the data transfer rate from the research center to AWS is 20 Mbps. What is the total time required to transmit and process the experiment data using the EC2 service?

(c) The Bumbershoot Corperation has 24 computers itself, each taking 2 hours to process a GB of data. Suppose the maintenance fee (including electricity, software, hardware, etc) is \$15 per computer per experiment. What is the total amount of time and cost required to process the experiment? Will Bumbershoot Corperation be willing to use the EC2 service?

(d) We see in (b)(c) that transmission time is a problem in cloud computing. Can you think of a way to overcome this obstacle, so that Bumbershoot can still process the experiment using EC2 service within a day?

16.2 *Rearrangably nonblocking Clos networks* $\star\star\star$

(a) Give an algorithm for routing unicast traffic on an (m, n, r) Clos network with $m \geq n$.

(Hint: Suppose the new unicast traffic to be routed is from input port a to output port b. Arbitrarily select middle switches α, β not being used by a, b, respectively. Assign the new call with middle switch α and rearrange only the calls that use middle switches α, β.)

(b) Consider an $(3, 3, 4)$ Clos network along with its traffic illustrated in Figure 16.11(a). In Figure16.11(b) each node represents an input/output switch, and links with 3 different dotted styles represent the middle switches assigned: the calls $(I2, O4), (I3, O1), (I4, O2)$ are routed through middle switch 1; the calls $(I1, O4), (I2, O1), (I3, O3)$ are routed through middle switch 2; the calls $(I1, O2), (I3, O4), (I4, O1)$ are routed through middle switch 3.

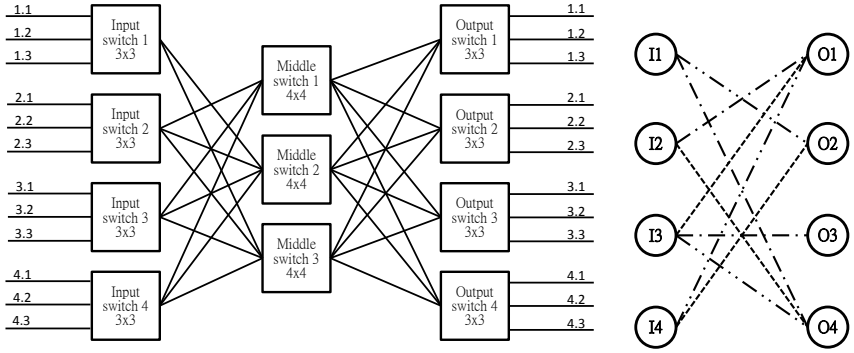
Now, route a new call $(I4, O3)$ based on your algorithm in (a).

16.3 *Ideal throughput* $\star\star$

Consider an interconnection network on a microprocessor chip, represented by a directed graph $G = (V, E)$, where $V = \{v_i : i = 1, \cdots, |V|\}$ is the set of nodes representing the terminals and routers on the chip, $E = \{e_c : c = 1, \cdots, |E|\}$ is the set of links called "channels." Define the following symbols:

$\lambda_{s,d}$: The traffic from input port s to destination d, given a unit of throughput.

$x_{d,c}$: The traffic with destination d on channel c, given a unit of throughput.



(a) (3,3,4) Clos network (b) Network Traffic

Figure 16.11 An example of unicast traffic rearrangement. In (b) the left/right nodes represent input/output switches, and the links with three different types of dotted styles represent the assignment of the middle switches.

b_c**:** The bandwidth of channel c.
γ_c**:** The load of channel c, given a unit of throughput.
A: The node-channel incidence matrix, where

$$A_{ic} = \begin{cases} +1 & \text{if } c \text{ is an outgoing channel from node } i \\ -1 & \text{if } c \text{ is an incoming channel to node } i \\ 0 & \text{otherwise} \end{cases}$$

(a) What is the incidence matrix of the network in Fig.16.12?

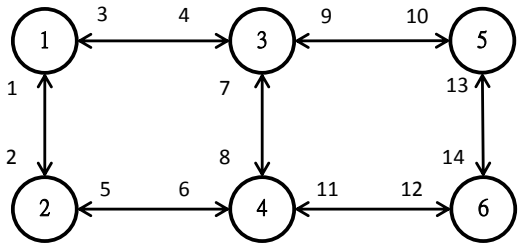


Figure 16.12 A simple network represented by a directed graph.

(b) Define

$$f_{d,i} = \begin{cases} \lambda_{i,d} & \text{if } i \neq d \\ -\sum_{j \neq d} \lambda_{j,d} & \text{if } i = d \end{cases}$$

What is the relationship between $f_{d,i}$ and $x_{d,i}$ under unit throughput?

(c) Express γ_c in terms of $x_{d,c}$ and b_c.

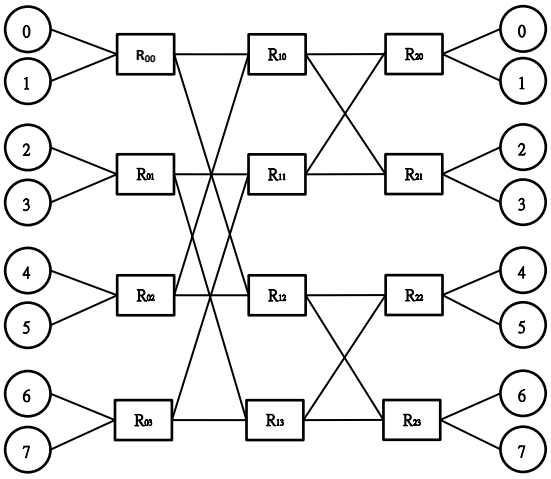


Figure 16.13 An example of butterfly networks.

(d) The ideal throughput Θ^* is the maximum throughput achievable in the network. What is the ideal throughput in terms of γ_c?

(e) Formulate the optimization problem of maximizing the ideal throughput via flow control, for a given traffic pattern $\lambda_{s,d}$ and network topology $\mathbf{A}$.

16.4 *Alternatives to Clos networks* ⋆⋆⋆

(a) Consider the **butterfly network** as shown in Figure 16.13, where each channel has unit bandwidth, and the packets are sent from the input ports (denoted by the left circles) to the output ports (denoted by the right circles).

What is the ideal throughput assuming the random traffic pattern, *i.e.*, each input port s sends $\frac{1}{8}$ unit of traffic to each output port d under unit throughput?

What is the ideal throughput assuming the "bit rotation permutation" traffic pattern? That is, the input port with the address (in binary) $a_2a_1a_0$ sends packets to the output port with the address $a_1a_0a_2$. For example input port $5 = (101)_2$ sends packets only to output port $(011)_2 = 3$.

(b) Repeat (a) for the **cube network** as shown in Figure 16.14, where each channel has unit bandwidth and each node being both the input and output port.

16.5 *Packaging optimization* ⋆⋆

The nodes and links (channels) of an on-chip interconnection networks are

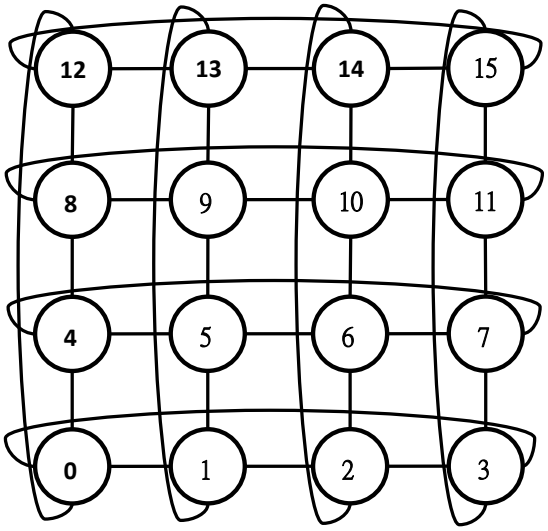


Figure 16.14 An example of cube networks.

constructed on packaging modules. The network topology along with the packaging technology determines the constraints on the chennal's bandwidth. In this question we aim to derive an upper bound on the smallest channel width w_{min}.

Consider a network where channels are composed of unidirectional wires, each having a bandwidth of f units. For an arbitrary node v_n, suppose it has W_n pins available, along with δ_n^+ outgoing channels and δ_n^- ingoing channels. Since all $\delta_n = \delta_n^+ + \delta_n^-$ channels connecting to node v_n need to share the W_n pins, we have

$$w_{min} \leq f \frac{W_n}{\delta_n}$$

Furthermore, consider an arbitrary bipartition C of the network, where there are B_C channels in between the two sets of nodes. In a practical packaging technology, because of the limited space in between the two sets of nodes, the number of wires in-between is bounded by some number W_C as well. So we have the following constraint

$$w_{min} \leq f \frac{W_C}{B_C}$$

Consider a **Cayley graph**, along with a bipartition, as shown in Figure 16.15. Suppose each wire has bandwidth $f = 1$ Gb, each node has $W_n = 140$ pins, and there can be at most $W_C = 200$ wires in between bipartition C. Give an upperbound of the minimum channel bandwidth w_{min} of this network.

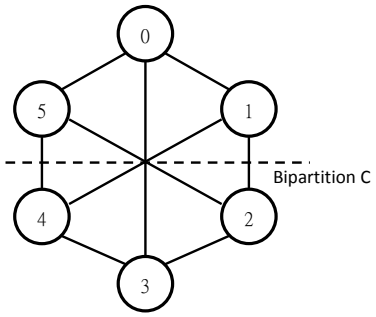


Figure 16.15 A bisection of Cayley graph with 6 nodes. Each link represents two unidirectional channels going in opposite directions.

17 IPTV and Netflix: How can the Internet Support Video?

The Internet provides a "best effort", *i.e.*, "no effort" service. So, how can it support video distribution that often have stringent demands on throughput and delay?

17.1 A Short Answer

17.1.1 Viewing models

Watching video is a significant part of many people's daily life, and it is increasingly dependent on the Internet and wireless networks. Movies, TV shows, and home videos flow from the cloud through the IP network to mobile devices. This trend is changing both the networking and entertainment industries. As of 2011, there were more than 100 million IPTV users in the U.S., and Youtube and Netflix together takes about half of the Internet capacity usage.

This trend is bringing about a revolution in our viewing habits:

- *Content type*: Both user generated and licensed content have become prevalent. Clearly, more user generated content implies an increasing need of upload capacity, which is traditionally designed to be much smaller than download capacity.
- *When*: For many types of video content, we can watch them anytime we want, with the help of devices like Digital Video Recorder (DVR) on IPTV or services like HBO Go.
- *Where*: We can watch any video content almost anywhere, at least anywhere with a sufficiently fast Internet connection.
- *How*: Instead of just the TV and desktop computers, we can watch it on our phones, tablets, and any device with a networking interface and a reasonable screen.
- *How much*: We are watching more video, thanks to applications like Netflix, Hulu, Deja, and embedded videos on many websites. For example, 27 million unique viewers watched 645 million videos on Hulu in a month in summer 2011. Similarly, NBC had 30 million unique viewers watching 96 million videos. Some of these are free, some are free but with intrusive advertisement, some require a monthly subscription, some are pay per view,

and some are part of a bundled service, such as the triple play of IPTV, Internet access, and VoIP. If the Internet connection charge is usage-based, there is also the transportation cost per GB as in Chapter 11.

We can categorize viewing models along four dimensions. Each combination presents different implications to the network design in support of the specific viewing model:

- *Real time vs. precoded*: Some videos are watched as they are generated in real time, *e.g.*, sports, news, weather videos. However, the vast majority is precoded: the content is already encoded and stored somewhere. In some cases, each video is stored with hundreds of different versions, each with a different playback format or bit rate. Real time videos are more sensitive to delay, while precoded videos have more room to be properly prepared. Some other video-based services are not only real time, but also two-way interactive, *e.g.*, video calls, video conferencing, and online gaming. Clearly, interactive video has even more stringent requirements on delay and jitter (*i.e.*, variance of delay over time).
- *Streaming or download*: Some videos, like those on Netflix and YouTube, are streamed to you, meaning that your device does not keep a local copy of the video file (although Netflix movies sometimes can be stored in a local cache, and YouTube has started a movie rental business). In other cases, the entire video is downloaded first before played back at some later point, *e.g.*, in iTunes. Of course the content itself may be automatically erased from local storage if its digital rights are properly managed, like in movie rentals. In between these two modes is also the possibility of *partial* download and playback. As shown in Advanced Material, this reduces the chance of jitter, and is followed in practice almost all the time except for interactive or extremely real time content.
- *Channelized or on demand*: Some contents are organized into channels, and you have to follow the schedule of each channel accordingly. This is the typical TV experience we have had for decades. Even with DVR, you still cannot jump the schedule in real time. In contrast, Video on Demand (VoD) allows you to get the content when you want it. Both YouTube and Netflix are VoD. There are also VoD services on TV, usually charging a premium. Sometimes the content owner changes the model, *e.g.*, in 2011 HBO in the US changed to VoD model with its HBO Go services on computers and mobile devices. In between the two extremes, there is *NVoD*, Near Video on Demand, which staggers the same channel every few minutes, so that within a latency tolerance of that few minutes, you get the experience of VoD.
- *Unicast or multicast*: Unicast means transmission from one source to one destination. Multicast means from one source to many destinations, possibly millions, that belong to a multicast group. An extreme form of multicast

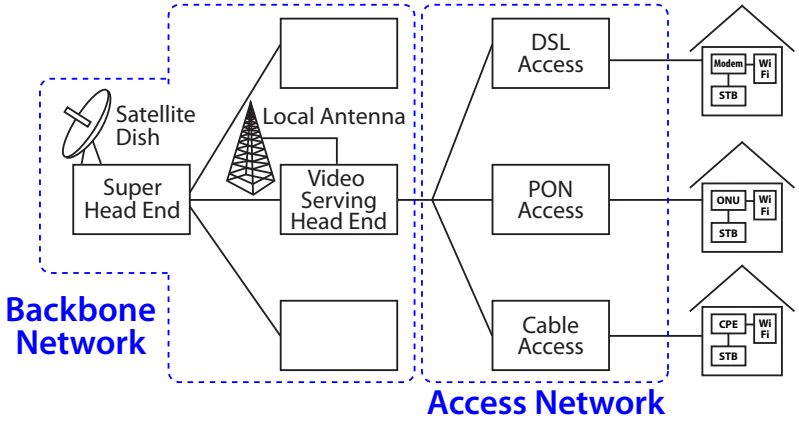


Figure 17.1 A typical architecture of IPTV. The content is collected at the super head-end and distributed to different local video serving head-ends across the country, which also collect local content. Then it is further distributed to access networks running on copper, fiber, or cable, before reaching the homes. This is often carried out in private networks owned and managed by a single operator.

is *broadcast*: everyone is in the multicast group. If you do not want certain content, you do not have to watch it, but it is sent to you anyway. TV is traditionally multicast, sometimes through physical media that are intrinsically multicast too, such as satellite. The Internet is traditionally unicast. Now the two ends are getting closer. We see unicast capabilities in IP-based video distribution, but also multicast in IP networks (carried out either in network layer through IP multicast routing or in application layer through P2P).

It seems that there are $2^4 = 16$ combinations using the above taxonomy of video viewing modes. Obviously some combinations do not make sense, for example, real time video must be streaming based and cannot be download based. But precoded video can be either streaming or download based. Or, true VoD cannot be multicast since each individual asks for the content at different times, but channelized content can be either unicast or multicast.

17.1.2 IP video

The term "IP video" actually encompasses two styles: IPTV and VoI.

The first is **IPTV**, often included as part of the triple or quadruple play service bundle provided by an ISP. It is delivered over a *private and managed network*, with a set top box on the customer's premise. This private network uses IP as a control protocol, but many parts of it are deployed and operated by a single ISP, *e.g.*, a telephone or cable company offering the Internet access. This

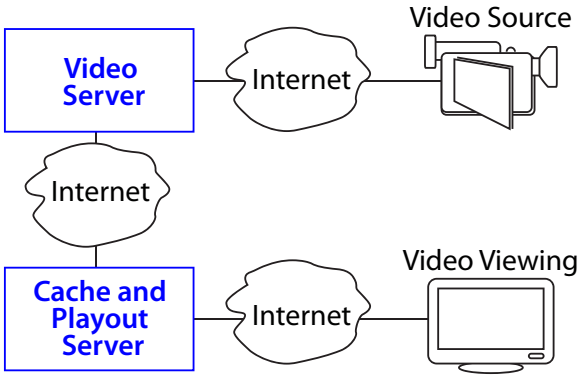


Figure 17.2 A typical architecture of video over the Internet. Video sources, ranging from iPhones to professional video cameras, upload content to video servers, which then distribute them through local caches to the viewers around the world who download the video to their devices. This is all carried out in the public Internet.

makes it easier to control the quality of service. The content is often channelized, multicast, and streaming-based but with recording capability by DVR.

Before TV turned to the Internet access networks, it was delivered primarily through one of the following three modes: broadcast over the air, via satellites, or through cables. So why is the IPTV revolution happening now? There are a few key reasons:

- *Convergence*: almost everything else is converging on IP, including voice. Putting video on IP makes it a unified platform to manage.
- *Cost*: Having a uniform platform reduces the costs of maintaining separate networks.
- *Flexibility*: IP has demonstrated that one of its greatest strengths is the ability to support diverse applications arising in the future.
- *Compression* gets better and access network *capacity* increases sufficiently that it has become possible to send HD TV channels.

The second is **Video over the Internet** (**VoI**). It is delivered entirely over *public networks*, often via unicast, and to a variety of consumer devices. Given the current evolution of business models, VoI is increasingly taking over the IPTV business as consumers access video content over the IP pipes without subscribing to TV services. There are three main types of VoI:

- The content owner sends videos through server-client architectures without a fee, *e.g.*, YouTube, ABC, and BBC.
- The content owner sends videos through server-client architectures with a fee, *e.g.*, Netflix, Amazon Prime, Hulu Plus, HBO Go.

- Free P2P sharing of movies, *e.g.*, Bit Torrent, PPLive.

Whether it is IPTV or VoI, the quality measures depend on bit rate, delay, and variation of delay called jitter. What kind of bit rates do we need for videos? It depends on a few factors, *e.g.*, the amount of motion in the video, the efficiency of the compression methods, the screen resolution, and the ratio of viewing distance and screen size. But generally speaking, the minimum requirement today is about 300 kbps. Below that the visual quality is just too poor even on small screens. For standard definition movies we need about 1Mbps, and a typical movie takes 1-2 GB. For high definition movies we need at least 6-8 Mbps, and a typical movie takes 5 GB. Truly HD video needs 19-25 Mbps to be delivered.

We touched upon the revenue models for IPTV and VoI. As to the cost models, they often consist of the following items:

- *Content*: The purchase of content distribution rights. Popular and recent movies and TV series naturally charge more.
- *Servers*: The installation and maintenance of storage and computing devices.
- *Network capacity*: The deployment or rental of networking capacities to move content around and eventually deliver to consumers.
- *Customer premise equipment*, such as set top boxes or game consoles.
- *Software*: The software systems that manage all of the above and interface with consumers.

17.2 A Long Answer

As shown in Figure 17.3, the overall protocol stack for IP video includes the following: MPEG over HTTP/SIP/IGMP/RTSP, over RTP/UDP/TCP, over IP, over ATM or Ethernet, over wireless/fiber/DSL/cable. In this section, we go into some detail of the top 3 layers: compression, application, and transport, trying to highlight interesting networking principles beyond the alphabet soup of protocol acronyms.

17.2.1 Compression

A video is a sequence of frames moving at a particular speed. Each frame is a still picture consisting of **pixels**. Each pixel is described by its colors and luminance digitally encoded in bits. The number of bits per frame times the number of frames per second gives us the **bit rate** of a video file. Typical frame rates are 25 or 29.97 frames per second for standard definition, 50 or 60 frames per second for high definition. Typical pixels per frame for high definition video are $1280 \times 720 = 921,600$ or $1920 \times 1080 = 2,073,6004$.

In order to put more content into a given pipe, we need **compression**. It is the process of taking out redundancies in signals. If the resulting file can be later recovered, say at the consumer device, to be exactly the same as the original

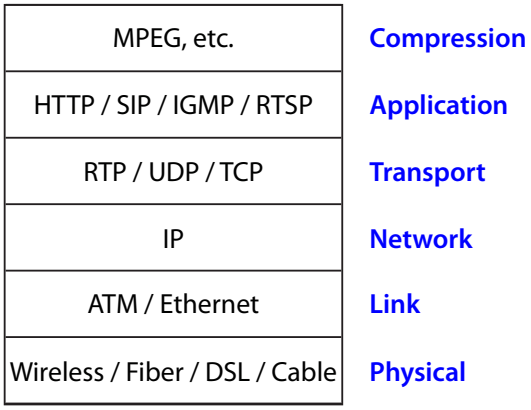


Figure 17.3 Layered network architecture in support of video traffic. Compression standards such as MPEG use various application layer protocols, which in turn rely on combinations of transport and network layer protocols. The focus of this section is on a few key ideas in the top three layers.

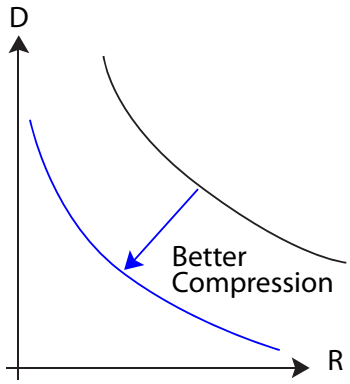


Figure 17.4 Rate distortion curves for two different lossy compression schemes. Distortion can be measured by objective metrics or subjective tests. Higher rate leads to lower distortion. The closer to the origin the tradeoff curve, the better the tradeoff.

one, that is called **lossless compression**, *e.g.*, Lempel Ziv compression used in zipping files. Otherwise, it is called **lossy compression**, and there is a tradeoff between the compression ratio (the size of the file after compression relative to that before compression) and the resulting fidelity. This is called **rate distortion** tradeoff, as shown in Figure 17.4.

There are many techniques to help achieve the best and largest range of such

a tradeoff by taking out redundancies in the bits, *e.g.*, transform coding (seek structures in the frequency domain of the signals), Huffman coding (reduce the expected length of the compressed signal by making frequently appearing codes shorter, like what we saw in Chapter 10), and perceptual coding for video (let people's perceptual process guide the choice of which pixels and which frames to compress). Many frames also looks alike. After all, that is how a perception of continuous motion can be registered in our brain. So video compressors often only need to keep track of the differences between the frames, leading to a significant saving in the number of bits needed to represent a group of pictures.

Video compression has made significant gains over the past two decades:

- **MPEG1** standard in 1992: this was used in VCD (which uses 1Mbps bit rate).
- **MPEG2** (or H.262 by a different standardization body called ITU-T) standard in 1996: this was used in DVD (which uses about 10Mbps bit rate).
- **MP3** is the layer 3 of MPEG2 standard (there is no MPEG3, that was absorbed into MPEG2): this popular standard for the online music industry is for encoding just audio, and can achieve a 12:1 compression ratio.
- **MPEG4** standard in 2000: this is the current video compression standard family.
- MPEG4 Part 10 (also called AVC or H.264) in 2004: with 16 profiles, it offers substantial flexibility. It is also at least twice as good as MPEG2's compression capability. It is used in HDTV (with 15-20 Mbps bit rate) and Blu-ray (with 40 Mbps bit rate). It can readily achieve a compression factor of 100.
- There are also other non-MPEG formats: H.261 was popular in IP video in the early days, and Quick Time by Apple is merging into MPEG4. There are also Windows Media Player by Microsoft, Flash by Adobe, and Real Media Viewer by Real Networks.

A key idea in MPEG compression is to exploit the redundancy across frames when the motion is not rich. This is called motion compensation with inter-frame prediction. There are three types of frames, and collectively a set of them form a **Group of Pictures** (GoP):

- **I frame**: this is an independent frame. Its encoding does not depend on the frames before or after it.
- **P frame**: this type of frame depends on the previous I (or P) frame, but not the one after it.
- **B frame**: this type of frame depends on both the I (or P) frames before and after it.

Each GoP must start with an I frame, followed by a sequence of P and B frames, as shown in Figure 17.5. The I frame is the most important one, while P and B frame losses are much more tolerable. At the same time, I frames are harder to compress than P and B frames, which use motion prediction to assist in compression.

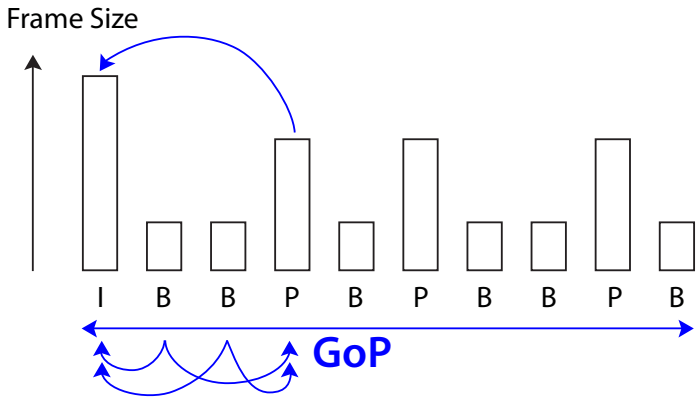


Figure 17.5 A typical structure of a Group of Pictures (GoP). Each GoP starts with an I frame, followed by a sequence of B and P frames. I frame is independent and the most important one in each GoP. P frame depends on the previous I/P frame. B frame depends on both the previous and following I/P frames. Some of these dependence relationships are shown in arrows. Choosing the length and structure of a GoP affects the bit rate, error resiliency, and delay in channel change.

The length of a GoP influences the following metrics:

- *Bitrate efficiency*: If GoP is longer, there are more P and B frames (the more easily compressible ones). The bit rate becomes lower.
- *Error resilience*: If an I frame is lost and, consequently, the entire GoP needs to retransmitted, a longer GoP means that more frames need to be retransmitted. There is a tradeoff between efficiency and resilience, as we will see in a homework problem.
- *Instant channel change*: for channelized video content, the ability to change channels fast is important if the traditional TV viewing experience is to be replicated on IPTV. Since GoP represents the logical basic unit for playback, longer GoP means that the viewer needs to wait longer in changing channels. There are also other factors at play for channel change, such as multicast group operations explained next.

17.2.2 Application layer

In addition to the ability to do multicast routing, we also need Internet Group Management Protocol (**IGMP**). It runs the multicast group management and tells the router that a client (an end user device) wants to join a particular multicast group. There are only two essential message types: the router asks a `membership-query` to clients, and each client replies with a `membership-report` telling the router which groups it belongs to. There is an optional message

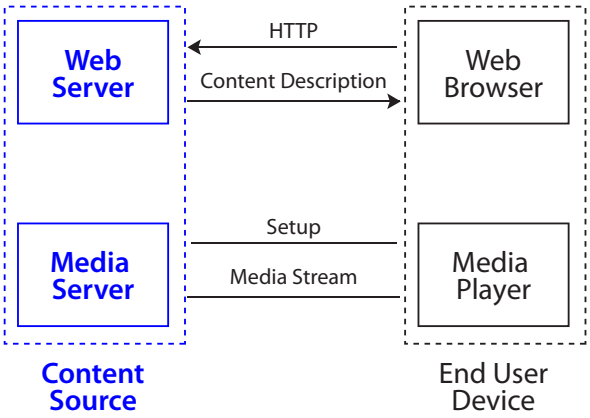


Figure 17.6 Real Time Streaming Protocol (RTSP) at work. An HTTP session first enables control information to be exchanged between the client web browser and the web server. This is followed by the actual media stream from the media server to the media player software on the end user device.

`leave-group` from clients to routers. This message is optional because by the principle of *soft state* explained in Chapter 19: if the membership report information does not periodically refresh the groups, it is assumed to leave the group.

Joining and leaving multicast groups incur propagation and processing delays. Instant channel change may become frozen. To accelerate a channel change, ISPs may send some unicast GoP to the end user when she first requests a channel change. It goes into multicast mode once the "join group" request is processed. There is clearly a tradeoff between network efficiency and user experience. We will see more tradeoffs involved in optimizing the networked delivery of video.

For streaming application, we often use Real Time Streaming Protocol (**RTSP**). It allows a media player to control the transmission of a media stream, *e.g.*, fast forward, rewind, pause, play. It is independent of compression standard or transport protocol.

A typical procedure is shown in Figure 17.6. The request for content first runs over HTTP from a web browser to a web server. Then, knowing the type of media and compression used by reading the response received from the web server, the client can open the right media player, which then uses RTSP to carry out message passing between the client and the media server that actually holds the video file. Unlike HTTP, RTSP must keep track of the current state of the file at the client media player so that it can operate functionalities like pause and play. You must have realized there is a lot of overhead associated with managing video traffic, and we will see more of such overhead in Chapter 19.

Another protocol often used for IP multimedia transmission is Session Initia-

tion Protocol (**SIP**) from IETF. It establishes a call between a caller and callee over an IP network, determines the IP address, and manages the call, *e.g.*, adds callers, transfers or holds calls, or changes voice encoding. Together with video standards, SIP can also provide a mechanism for video conferencing.

Yet another commonly used protocol is **H.323** from ITU. It is actually a large suite of protocols involving many components discussed in this section.

You probably have realized that there are many standardization bodies: IETF for many Internet related protocols, ITU and IEEE have many standardization bodies, and some major standards have their own governing bodies, *e.g.*, 3GPP and 3GPP2 for cellular, WiMax Forum for WiMax, DSL Forum for DSL, MPEG for video compression, *etc.* The way they operate is a mix of technology, business, and political factors, but they all share a common goal of inter-operability among devices so that the networking effect of technology adoption can be achieved.

17.2.3 Transport layer

We have seen the *connection-oriented* transport protocol of TCP in Chapter 14. But much multimedia traffic, especially real time or interactive ones, runs instead over User Datagram Protocol (**UDP**) in the transport layer. UDP is *connectionless*, meaning that it does not try to maintain end-to-end reliability or even sorting packets in the right order. It works with IP to deliver packets to the destination, but if the packets do not get there, it will not try to solve that problem. For example, Skype uses UDP unless the client sits behind a firewall that only allows TCP flows to pass through.

UDP is fundamentally different from TCP in the end-to-end control of the Internet. Why would applications with tight deadlines prefer UDP? It boils down to the tradeoff between timeliness and reliability:

- TCP uses 3-way handshake (explained in Chapter 19) to establish a session, whereas UDP does not introduce that latency. TCP uses congestion control to regulate the rate of sending, whereas UDP sends out the packet as soon as it is generated by the application layer.
- TCP ensures reliability through packet retransmission, but many multimedia applications have their own built-in error resilience, for example, losing a B frame in a GoP can often be concealed in a media player so that the viewers cannot tell. Moreover, a lost and retransmitted packet will likely be too late to be useful in playback by the time it arrives at the destination. It is instead more important to avoid holding back playback and just proceed.

In addition to real time or interactive multimedia, many network management or signaling protocols, like SNMP in Chapter 19 and RIP in Chapter 13, also run on top of UDP. For these signaling protocols, there are two more reasons to prefer UDP:

- TCP maintains too many states for each session compared to UDP. So UDP can support many more parallel sessions at the same time.

- TCP header is 20 bytes and UDP is 8 bytes. For small control packets, this difference in overhead matters.

A protocol on top of UDP is Real Time transport Protocol (**RTP**), heavily used in many IP multimedia applications including VoIP. It specifies a format to carry multimedia streams. The key challenge here is to support many types of media formats, including new ones coming up. And the key solution idea is to specify a range of profiles and payload formats, specific to each media type, without making the header dependent on the media type. RTP runs on the data plane, and its companion RTP Control Protocol (**RTCP**) runs the control plane. RTCP keeps track of the RTP stream's information, such as the number of packets, the number of bytes, and the time stamp information. It monitors the statistics and synchronizes multiple streams.

17.3 Examples

17.3.1 Video quality and I/P/B frames

In this example, we will explore the effect of dropped I, P, and B frames on video quality. Let each grayscale value of the pixel at position (x, y) in frame i be represented by $p_i(x, y)$. The transmitted frame is $\bar{p}_i$ and the received frame is p_i. For our metric of video quality, we use $l - 1$ norm, the sum of absolute differences, *i.e.*, error $= \sum_{x,y,i} |p_i(x, y) - \bar{p}_i(x, y)|$.

Consider a pixel resolution of 2 by 2. For simplicity, assume all pixel values in a given frame are the same (this is a very boring video). An entire GoP is indexed by $i = 1, 2, 3, 4$. We will drop each frame of the GoP in turn, and quantify the effect on our error metric. Assume that frame 0 and frame 5 (belonging to the preceding and following GoPs, respectively) are always successfully received.

0 0	1 1	2 2	3 3	4 4
0 0	1 1	2 2	3 3	4 4
$i = 0$	$i = 1$	$i = 2$	$i = 3$	$i = 4$
Last GoP B frame	I frame	P frame	B frame	B frame
		5 5		
		5 5		
		$i = 5$		
		Next GoP I frame		

Recall that an I frame has no reference frame, a P frame uses the last I or P frame as the reference frame, and a B frame uses the last I or P frame and the

next I or P frame as reference frames. An "error" in frame i means that either (i) frame i is missing, or (ii) frame i had to perform error concealment because frame i's reference is missing. We can set up a few error-concealment rules at the receiver:

- If the receiver misses any frame, it instead displays the last available frame.
- If the receiver detects an error in the reference frame of a P frame, it also displays the last available frame in place of the P frame.
- If a receiver detects an error in a reference frame of a B frame, it displays the other reference frame.

If the I frame of this GoP is dropped, the receiver displays what is summarized in Table tab:gop1:

	$\bar{p}(1,1)$	$\bar{p}(1,2)$	$\bar{p}(2,1)$	$\bar{p}(2,2)$
$i=0$	0	0	0	0
$i=1$	0	0	0	0
$i=2$	0	0	0	0
$i=3$	5	5	5	5
$i=4$	5	5	5	5
$i=5$	5	5	5	5

Table 17.1 Frame 1: Dropped, so repeat frame 0. Frame 2: Error in reference frame 1, so repeat frame 1. Frame 3: Error in reference frame 2, so display frame 5. Frame 4: Error in reference frame 2, so display frame 5.

Then the error between the transmitted and the received picture is:

$$\begin{aligned} error &= |p_1(1,1)-\bar{p}_1(1,1)| + |p_1(1,2)-\bar{p}_1(1,2)| + |p_1(2,1)-\bar{p}_1(2,1)| + |p_1(2,2)-\bar{p}_1(2,2)| \\ &+ |p_2(1,1)-\bar{p}_2(1,1)| + |p_2(1,2)-\bar{p}_2(1,2)| + |p_2(2,1)-\bar{p}_2(2,1)| + |p_2(2,2)-\bar{p}_2(2,2)| \\ &+ |p_3(1,1)-\bar{p}_3(1,1)| + |p_3(1,2)-\bar{p}_3(1,2)| + |p_3(2,1)-\bar{p}_3(2,1)| + |p_3(2,2)-\bar{p}_3(2,2)| \\ &+ |p_4(1,1)-\bar{p}_4(1,1)| + |p_4(1,2)-\bar{p}_4(1,2)| + |p_4(2,1)-\bar{p}_4(2,1)| + |p_4(2,2)-\bar{p}_4(2,2)| \\ &= 4|p_1(1,1)-\bar{p}_1(1,1)| + 4|p_2(1,1)-\bar{p}_2(1,1)| + 4|p_3(1,1)-\bar{p}_3(1,1)| + 4|p_4(1,1)-\bar{p}_4(1,1)| \\ &= 4|1-0| + 4|2-0| + 4|3-5| + 4|4-5| \\ &= 24. \end{aligned} \tag{17.1}$$

If instead the P frame is dropped, the receiver displays what is summarized in Table 17.2.

The error between the transmitted and the received picture is:

	$\bar{p}(1,1)$	$\bar{p}(1,2)$	$\bar{p}(2,1)$	$\bar{p}(2,2)$
$i=0$	0	0	0	0
$i=1$	1	1	1	1
$i=2$	1	1	1	1
$i=3$	5	5	5	5
$i=4$	5	5	5	5
$i=5$	5	5	5	5

Table 17.2 Frame 2: Dropped, so repeat frame 1. Frame 3: Error in reference frame 2, so display frame 5. Frame 4: Error in reference frame 2, so display frame 5.

$$\begin{aligned} error &= 4|p_1(1,1) - \bar{p}_1(1,1)| + 4|p_2(1,1) - \bar{p}_2(1,1)| + 4|p_3(1,1) - \bar{p}_3(1,1)| + 4|p_4(1,1) - \bar{p}_4(1,1)| \\ &= 4|1-1| + 4|2-1| + 4|3-5| + 4|4-5| \\ &= 20. \end{aligned} \tag{17.2}$$

If the first B frame, frame 3, is dropped, the receiver displays what is summarized in Table 17.3.

	$\bar{p}(1,1)$	$\bar{p}(1,2)$	$\bar{p}(2,1)$	$\bar{p}(2,2)$
$i=0$	0	0	0	0
$i=1$	1	1	1	1
$i=2$	2	2	2	2
$i=3$	2	2	2	2
$i=4$	4	4	4	4
$i=5$	5	5	5	5

Table 17.3 Frame 3: Dropped, so repeat frame 2.

The error between the transmitted and the received picture is:

$$\begin{aligned} error &= 4|p_1(1,1) - \bar{p}_1(1,1)| + 4|p_2(1,1) - \bar{p}_2(1,1)| + 4|p_3(1,1) - \bar{p}_3(1,1)| + 4|p_4(1,1) - \bar{p}_4(1,1)| \\ &= 4|1-1| + 4|2-2| + 4|3-2| + 4|4-4| \\ &= 4. \end{aligned} \tag{17.3}$$

If the second B frame, frame 4, is dropped, the receiver displays what is summarized in Table 17.4.

	$\bar{p}(1,1)$	$\bar{p}(1,2)$	$\bar{p}(2,1)$	$\bar{p}(2,2)$
$i=0$	0	0	0	0
$i=1$	1	1	1	1
$i=2$	2	2	2	2
$i=3$	3	3	3	3
$i=4$	3	3	3	3
$i=5$	5	5	5	5

Table 17.4 Frame 4: Dropped, so repeat frame 3.

The error between the transmitted and the received picture is:

$$\begin{aligned} error &= 4|p_1(1,1) - \bar{p}_1(1,1)| + 4|p_2(1,1) - \bar{p}_2(1,1)| + 4|p_3(1,1) - \bar{p}_3(1,1)| + 4|p_4(1,1) - \bar{p}_4(1,1)| \\ &= 4|1-1| + 4|2-2| + 4|3-3| + 4|4-3| \\ &= 4. \end{aligned} \tag{17.4}$$

The greatest error resulted from dropping the I-frame, followed by the P-frame, and finally the B-frames. More important frames cause more error in the GoP when dropped, causing a bigger drop in the visual quality.

17.3.2 Latency-jitter tradeoff

In this example, we look at the effect of buffering on video playback. We will see what the playback delay should be to provide a smooth viewing experience.

Assume there is one frame per packet. Figure 17.7 shows the transmitted, arrived (at the receiver), and played frames over time. We refer to these as the timing curves, abbreviated as the V, A, and P curves. V and A curves are given by the source and the network, and our job is to design the best P curve.

The source transmits at a constant rate, so V is a superposition of unit step functions. Let V_i denote the time at which packet i is transmitted, A_i the time at which packet i is received, and P_i the time at which packet i is displayed to the user. The delay between the transmitted and received packet is given by $d_i = A_i - V_i$. Then, abusing the notation a little to use vectors to represent the discrete jumps on the curves, we have

$$\mathbf{V} = \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \end{bmatrix}, \qquad \mathbf{A} = \begin{bmatrix} 5.8 \\ 7.5 \\ 8 \\ 9.4 \end{bmatrix}, \qquad \mathbf{d} = \mathbf{A} - \mathbf{V} = \begin{bmatrix} 4.8 \\ 5.5 \\ 5 \\ 5.4 \end{bmatrix}. \tag{17.5}$$

In reality, we cannot know $\mathbf{A}$ ahead of time and have to either estimate it or

adapt in real-time, like in a homework problem. For now, we assume it is known. P must be a unit step function since frames need to be displayed at a constant rate. When should playback begin, *i.e.*, what is the value of P_1? The goal is to minimize the total delay experienced by the user:

$$\begin{aligned} \text{minimize} \quad & \sum_i (P_i - V_i) \\ \text{subject to} \quad & P_{i+1} = P_i + 1, \ \ \forall i \\ & P_i \geq A_i, \ \ \forall i \\ \text{variables} \quad & \{P_i\}. \end{aligned}$$

The first constraint in the above optimization ensures that P is a unit step function, and the second constraint says that playback of a packet can only occur after the packet has been received. The objective function can be further simplified; since P and V are both unit step functions, $P_i - V_i$ is the same for all i, so the objective function is equivalent to minimizing any single $P_i - V_i$.

This problem can be solved easily through visual inspection. This problem is essentially shifting a unit step function P to the left, until $P_k = A_k$ for some k, and $P_i \geq A_i \ \ \forall i \neq k$. So we want to make curve P as close to curve A as possible but still rest below A.

From Figure 17.7, clearly $k = 2$. Since $P_2^* = A_2 = 7.5$ s, we have $P_1^* = 6.5$ s. This means playback should begin at 6.5 s, which in turn means delaying playback by $P_1 - A_1 = 6.5 - 5.8 = 0.7$ s, so as to avoid frozen video due to the variation of packet arrivals through the network.

In general, for a constant rate source, the playback should begin at $P_1^* = A_1 + D$, where $D = \max_i(d_i - d_1)$ and represents the maximum delay variation. This formula applies to the above example. Since $D = d_2 - d_1 = 5.5 - 4.8 = 0.7$ s, we have $P_1^* = A_1 + D = 5.8 + 0.7 = 6.5$ s.

In a homework problem, we will explore how to control jitter if the arrival times are not known, as is almost always the case in reality.

17.4 Advanced Material

There are three general approaches in managing Quality of Service (**QoS**) on top of the simple connectivity services, in the best effort style, offered by the thin waist of TCP/IP:

- Treat different sessions differently during resource allocation.
- Regulate which clients can be admitted.
- Distribute servers to strategic locations.

As an analogy, think of a highway's traffic control. Differentiating resource allocation is like reserving a lane for certain vehicles, like a car-pool lane. Regulating admission is like using the on-ramp traffic lights during congestion hours. Distributing servers is like constructing new exits of popular destinations like grocery

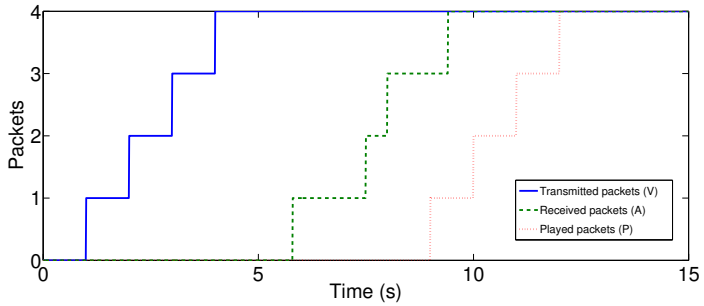


Figure 17.7 Playback buffer smoothes video playback. This graph shows V the timing curves at the source: how packets are transmitted with a constant rate; A at the receiver: how the packets arrival times vary as they traverse the network, and P at playback: how playback latency can smooth the arrival jitter. V and A curves are given, and the P curve needs to be designed to make sure it is a superposition of unit step functions, lies below the A curve, and yet is as far to the left as possible.

stores, which is clearly a much longer timescale operation compared to the other two.

17.4.1 Different queue service policies

Different sessions can be treated differently, inside a node or along a link in a network. For example, there are several standard queuing disciplines in a router. As shown in Figure 17.8, the following three methods will create different sequences of service timing among the incoming sessions:

- *Round robin*: each class of traffic takes turns.
- *Weighted fair queuing*: while taking turns, one class can receive a higher rate than another.
- *Priority queuing*: higher priority class packets are processed before lower priority ones, which have to wait until there are no more higher priority packets in the queue.

What constitutes a *fair* allocation among competing sessions? This is yet another instance where we touch upon the notion of fairness. We will see in Chapter 20 a systematic treatment of the subject.

Of course, differential treatment methods as above do not provide a guarantee on quality of service. For that, we need resource *reservation* methods. There are dynamic versions of establishing end to end circuits in the network layer and reserving adequate resources (such as capacity, timeslot, processing power) so that the end user experience is guaranteed to be good enough. Handoff in mobile networks in Chapter 19 will offer such an example.

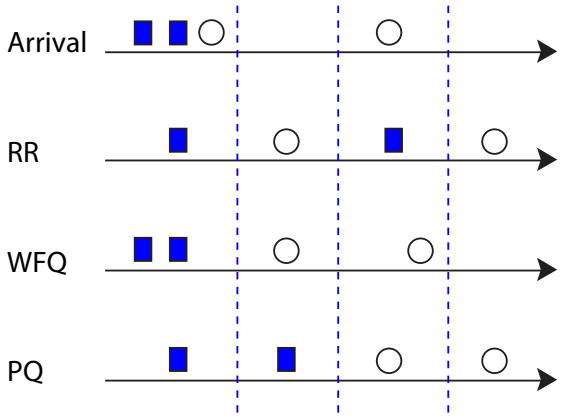


Figure 17.8 Three queuing disciplines give different orders of packet service. There are two classes arriving over the time slots denoted by dotted lines. Each class has two packets. In round robin scheduling, the square packets and circle packets are served in turn, one packet in each timeslot. In weighted fair queuing, the two classes take turns, but square packets get a higher service rate. In priority queuing, square packets have strict priority, and circle packets only get their chances when all square packets have been sent.

17.4.2 Admission control

An alternative approach to managing resource competition is to regulate the demand. In some sense, TCP congestion control does that in a feedback loop, on the timescale of RTT. Policing or throttling further shapes the rate of traffic injection into the network.

For an *open loop* control, we can use admission control: deciding whether a session should be allowed to start transmitting at any given timeslot. Such admission control is usually carried out at the edge of the network, often at the first network element facing the end user devices. In fact, even time dependent pricing can be viewed as a generalization of admission control right on users' devices.

One possible admission control is to control the peak rate, over some timescale, of traffic injection. This is like the ramp light that regulates cars getting onto a highway and smoothes the traffic injection into the highway network during congested hours. By changing the rate of showing green lights at the ramp, we can control the rate of adding traffic onto the highway (the backbone network), at the expense of causing congestion at the ramp (the access network).

An example of admission control is **leaky bucket**, shown in Figure 17.9. In order for each packet to be admitted, there must be a token dripping from a (conceptual) leaky bucket. The bucket drips tokens at a rate of r, and has a volume of B tokens. In a homework problem, we will see that weighted fair

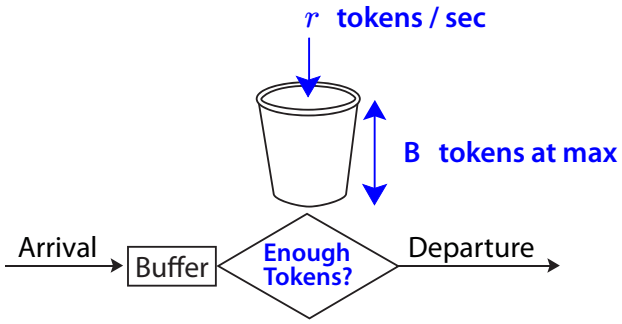


Figure 17.9 Leaky bucket for admission control. Each bucket can hold at most B tokens, and r tokens are added to the bucket per second. Each transmitted packet consumes one token. In order for a packet in the buffer to be transmitted on the egress link, there must be a token available in the bucket.

queuing and leaky bucket can together shape any arrival patterns to a desirable one.

17.4.3 Content distribution

Yet another approach, in addition to differentiating resource allocation and controlling user admission, is to change the location of the source of content, so that the content is brought closer to the users, as shown in Figure 17.10. This leverages the drop of storage cost and the popularity of certain content to create *spatially pipelined* distribution of content.

The idea of changing the location of source has long been practiced in web proxy servers. ISPs cache popular web content at local storage closer to end users. A whole industry sector has also been generated, operators of a **Content Distribution Network** (CDN). They serve either ISPs or content owners, and manage the following processes:

- Deploy many servers, sometimes in private server farms and sometimes in shared data centers. These are often called mirror sites.
- Replicate content and place them in different servers. This involves optimization based on the availability of high speed links and high capacity storage, as well as prediction of content popularity in different geographic areas.
- For each content request, select the right server to serve as the content source. This server selection optimization minimizes the user perceived delay, for a given routing, and is run by the CDN. It may also have unintended

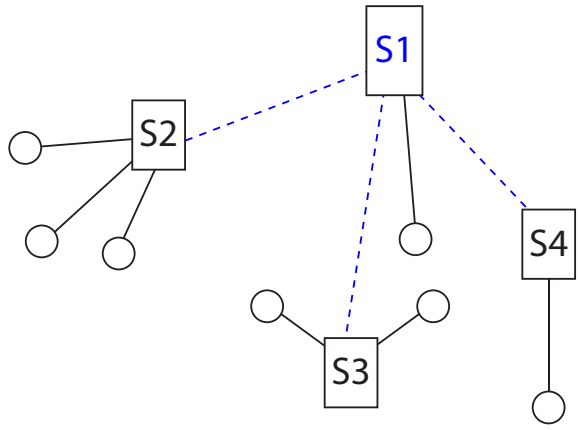


Figure 17.10 Content distribution network operation. The content originally placed in server S1 is replicated in three other locations, S2, S3, and S4, sent through fiber links in dotted lines. When a client requests a piece of content, a particular server is selected as the source to serve that demand.

interactions with traffic engineering by the ISP that selects routes based on a given source-destination relationship.

The convergence of content owner and content transporters is creating interesting dynamics. Proper operation of CDNs can create a win-win: consumers get better quality of service as delay is reduced and throughput increased, and content owners or ISPs reduce the cost of deploying large capacity servers and pipes (as the congestion in the network and around the servers is reduced).

Further Reading

The subject matter of this chapter spans both analytic models of quality of service and systems design of protocols.

1. The fundamentals of video signal processing can be found in many graduate textbooks on the subject, including the following recent one:
[Bov09] A. C. Bovik, *The Essential Guide to Video Processing*, Academic Press, 2009.

2. The following book provides a concise summary of all the major video over IP systems, including IPTV and Video over the Internet:
[Sim08] W. Simpson, *Video over IP*, 2nd Ed., Focal Press, 2008.

3. The following computer networking textbook provides much more detail

about the application and transport layer protocols like IGMP, RTSP, SIP, UDP, and RTP:

[KR09] J. Kurose and K. Ross, *Computer Networking: A Top-Down Approach*, 5th Ed., Addison Wesley 2009.

4. The following classic paper combines leaky bucket admission control and generalized processor sharing to provide quality guarantee:

[PG93] A. Parekh and R. G. Gallager, "A generalized processor sharing approach to flow control in integrated services networks: The single-node case," *IEEE/ACM Transactions on Networking*, vol. 1, no. 3, pp. 344-357, June 1993.

5. The following book provides a concise survey of both the deterministic, algebraic approach and stochastic, effective bandwidth approach to the design of quality guarantee in a network:

[Cha00] C. S. Chang, *Performance Guarantees in Communication Networks*, Springer Verlag, 2000.

Problems

17.1 *Video viewing models* ⋆

Fill in the table below indicating which video watching models are infeasible, or provide examples of companies that follow the model. Some rows have been filled out as an example.

Real-time or precoded	Streaming or download	Channelized or on-demand	Unicast or multicast	Companies
Real-time	Streaming	Channelized	Unicast	
Real-time	Streaming	Channelized	Multicast	
Real-time	Streaming	On-demand	Unicast	
Real-time	Streaming	On-demand	Multicast	
Real-time	Download	Channelized	Unicast	
Real-time	Download	Channelized	Multicast	
Real-time	Download	On-demand	Unicast	
Real-time	Download	On-demand	Multicast	
Precoded	Streaming	Channelized	Unicast	
Precoded	Streaming	Channelized	Multicast	
Precoded	Streaming	On-demand	Unicast	YouTube, Hulu, NBC, HBO Go
Precoded	Streaming	On-demand	Multicast	Infeasible (on-demand multicast)
Precoded	Download	Channelized	Unicast	
Precoded	Download	Channelized	Multicast	
Precoded	Download	On-demand	Unicast	
Precoded	Download	On-demand	Multicast	

17.2 *Compression-reliability tradeoff* ⋆

In this question, we will examine the tradeoff between compression and error resilience through a back-of-the-envelope calculation. Suppose we have 15 frames to transmit and two possible GoP structures: (A) IPB and (B) IPBBB. Suppose an I frame costs 7 kB, a P frame costs 3 kB, and a B frame costs 1 kB.

If an entire GoP is not received correctly, we assume that the GoP must be sent again. As our metric of error resilience, consider the expected number of bits that must be retransmitted at least once. The probability of dropping a frame is 1% and assumed to be independent. (These assumptions are made to simplify this homework problem. In a realistic setting, if a P or B frame in a

GoP is lost, the entire GoP does not need to be retransmitted. Loss is not independent and usually much less than 1%. And there should be many more frames.)

(a) In case A, the video frame structure is IPB/IPB/IPB/IPB/IPB. What is the total cost of the video in kB? What is the cost per GoP?

(b) What is the probability that an entire GoP is transmitted successfully in case A? What is the expected number of GOPs that are successful on the first transmission attempt of the entire video? What is the expected number of GoPs that must be retransmitted at least once? How much does the first retransmission cost in kB?

(c) Repeat (a) for case B, where the video frame structure is now IPBBB/IPBBB/IPBBB.

(d) Repeat (b) for case B.

(e) Compare you results from (a),(b),(c),(d) in terms of compressibility and cost of retransmission. What can you conclude?

17.3 *Playback buffer with random arrival time* ⋆⋆

We will look at a question similar to the example in Section 17.3.2 examining the latency-jitter tradeoff, but with a probabilistic packet arrival time. Suppose $V_1 = 0, V_2 = 1, V_3 = 2$, *i.e.*, a step function. Now the packets arrive independently at time A_1, A_2, A_3, where A_i is drawn randomly between $\bar{A}_i - 1$ and $\bar{A}_i + 1$. $\bar{A}_1 = 3, \bar{A}_2 = 4.2, \bar{A}_3 = 4.6$. What is the optimal playback time of the first packet p^* that minimizes latency but ensures that all packets were received with at least 95% probability?

17.4 *Round robin, weighted fair queuing, and priority queuing* ⋆⋆

Let us compare three resource allocation policies. Recall that

- Round robin simply gives each queue a turn to transmit a packet.
- Priority queuing allows the queue with the highest priority to be continuously serviced until it is empty.
- A particular implementation f weighted fair queuing looks at the head of each queue and transmits the packet that would finish transmission quickest under the **Generalized Processor Sharing** (GPS) scheme. GPS is an ideal fluid flow scheduler, and is defined as: if we have n queues with priority $p_1, p_2, \ldots, p_n$, then the bandwidth allocated to queue j per time step is $p_j / \sum_i p_i$.

Suppose we have queue A and queue B with packets arriving:

Time (s)	Queue A arrived packet size (MB)	Queue B arrived packet size (MB)
$t = 0$		3
$t = 1$	1	
$t = 2$	1	
$t = 3$		2
$t = 4$		
$t = 5$		4

Queue A has priority 1 and Queue B has priority 3 (higher number indicating higher priority). The outgoing link has bandwidth 1 MBps. Once a packet begins transmitting, it cannot be pre-empted by other packets. Fill in the following table for round robin scheduling, priority queuing, and weighted fair queuing.

Time (s)	Queue A departed packet size (MB)	Queue B departed packet size (MB)
$t = 0$		
$t = 1$		
$t = 2$		
$t = 3$		
$t = 4$		
$t = 5$		

17.5 *Leaky bucket and GPS* ⋆⋆⋆

A link becomes congested when packets arrive at a faster rate than the link can support, or packets arrive in large bursts. The **leaky bucket** queuing model is one way to solve this problem. In this model, there is a bucket that contains tokens. For traffic class k, the bucket has size B_k and tokens refill the bucket at a rate a_k. Packets wait in the queue and can only be released with a token from the bucket. Therefore, the maximum number of packets that leave the queue during time interval $[u, t]$ is $B_k + a_k(t - u)$. This is illustrated in Fig. 17.11.

Several leaky buckets drain into the same buffer. This buffer follows the Generalized Processor Sharing (GPS) model. In this model, each traffic class k has weight w_k. And $\rho_k = \frac{w_k C}{\sum_j w_j}$ is the instantaneous rate at which packets of traffic class k leave the GPS buffer, where C is the rate supported by the link out of the buffer, in bps. This is illustrated in Fig. 17.12.

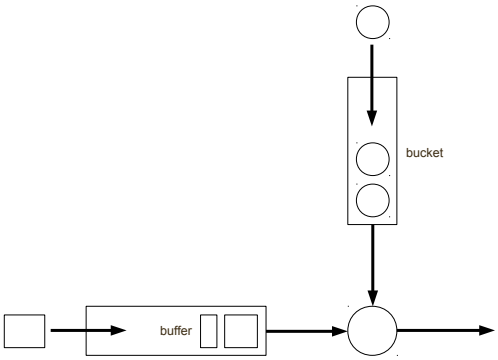


Figure 17.11 An illustration of a leaky bucket for admission control.

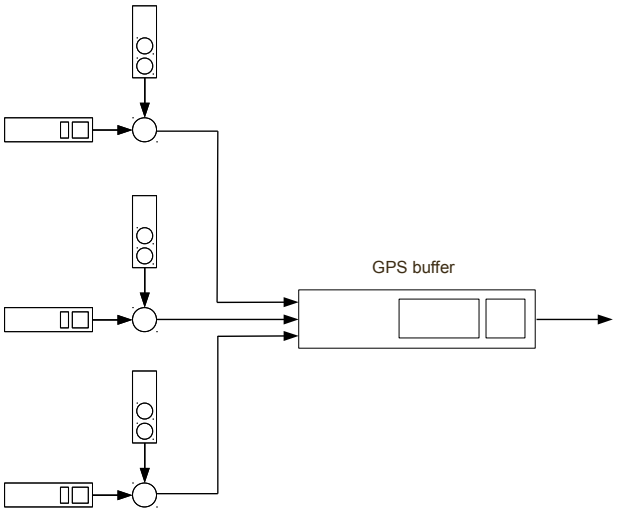


Figure 17.12 An illustration of Generalized Processor Sharing.

(a) During the time interval $[u, t]$, at least $\rho_k(t-u)$ packets leave the GPS buffer. With some abuse of notation, let B_t be the backlog of traffic class k in the GPS buffer at time t. Prove that the backlog of class k traffic in the buffer cannot exceed B_k if $\rho_k \geq a_k$. To start, assume that there is a time t when the backlog $B_t \geq B_k$. Consider the largest $u < t$ such that $B_u = 0$, and write the inequality relating the change in backlog size from time u to t.

(b) Prove that the delay experienced by a bit in class k in Fig. 17.12 cannot

exceed $\frac{B_k}{\rho_k}$ if $\rho_k \geq a_k$.

Now we compare the performance of GPS and WFQ. Let F_k denote the transmission time of packet k (the time packet k leaves the buffer) under WFQ. Similarly define G_k for GPS. Let L_k denote the size (in bits) of packet k. We will show that

$$F_k \leq G_k + \frac{L_{max}}{C}, \ \forall k. \tag{17.6}$$

GPS and WFQ both process packets at the same rate, so the total amount of bits in the system remains the same. Therefore, their busy and idle periods are the same, and we only need to show that the result holds for a single busy period. Assume $F_1 < F_2 < \ldots < F_K$ correspond to packets in one busy period of WFQ.

(c) Pick any $k \in \{1, 2, \ldots, K\}$ and find the maximum $m < k$ such that $G_m > G_k$. (If there is no such m, let $m = 0$.) This implies that $G_n \leq G_k < G_m, n \in P = \{m+1, m+2, \ldots, k-1\}$. Now consider the time $S_m = F_m - T_m$, when WFQ chose to transmit packet m. Show that the packets in set P must have arrived after S_m.

(d) Now consider the time interval $[S_m, G_k]$. Packets P arrived and departed during this interval. In addition, packet k was transmitted during this interval. Recall that the system is work conserving. Write an inequality relating the $[S_m, G_k]$ to the transmission times of packets in set P and use this to show the main result (**??**).

(e) Suppose that multiple leaky bucket queues are multiplexed to a single WFQ buffer, similar to Fig. 17.12. Combine (b) and (d) to show that the maximum delay experienced by a packet of class k in this scenario is $\frac{B_k}{\rho_k} + \frac{L_{max}}{C}$.

18 Why is WiFi faster at home than at a hotspot?

A crude answer is that the interference management method in WiFi does not scale well beyond several devices sharing one access point. When the crowd is big, the "tragedy of the commons" effect, due to mutual interference in the unlicensed band, is not efficiently mitigated by WiFi. To see why, we have to go into the details of WiFi's medium access control in the link layer of the layered protocol stack.

18.1 A Short Answer

18.1.1 How is WiFi different from cellular

Since its first major deployment in the the late 1990s, WiFi hotspots have become an essential feature of our wireless lifestyle. There were already more than 1 billion WiFi devices around the world by 2010, and hundreds of millions added each year. We use WiFi at home, in office, and around public hot spots like those at airports, in coffee shops, or even around street corners.

We all know WiFi is often faster than 3G cellular, but you cannot move around too fast on WiFi service or be more than 100m away from an Access Point (**AP**). We have seen many letters attached to 802.11, like 802.11a,b,g,n, shown on the WiFi AP boxes you can buy from all the electronic stores, but may not appreciate why we are cooking an alphabet soup. We have all used hotspot services at airports, restaurants, hotels, and perhaps our neighbor's WiFi (if it does not require a password), and yet have all been frustrated by the little lock symbol next to many WiFi network names that our iPads can see but not use.

When Steve Jobs presented iPhone 4 in a large auditorium, that demo iPhone could not get on the WiFi. Jobs had to ask all the attendants to get off the WiFi, and his iPhone managed to get on the WiFi afterwards. Is there some kind of limit as to how many users a given WiFi hotspot can support?

In June 2011, the Korean government announced a plan to cover the entire city of Seoul, including every corner of every street, with WiFi service by 2015. If many WiFi users aggregate around one popular street corner, how many hotspots need to be created to take care of that demand? And more importantly, how can this traffic be backhauled from the WiFi air-interface to the rest of the Internet?

At home, a residential gateway of the Internet access is often connected to a

WiFi AP, which provides the in-home wireless connectivities to desktops, laptops, game consoles, phones, tablets, and even TV's set-top boxes using the latest high speed WiFi variant. As each home adds more WiFi devices, will the quality of connection be degraded, especially in high-rise multi-tenant buildings?

The exact answers to these questions are not available, although we do know quite a bit about WiFi architecture and performance. Officially, WiFi should be called the **IEEE 802.11** standard. It is part of the 802 family of standards on Local Area Networks (**LAN**) prescribed by IEEE. The .11 part of the family focuses on wireless LAN using the **unlicensed spectrum**.

You must have a license from the government to transmit in the frequency bands used in all generations of cellular networks. For 3G and 4G, governments around the world sold these spectral resources in the air for tens of billions of dollars, sometimes through auction as we saw in a homework problem in Chapter 2. It tries to avoid too many transmitters crowding and jamming into those frequency bands.

In contrast, governments around the world also leave some bands as unlicensed and free, as long as you do not operate at too high a transmit power. For example, the Industry, Science, and Medical (**ISM**) frequency ranges in the S-band around 2.4-2.5 GHz and in the C band around 5.8 GHz. It was originally designed for use in the three fields as suggested by the name ISM. But the most widely used appliance in the ISM S-band, other than WiFi, is actually the microwave oven. That band works well to excite water molecules. There are also other wireless communication devices running on bluetooth, zigbee, *etc.* sharing the same ISM band. Handling interference on an unlicensed spectrum is a major challenge.

In the mid 1990s, as the 2G cellular industry took off, people started wondering if they could create an alternative in wireless networks: use the small amount of power allowed in ISM and short-range communication (around 100 meters outdoors, one order of magnitude smaller transmission radius than cellular) for mostly stationary devices. Because this is not a single provider network, there needed to be an industry forum to test inter-operability of all the devices. It was established in 1999 as Wi-Fi Alliance, where Wi-Fi stands for "Wireless Fidelity" and is a catchier name than "IEEE 802.11b".

There have been many versions of WiFi standards, created by the IEEE 802 organization, starting with 802.11b that uses the 2.4 GHz band and can transmit up to 11 Mbps. This was followed by two other main versions: 802.11g that uses a more advanced physical layer coding and modulation to get to 54 Mbps in the 2.4 GHz band, and 802.11a that can also achieve up to 54 Mbps in the 5.8 GHz band. Some of these standards divide the frequency band into smaller blocks in Orthogonal Frequency Division Multiplexing (**OFDM**). In contrast to anti-resource-pooling in Paris Metro Pricing in Chapter 12, this resource fragmentation is motivated by better spectral efficiency as signals on smaller chunks of frequency bands can be more effectively processed. More recently, 802.11n uses multiple antennas on the radios to push transmission rate to over 100 Mbps. Aug-

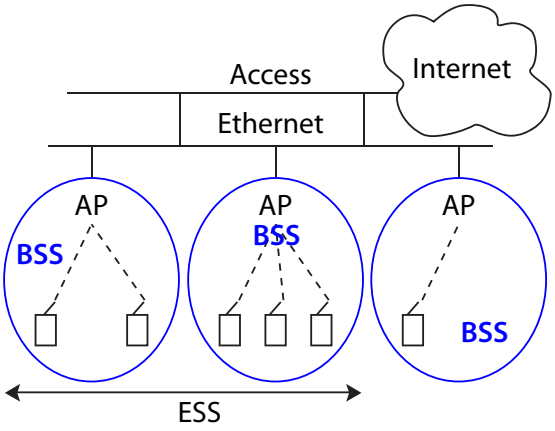


Figure 18.1 A typical topology of WiFi deployment. The air interface provides bidirectional links between the APs and end user devices. Each BSS has an AP. A collection of BSS that can readily support handoff is called an ESS. The air interface is connected to a wireline backhaul, often an Ethernet, which is in turn connected to the rest of the access network and further to the rest of the Internet.

menting the channel width to 40 MHz also helped increase the data rate. We will discuss OFDM and multiple antenna systems in Advanced Material.

There have also been many supplements that improve specific areas of WiFi operation. For example, 802.11e improved the quality of service in its medium access control, a topic we will focus on in this chapter. 802.11h improved encryption and security, a major issue in the early days of WiFi. 802.11r improved the roaming capability in WiFi to support, to some degree, the mobility of people holding WiFi devices.

Even though the nature of spectral operation is different, WiFi does share a similar topology (Figure 18.1) with cellular networks, except this time it is not called a cell (since there is often no detailed radio frequency planning before deployment), but a Basic Service Set (**BSS**). In each BSS there is an AP rather than a Base Station. A collection of neighboring BSSs may also form an Extended Service Set (ESS).

When your laptop or phone searches for WiFi connectivity, it sends probe messages to discover which APs are out there in its transmission range, and shows you the names of the BSSs. You might want to connect to the BSS, but if it is password protected, that means your device can only associate with the AP if you have the password to authenticate your status, *e.g.*, a resident in the building if the AP is run by the building owner, an employee of the company if the AP is run by the corporation, or a paying customer if the AP is part of the WiFi paid service offered by a wireless provider. Increasingly you see more WiFi deployment, but *free* WiFi's availability may be on the decline.

These APs are tethered to a backhauling system, often a wireline **Ethernet**

(another, and much older, IEEE 802 family member) that connects them further to the rest of the Internet. This is conceptually similar to the core network behind the base stations in cellular networks, although the details of mobility support, billing, and inter-BSS coordination are often much simpler in WiFi.

If the channel conditions are good, *e.g.*, you are sitting right next to the WiFi enabled residential gateway at your home and no one else's signal interferes with yours, the data rate can be very impressive, especially if you are using 802.11n. It is faster than 3G, and probably even faster than the DSL or fiber access link that connects the residential gateway to the rest of the Internet. But if you sit outside the limited range of the AP or you start moving across boundaries of an ESS, you can easily get disconnected. And if you share the air with 10 or so other WiFi devices, the speed can drop substantially as you may have experienced in a crowded public WiFi hotspot.

There is actually also a peer-to-peer mode in 802.11 standards, the *infrastructureless*, ad hoc mode. WiFi devices can directly communicate with each other without passing through any fixed infrastructure like APs. You see this option when you configure the WiFi capability on your computers. But very few people use this mode today, and we will only talk about the infrastructure mode with APs.

18.1.2 Interference management in WiFi

Summarizing what we have talked about so far: WiFi is an evolving family of standards that enables short-range wireless communication over the ISM unlicensed bands for largely stationary devices. In contrast to cellular networks, WiFi networks are often deployed with very limited planning and managed only lightly, if at all.

It is quite a different type of wireless networking compared to cellular, and its performance optimization requires some different approaches. Whether a WiFi hotspot works well or not really depends on how effectively such optimizations are carried out. We focus on the air-interface part between the AP and the devices (the terminology **station** covers both), even though the backhaul part could also become a bottleneck (*e.g.*, when the DHCP server has a bug and cannot keep track of the IP addresses assigned to the devices, or simply because the backhauling capacity is limited.)

The first set of performance tuning involves the correct selection of the AP, of the channel, and of the physical layer transmission rate:

- *AP association*: A WiFi device has to regularly scan the air and then associate with the right AP, *e.g.*, the one that offers the best SIR (and, of course, authenticates the device).
- *Channel selection*: The overall ISM frequency band is divided into channels. In 802.11b in the U.S., for example, each channel is 22 MHz wide and 5 MHz apart from the neighboring channels. As shown in Figure 18.2, only those

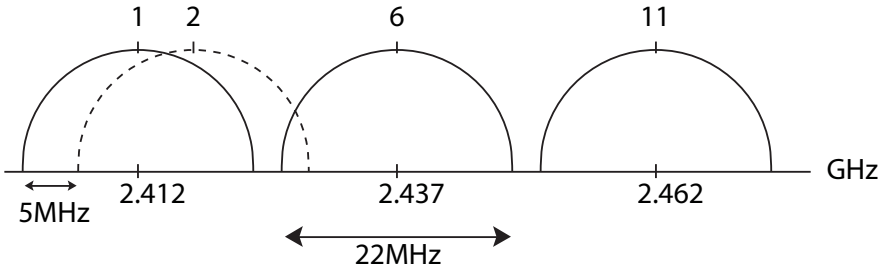


Figure 18.2 The 802.11b spectrum and channels. There are 11 channels in the U.S. Each channel is 22 MHz wide, and 5MHz apart from the neighboring channels. Therefore, only 3 channels, Channel 1, Channel 6, and Channel 11, are non-overlapping.

channels that are 5 channels apart are truly non-overlapping. So if you want to have three devices on non-overlapping channels, the only configuration is for each of them to choose a different channel from among Channels 1, 6, and 11. Many WiFi deployments just used the default channel in each AP. If they are all on channel 6, interference is created right there.

- *Rate selection*: We mentioned that each of 802.11abgn can transmit *up to* a certain data rate. That is assuming a really good channel, with no interference, and no mobility. In many WiFi hotspots, channel condition fluctuates and interferers come and go. The maximum rate is rarely achieved, and the AP will tell the devices to backoff to one of the lower rates specified, *e.g.*, all the way down to 1 Mbps in 802.11b, so that the decoding is accurate enough under the lower speed. A device knows it is time to fallback its rate to the next lower level if its receiver's SIR is too low for the current rate, or if there have been too many lost frames.

Suppose your WiFi device gets the above three parameters right. We need to take care of interference now. When two transmitters are in interference range of each other, and they both transmit a frame at similar times (t_1, t_2), these two frames collide. There are three possible outcomes of a collision:

- Both frames are lost: neither receiver can correctly decode the intended frame.
- The stronger frame is properly received, but the weaker frame is lost: here, "strength" refers to SIR. This is called **capture**.
- Both frames are properly received. This is called **double capture**.

Now, which outcome will prevail depends, quite sensitively, on the following factors:

- How long the frames overlap (based on their timing difference $t_1 - t_2$, frame sizes, and transmission rates).

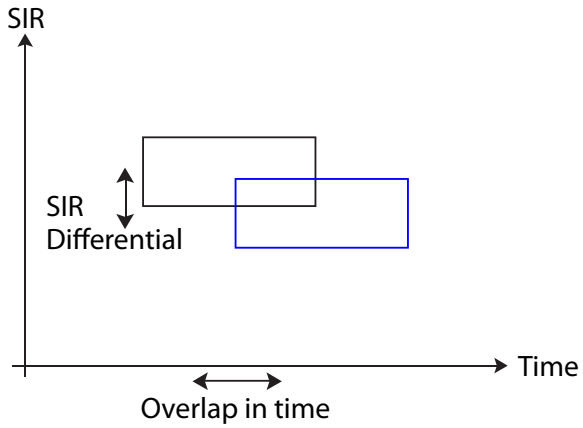


Figure 18.3 An energy-time diagram of two colliding frames. If collision is defined as two frames overlapping in their transmission time, the outcome of a collision depends quite sensitively on several factors: how long is the overlap, how big is the differential in the received SIRs, and how large an SIR is needed for proper decoding at each receiver.

- How big the differential in SIR between the two frames is (based on channel conditions and transmit powers). This is illustrated in Figure 18.3.
- How large an SIR is required for proper decoding at the receiver (based on transmission rates, coding and modulations, and receiver electronics).

It is an unpleasant fact that wireless transmissions may interfere, since wireless transmission is energy propagating in the air. It is a particularly challenging set of physics to model, since collision is not just one type of event. In the rest of the chapter, we will simply assume that when collision happens, both frames are lost.

We have been discussing interference's impact on two frames in quite some detail. Compared to power control in Chapter 1 for cellular networks, WiFi also uses a fundamentally different approach to manage interference, due to its much smaller cell size, the typical indoor propagation environment, a much smaller maximum transmit power allowed and more uncontrolled interferences in an unlicensed band. Instead of adjusting transmit powers to configure the right SIRs, WiFi tries to avoid collision altogether, through the mechanisms of **medium access control**.

Think of a cocktail party again like in Chapter 1, where guests' voices overlap in the air. With enough interference you cannot understand what your friend is trying to say. Cellular network power control is like asking each guest to adjust her volume without running into an arms race. WiFi medium access control is like arranging the guests to talk at different times.

You can either have a centralized coordinator to assign different timeslots for

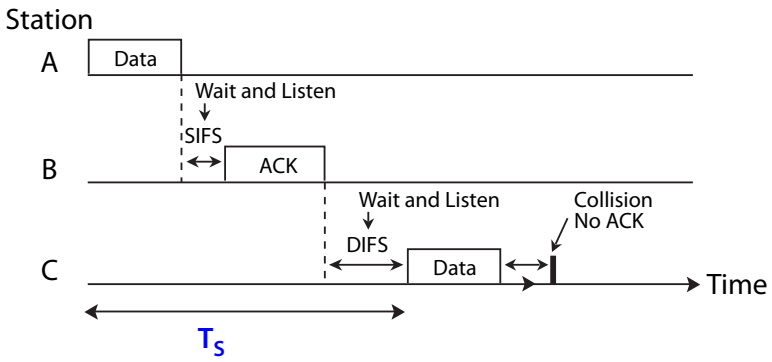


Figure 18.4 A timing diagram of basic WiFi transmissions. A station can be either a user device or an AP. First the transmitter of a session, station A, sends a data frame to its intended receiver, station B. Then after a very short period of time with a predetermined length called SIFS, B sends an acknowledgment frame back to A. After waiting for another slightly longer period of time called DIFS, other nodes like station C can start sending new data frames. In the above example, node C's packet collides with some other packet transmitted by, say, station D.

each guest to talk (scheduling), or you can ask each of them to obey a certain procedure in deciding locally when to talk and how long to talk (random access). We call these the Point Coordination Function (**PCF**) and the Distributed Coordination Function (**DCF**), respectively, in WiFi. PCF, like token ring in the wireline Ethernet protocol, represents centralized control (and for dedicated resource allocation). It is complicated to operate and rarely used in practice. DCF, in contrast, enables shared resource allocation. As you might suspect of any distributed algorithm, it can be less efficient, but is easier to run. In practice, DCF is used most of the time. We will discuss WiFi's DCF, which is a particular implementation of the Carrier Sensing Multiple Access (**CSMA**) random access protocol.

The basic operation of CSMA is quite simple and very intuitive. Suppose you are a transmitter. Before you send any frame, you regularly listen to the air (the part of the spectrum where your communication channel lies). This is called **carrier sensing**. As Figure 18.4 illustrates, every transmitter must obey a wait-and-listen period before it can attempt transmission. If the channel is sensed as busy (someone is using the medium to transmit her frames), you just stay silent. But if it is idle (no one is using it), you can go ahead and send a sequence of frames. You might want to send a lot of frames in a row, so you can send a control message declaring how long you intend to use the channel. Of course channel holding time has some upper bounds, just like treadmill sharing in the gym.

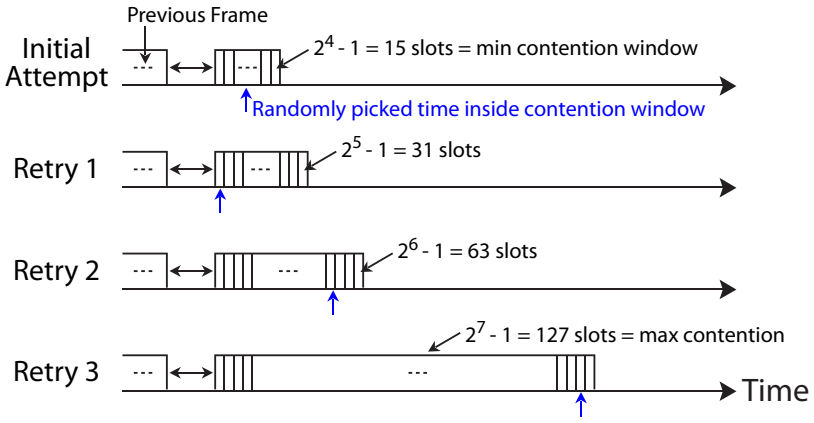


Figure 18.5 Exponential backoff in DCF. There are two key ideas. First, when two frames collide, both need to back off. In order to avoid both picking the same time to retransmit, each picks a random point over the contention window to retransmit. Second, if collisions continue, each sender needs to back off more. Doubling the contention window is a reasonable way to increase the degree of backing off. A homework problem will explore this further. The minimum window size is W_{min}, and the maximum number of backoffs allowed (before the frame is discarded) is B.

But if your frame collides with some other frames when you try to send it, your receiver will not get it (since we assumed a collision kills both frames). You will not get her acknowledgement. So you know you suffered a collision, and you need to *backoff*. This WiFi backoff is similar to TCP backoff by halving the congestion window in Chapter 14: you double the **contention window** in WiFi. And then you draw a random number between now and the end time of the contention window. That will be your next chance of sending the lost frame.

The protocol description above might sound unmotivated at first. But there are actually two clever ideas of distributed coordination here: randomization and exponential backoff.

First, if stations A and B have their frames collide at one time, you do not want them to backoff to a common time in the future: there will be just another collision. They need to *randomly* backoff to minimize the chance of hitting each other again. Of course, they may so happen pick exactly the same timeslot again, but that is the price you pay for a distributed coordination.

Second, if frames keep colliding, you know the interference condition is very bad, and you, as well as all those stations experiencing persistent collisions of their frames, should start backing off more. Linearly increasing the contention window size is one option, but people thought that would not be aggressive enough. Instead, WiFi mandates *multiplicatively* backing-off. Since the multiplicative factor is 2, we call it **binary exponential backoff**. This is similar to the multiplicative decrease of the congestion window size in TCP. As illustrated

in Figure 18.5, when your contention window exceeds a *maximum value*, *i.e.*, you have been backing off through too many stages, you should just discard that frame altogether and report the loss to upper layer protocols so that they can try to fix it. The contention window may also have a *minimum value*, in which case a sender has to wait before its first attempt to send a frame.

So far so good. But it is another unpleasant fact of wireless transmission that sensing range is *not* the same as interference range: stations A and B might collide but they cannot hear each other, like in Figure 18.6. This is the famous **hidden node** problem, one of the performance bottlenecks of WiFi hotspots. This problem does not arise in TCP congestion control.

But there is a clever solution, using a little explicit message passing this time, to help navigate through this challenging interference problem. It is called **RTS/CTS**. When station A wants to send a frame, it first sends a short control message called Request To Send (RTS). All stations within the sensing range of A receive that message, and each of them in turn sends a short control message called Clear To Send (CTS). All stations within sensing range of them receive the CTS and refrain from transmitting any frames in the near future. Of course station A itself also gets the CTS message back, and when it sees that CTS, it knows all those hidden nodes have also received the CTS and thus will not send any frames now. At that point, station A sends the actual frames.

As Figure 18.7 illustrates, the brief period of idle time in between an RTS and the CTS is shorter than the wait-and-listen time between data transmission. This is yet another clever idea in distributed coordination over wireless channels. By creating multiple types of wait-and-listen intervals, those transmissions that only need to obey a shorter wait-and-listen interval are essentially given higher *priority*. They will be allowed to send before those who must obey a longer wait-and-listen period.

RTS/CTS is not a perfect solution either, *e.g.*, RTS and CTS frames themselves may collide with other frames. But still, with the RTS/CTS message passing protocol, together with prioritization through different wait-and-listen intervals, distributed transmission through randomized transmit timing, and contention resolution through exponentially backed-off content window, we have a quite distributed MAC protocol that enables the success of WiFi hotspots as they scale up.

We will see several other wireless peculiarities and their effects on both efficiency and fairness in Advanced Material. But first let us work out the throughput performance of WiFi devices in a hotspot running DCF.

18.2 A Long Answer

Random access offers a complementary approach to power control as an interference management method. To be exact, there is a power control functionality

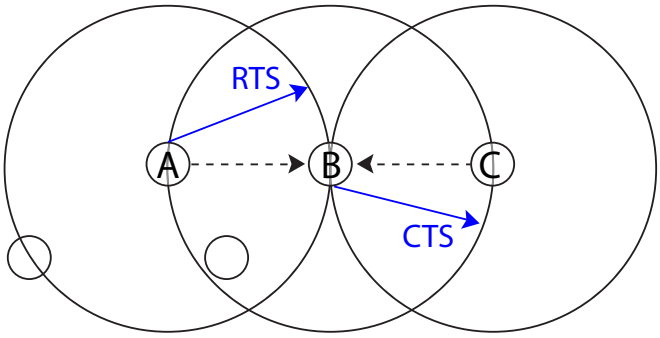


Figure 18.6 The hidden node problem: Stations A and C's transmissions to station B interfere with each other, but cannot sense each other. Dotted lines denote sensing/transmission range. RTS/CTS is a message passing protocol to help resolve the hidden node problem. Node A first sends an RTS. Upon hearing the RTS, all nodes (including node B) send a CTS. Upong hearing the CTS, all nodes (including node C) remain silent for a period of time, except node A itself who initiated the RTS in the first place. Node A now knows it is safe to send the actual data frames without worrying about hidden nodes.

in WiFi too, but it is mostly for conforming to unlicensed band energy limit and for saving battery energy, rather than to manage interference.

While power control can be analyzed through linear algebra (and some game theory and optimization theory) as presented in Chapter 1, random access involves probabilistic actions by the radios and its performance analysis evaluation requires some probability theory.

CSMA in WiFi DCF is not particularly easy to model either: collision of frames depends on the actions by each radio, and the history of binary exponential backoff couples with the transmission decision at each timeslot. A well-known performance analysis model uses a two-dimensional Markov chain, which exceeds our prerequisite of basic probability concepts and becomes too complicated for this chapter. There is a simplified version that uses very simple arguments in basic probability and a little bit of handwaving to get the gist out of the complicated derivation. And that is the approach we will follow.

18.2.1 Expressing S as a function of τ

The expected throughput S bps for CSMA random access in WiFi DCF is defined as:

$$S = \frac{\text{average number of bits transmitted successfully in a timeslot}}{\text{average length of a timeslot}}. \quad (18.1)$$

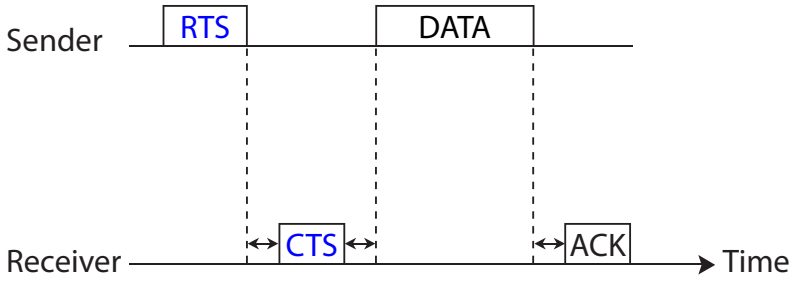


Figure 18.7 A timing diagram of RTS/CTS in WiFi DCF to help mitigate the hidden node problem. The durations of time between RTS and CTS, and between CTS and Data frames, are smaller than the period of time that other nodes need to wait for a clear channel before transmitting. This timing difference effectively provides the priority of CTS and the following Data traffic over competing traffic.

(1) First we examine the average number of bits transmitted successfully in a timeslot. It can be expressed as the product of three numbers:

$$P_t P_s L,$$

where P_t is the probability that there is at least one (could be more) transmission going on, P_s is the probability that a transmission is successful, and L is the average payload length (measured in bytes or bits).

Let τ be the probability that a station transmits at a given timeslot. Then we know

$$P_t = 1 - (1 - \tau)^N, \tag{18.2}$$

since P_t equals 1 minus the probability that no station transmits, which is in turn the product of the probabilities that each station does not transmit: $1 - \tau$, over all the N stations. This assumes that the stations make transmission decisions independently. This is another example of the *diversity gain* of network effect: except now it is about the probability of a good event (a transmission gets through without collision) rather than a bad event (some link fails).

We also know that

$$P_s P_t = N\tau(1 - \tau)^{N-1}, \tag{18.3}$$

since the left hand side is the probability that there is a successful transmission at a given timeslot. For each station, that should be the probability that it transmits (τ) but none of the other stations does $(1 - \tau)^{N-1}$. For the whole network, it is $N\tau(1 - \tau)^{N-1}$.

(2) Now we examine the average length of a timeslot. That depends on what happens at that timeslot. There are three possibilities, as illustrated in Figure 18.4:

- If there is no transmission at all, the probability of which is $1-P_t$, the timeslot is a backoff slot with length T_b.
- If there is a transmission but it is not successful, the probability of which is $P_t(1-P_s)$, the timeslot is a collision slot with length T_c.
- If there is a transmission and it is successful, the probability of which is P_tP_s, the timeslot is a successful slot with length T_s.

In summary, if we know how to compute τ, we can compute P_t as well as P_s, thus the expected throughput S:

$$S = \frac{P_tP_sL}{(1-P_t)T_b + P_t(1-P_s)T_c + P_tP_sT_s}. \tag{18.4}$$

Among the quantities shown above, N, L, T_b, T_c and T_s are constants. So we just need to compute τ.

18.2.2 Computing τ

First, we can express c, the probability that a frame transmitted (by a particular station, say station A) collides with frames from other stations, as a function of τ. Assuming that collision probability is independent of backoff stage, we have

$$c = 1 - (1-\tau)^{N-1}, \tag{18.5}$$

since c is simply the probability that at least one (could be more) of the other $N-1$ stations transmits in addition to station A. So (18.5) follows the same argument behind the P_t expression in (18.2).

Suppose we can also do the reverse: express τ as a function of c. In that case, we can substitute c as a function of τ (18.5) and numerically solve for τ. So now everything boils down to the following: Find τ (the probability a station transmits) in terms of c (the probability that a transmitted frame collides).

Since there are many backoff stages indexed by i, each with a contention window doubling the previous stage's, we look at the joint probability that a station transmits while at backoff stage i. We can express this joint probability in two ways:

$$\text{Prob(transmit)Prob(in backoff stage } i|\text{transmit)}$$

and

$$\text{Prob(in backoff stage } i\text{)Prob(transmit|in backoff stage } i\text{)},$$

and the two expressions above must be the same. We give shorthand notation

to the above expressions: $\tau P(i|T)$ and $P(i)P(T|i)$. So we have the following *Bayesian* expression:

$$\tau \frac{P(i|T)}{P(T|i)} = P(i).$$

Summing over all the i from 0 (no backoff) to B (the maximum number of backoffs allowed) on both sides, we have

$$\tau \sum_{i=0}^{B} \frac{P(i|T)}{P(T|i)} = \sum_{i=0}^{B} P(i) = 1. \tag{18.6}$$

If we can express $P(i|T)$ and $P(T|i)$ in terms of c, we have an expression for τ in terms of c, which completes our derivation for S.

Computing $P(i|T)$ is easy: if a station transmits at the backoff stage i, it must have suffered i collisions in the past and 1 non-collision now. We can just write down that probability, making sure that it is normalized:

$$P(i|T) = \frac{c^i(1-c)}{1-c^{B+1}}.$$

Computing $P(T|i)$ is also easy (with a little handwaving): the transmit slot is one slot on its own, so the lifetime of backoff stage i, on average, is $1 + T_i$ slots. Here, T_i is the average value of backoff counter at stage i:

$$T_i = \frac{1}{2}(0 + 2^i \mathrm{W}_{min}),$$

where W_{min} is the minimum contention window size. Obvious from the above, we assumed that the timeslot to transmit is picked randomly between "right now" and "the upper limit of binary exponential backoff". The actual contention window size W is 2 raised to some integral power then minus 1. We ignore the 'minus 1' part for simplicity. Now we have

$$P(T|i) = \frac{1}{1+T_i}.$$

Finally, we can put everything back together into (18.6), and have the following expression of τ in terms of c:

$$\tau = \frac{1}{1 + \frac{1-c}{1-c^{B+1}} \sum_i c^i T_i}. \tag{18.7}$$

Therefore, just plug (18.5) into (18.7). We can solve for τ numerically, as a function of number of backoff stages B and the minimum contention window size W_{min}.

18.2.3 Putting everything together

Once τ is found, by (18.2, 18.3) we have P_t, P_s as well, and can in turn compute S (18.4) in terms of the WiFi DCF protocol parameters: L, B, W_{min}, the length of the three types of timeslots T_b, T_c, T_s, and the number of stations N.

18.3 Examples

18.3.1 Parameters and timeslots

Before doing any calculations, we need to specify the DCF protocol parameters L, B, W_{min} and the timeslot lengths T_b, T_s, T_c.

We consider DCF being used in 802.11g at 54Mbps. By the protocol specifications, the relevant timing parameters are

$$\begin{aligned} \text{slot time} &= 9\mu\text{s} \\ \text{SIFS} &= 10\mu\text{s} \\ \text{DIFS} &= \text{SIFS} + (2 \times \text{ slot time}) = 28\mu\text{s} \end{aligned}$$

By the specifications $W_{min} = 15$. Also, we set $L = 8192$ bits and $B = 3$ as the default values. We will also later sweep their values to explore the impact.

Duration of T_b. It is simply the length of DIFS, *i.e.*, $T_b = 28$ μs.

Duration of T_s. A successful transmission consists of the transmission of a data frame from the sender and the transmission of an ACK frame from the receiver, together with appropriate spacing: [data frame] + SIFS + [ACK frame] + DIFS.

A data frame consists of a 16μs PHY layer preamble, a 40-bit PHY header, a 240-bit MAC header, the L-bit payload and a 32-bit CRC code. The protocol specifications state the PHY header is further split and sent at two different rates: the first 24 bits at 6 Mbps to be more robust to signal errors at the expense of a lower rate, and the remaining 16 bits at 54 Mbps. Hence the time to send a data frame is

$$16 + \frac{24}{6} + \frac{16 + 240 + 32}{54} + \frac{L}{54}\mu\text{s} = 25.33 + \frac{L}{54}\mu\text{s}$$

Similarly, an ACK frame consists of a 16μs PHY layer preamble, a 40-bit PHY header (again split and sent at different rates) and a 112-bit MAC layer frame (header and CRC). Therefore, the time to send an ACK frame is

$$16 + \frac{24}{6} + \frac{16 + 112}{54} = 22.37 \ \mu\text{s}$$

and we have $T_s = 25.33 + L/54 + 10 + 22.37 + 28 = 85.70 + L/54$ μs.

Duration of T_c. When there is a collision, the sender has to wait for the full duration of the ACK frame before deciding there is a collision (by absence of an ACK), so $T_c = T_s = 85.70 + L/54$ μs.

18.3.2 Throughput

First, plugging in $T_i = (0 + 2^i W_{min})/2$ into (18.7), we solve numerically for τ in

$$\tau = \frac{1}{1 + \frac{1-c}{1-c^{B+1}} \sum_{i=0}^{B} c^i 2^{i-1} W_{min}}, \tag{18.8}$$

where $c = 1 - (1 - \tau)^{N-1}$.

Then we plug the solution of τ into the formula of S:

$$S = \frac{N\tau(1-\tau)^{N-1}L}{(1-\tau)^N T_b + [(1-(1-\tau)^N) - N\tau(1-\tau)^{N-1}]T_c + N\tau(1-\tau)^{N-1}T_s} \tag{18.9}$$

while varying the values of N, B, W_{min} and L. Unless specified, their default values are $N = 5$, $B = 3$, $W_{min} = 15$, $L = 8192$.

Varying N (Figure 18.8). This is the key graph we have been looking for in this chapter, quantifying the impact of the crowd size in the tragedy of the WiFi commons.

- If N increases, the per station throughput $S(N)/N$ decreases because more stations are competing for the same channel. The drop is quite sharp from $N = 2$ to $N = 15$, and the throughput value becomes quite low, below 2 Mbps, once N becomes 10. This highlights the inscalability of CSMA.
- The aggregate throughput $S(N)$ initially increases for small N because stations are fundamentally limited by their exponential backoff mechanism. Even when a station has no competitors for the channel, it still picks a transmit slot uniformly at random between 0 and W_{min}, and this leads to inefficiency. Adding stations helps in utilizing the channel, given there is not much contention (when N is small).
- Despite the advertised throughput of 54 Mbps, the actual maximum throughput is around 25 Mbps. This is because only the payload is sent at 54 Mbps, and there is a significant overhead in the PHY layer (*e.g.*, preamble and header being sent at 6Mbps).

Varying W_{min} (Figure 18.9). A smaller W_{min} leads to higher aggressiveness of a station in using the channel.

- When the channel is not congested ($N = 5$), it helps to be more aggressive.
- When the channel is congested ($N = 20$), being aggressive leads to more collisions, so it is better to choose a larger W_{min} (to be less aggressive).

Varing B (Figure 18.10). A larger B tends to increase the average contention window size (*i.e.*, become less aggressive). Hence the observation is similar to that of W_{min}.

- When $N = 5$, increasing B does not help at all.
- When $N = 20$, being less aggressive (increasing B) helps a lot.

Varying L (Figure 18.11). Payload size relative to the overhead size also matters.

- Increasing the payload size helps because less overhead is incurred.
- But this model does not capture the effect of a frame with a larger payload having a larger collision probability. In reality we expect the throughput to increase until it reaches an optimum value, and then decrease.

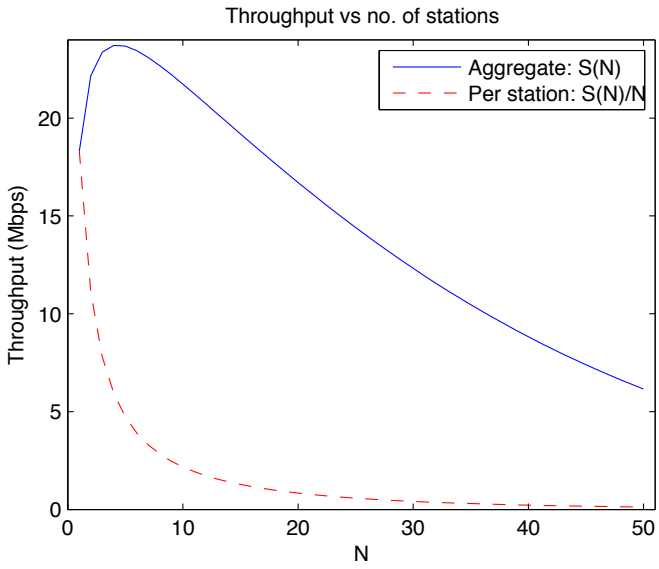


Figure 18.8 As the number of users increases, the total throughput drops. Per-user average throughput drops to small values around 2Mbps when there are 10 or more users. The rapid drop of the per-user average throughput even for small N highlights the inscalability of CSMA.

18.4 Advanced Material

18.4.1 Impact of interference topology

We saw that the DCF deploys several smart ideas to enable a reasonable medium access control with a small amount of message passing overhead. But it is also well-known that the DCF can sometimes be inefficient and is often unfair, as shown in a homework problem. We have encountered quantification of fairness many times by now, and in Chapter 20 we will go to great length just on fairness.

Part of the reason behind DCF's inefficiency and unfairness stems from the tough reality of sensing and interference in wireless networks. The hidden node problem arises from the *difference* between sensing range and interference range, as we saw. Sensing range is also *asymmetric*: one (transmitter, receiver) pair might be able to sense and interfere with another, but not the other way around.

An example is the flow-in-the-middle topology in Figure 18.12(a). The session in the middle can sense the other two sessions on each side, and has to be polite in waiting-and-listening to both. In contrast, either of the side sessions can only sense the middle session, and has less wait-and-listen to do. A proportionally fair solution should give the two side sessions the same throughput and the middle session half of that. But WiFi DCF almost certainly will starve the middle session to near zero throughput.

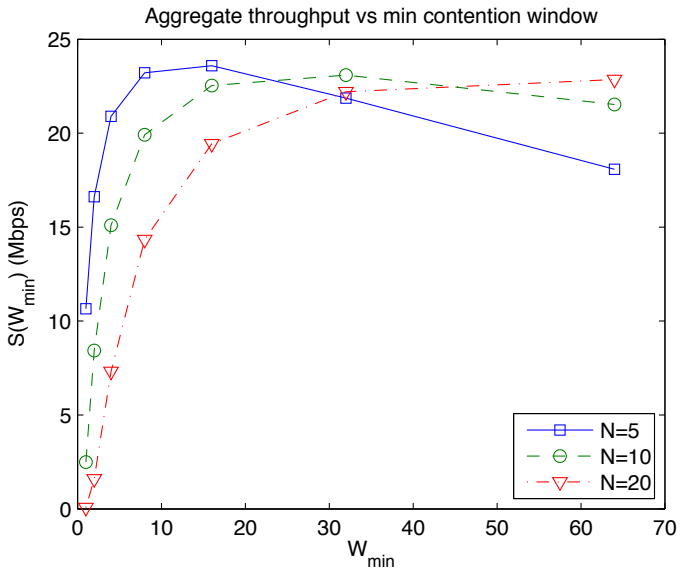


Figure 18.9 Increasing the minimum contention window size initially has a positive effect on average throughput, as contention is less aggressive. Past some threshold value, making the minimum window size larger hurts throughput (via its overhead in waiting) more than it helps. The more users there are, the higher this threshold value.

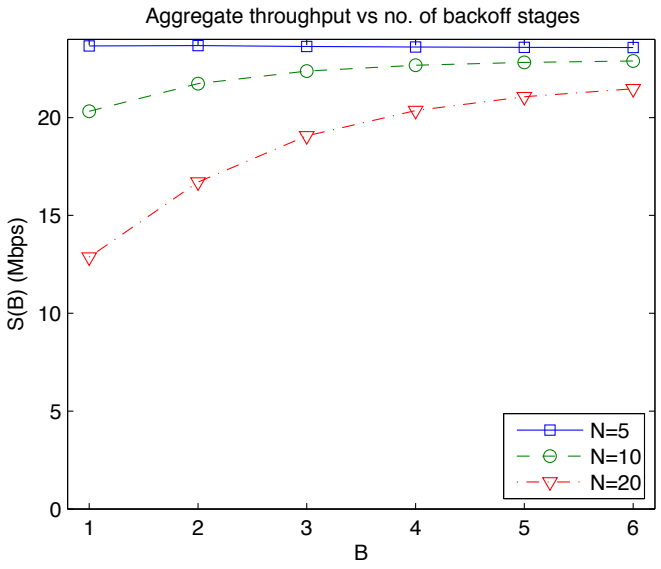


Figure 18.10 As B increases, more backoff stages are allowed, and the average backoff rises. More conservative contention increases average throughput (but also would increase latency).

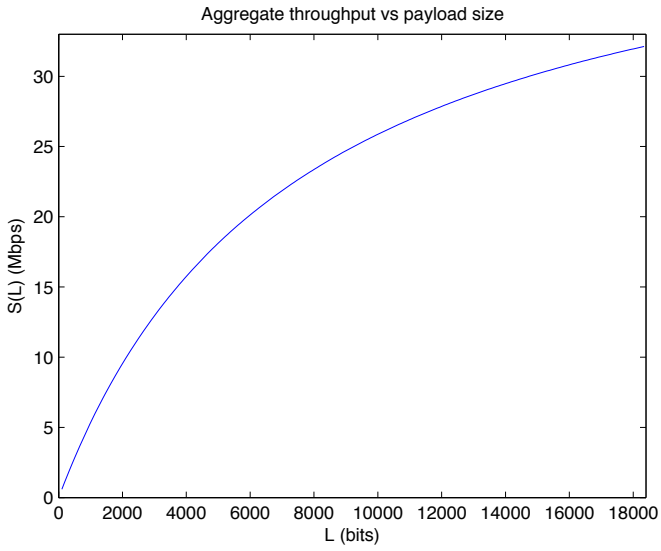


Figure 18.11 Larger payload means smaller percentage of overhead. Throughput naturally increases. If the model had incorporated the fact that a larger payload also increases the chance of collision, the curve would not have been monotonically increasing.

Another example is the asymmetric sensing and interference topology in Figure 18.12(b). Session A cannot sense session B, and consequently they often collide. In a proportionally fair allocation, both sessions should have the same throughput. But in WiFi DCF, session A's throughput will be substantially less than that of session B.

There have been hundreds of papers on improving WiFi DCF, mostly because it is much more feasible to experiment in unlicensed band than licensed band in university labs. One of the more recent improvements of DCF comes from an optimization model of CSMA parameter adaptation: each transmitter changing aggressiveness of contention and of channel holding duration by observing its recent throughput, balancing between demand of capacity and supply of allocated capacity (over the recent timeslots), similar to similar balancing acts in TCP congestion control or in cellular power control. Researchers then study if the distributed action without any message passing (beyond RTS/CTS) can converge to the optimal throughput for all. Hidden node, flow-in-the-middle, and asymmetric sensing are three of the *atomic topologies* that one needs to test for any of these alternatives to CSMA used in today's WiFi. The complicated nature of sensing (imperfect and asymmetric) and of interference (Figure 18.3) also need to be carefully incorporated in the evaluation.

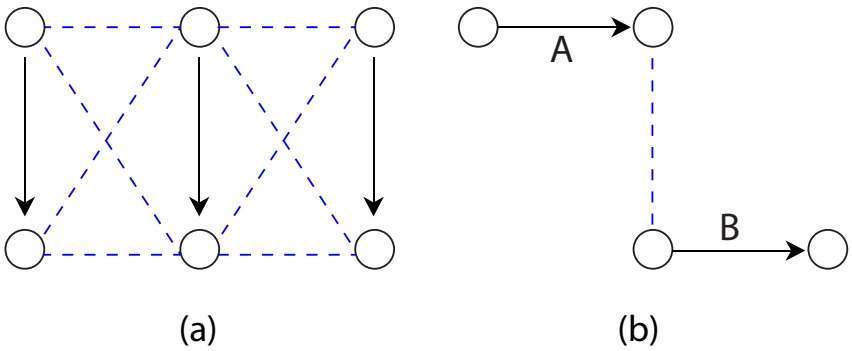


Figure 18.12 Two more atomic topologies: (a) flow in the middle and (b) asymmetric sensing. In (a), the middle session has to wait for both the left and right flows to be silent, and is often starved to near zero throughput in practice. In (b), session A is interfered with session B, but not the other way around. Session A sees much less throughput as a result.

18.4.2 WiFi management

There are many protocol parameters in WiFi. It is not easy to tune these parameters. Based on some noisy reporting of passive measurement data or active probing data, we need to carry out analysis through correlation over different timescales and physical locations, and even across different protocol layers, in order to discover the root cause of the observed inefficiency or unfairness. Then some of these WiFi parameters need to be adjusted, either in real time right on the spot, or on a much slower timescale remotely. Among these parameter adjustments are:

- Dynamic adjustment of rate-reliability tradeoff by using different modulation and codes, thus different transmission speed and decoding error probabilities for a given network condition.
- Dynamic adjustment of channel assignments (*e.g.*, out of the 11 channels available in 802.11b) and transmit power levels to load-balance the stations among neighboring APs.
- Scan frequency determines how frequently a WiFi device can have an opportunity to reassociate itself with a different AP. Too frequent a scanning causes unnecessary interruption of ongoing transmissions. Too infrequent a scanning creates undesirable transmitter-receiver pair configurations.
- There is an upper bound B on the number of retries a frame's transmission can go through. After the retry timer runs out, the frame is declared lost and the recovery is handed over to upper layers. Too early a retry timeout means lost opportunity of recovering the frame, and too late a retry timeout

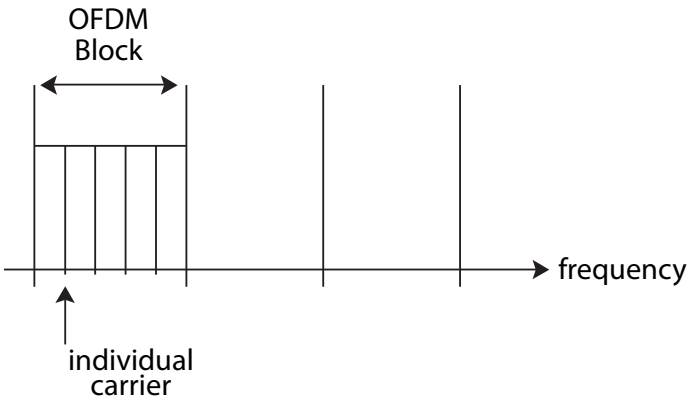


Figure 18.13 A conceptual summary of OFDM, where the frequency band is chopped up into blocks and each block is further divided into a set of carriers. Signals are modulated onto each carrier for transmission, and signal processing across the blocks helps reduce signal distortion.

means upper layers, such as TCP and its congestion control, might invoke their own timeout timers.

- RTS/CTS is a smart idea but it introduces overhead. Only when the payload L is large enough will using RTS/CTS make sense. We will see an example in a homework problem.

18.4.3 OFDM and MIMO

Given the limited basic resource (*i.e.*, the spectrum "crunch") and the interference-limited nature (*i.e.*, negative externality) of wireless communications, people have given a lot of thought into designing both cellular and WiFi network. We want to avoid tragedy of the commons. There are several dimensions along which we can orient our thinking.

Time: *e.g.*, CSMA is a distributed protocol that enables time sharing among WiFi stations and APs without either dedicated resource allocation or centralized control.

Frequency: We have seen dividing frequency duplex: dividing the frequency band into two parts, one for uplink and another for downlink. Another idea, illustrated in Figure 18.13, is to divide each of these parts further into many small frequency chunks, sometimes called carriers. In addition to Paris metro pricing, this is the second time we have seen the use of anti-resource-pooling. This time the justification behind anti-resource-pooling is signal processing rather than revenue maximization. There are efficient ways to process the signals transmitted on narrow frequency bands so that high spectral efficiency, measured in bits per second per Hz, can be maintained. Naturally, those carriers with a high channel

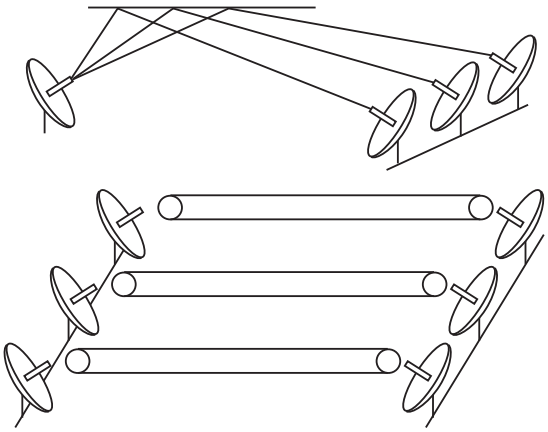


Figure 18.14 Multiple antennas can be used to deliver diversity gain or multiplexing gain in wireless networks. These two types of gains in the wisdom of crowds can also be found in social networks.

quality will be used more, *e.g.*, by spending more of the total power budget there. A common name for this idea is OFDM, and is used in WiFi 802.11a and 802.11g that can have a physical layer speed of up to 54 Mbps. Similar methods are also used in cellular 4G and in DSL technologies.

Space: In wireless networks, as in social networks and P2P networks, the very concept of a "link" is tricky. There are no pipes in the air, just an electromagnetic field with energy propagating in it.

- Since the early days of wireless communications, a first step in utilizing the spatial dimension is to install multiple antennas at the receiver, so that energies bouncing off from different paths can all be collected. These signals are then processed, *e.g.*, by quantifying how similar two signals are using correlation coefficients that we saw in Netflix recommendation in Chapter 4, and by combining them with a weighted sum similar to a round of AdaBoost in Chapter 6. This realizes the diversity gain (Figure 18.14(a)) in the wisdom of crowds, symbolically captured by $1-(1-p)^N$.
- For over two decades, people have also studied how to install and then leverage multiple antennas on the transmitter side, sending different versions of the signal out from each of these antennas. These are called antenna-arrays or smart antennas. Some cellular standards even used smart antennas to create Space Division Multiple Access.
- Since the late 1990s, Multiple Input Multiple Output (**MIMO**) systems have gone from academic research to widespread usage. For example, the new 802.11n WiFi systems are MIMO based, with up to 4 transmit antennas and 4 receive antennas, a 4×4 configuration. MIMO is one of the reasons they can provide a physical layer speed on the order of 100 Mbps. In the-

ory, if there is enough diversity in the channels and sufficient knowledge about the channel conditions at the transmitter or the receiver, we have a factor of N increase in speed for an $N \times N$ MIMO system. This is thanks to the multiplexing gain in the wisdom of crowds, as these signals sent along different channels created by MIMO collectively "add up" to create a high speed pipe. It realizes the multiplexing gain (Figure 18.14(b)). It also reminds us of the factor of $1/N$ reduction in averaging of independent guesses in Chapter 5.

We now take OFDM, MIMO, and physical layer speeds to the next chapter and examine the throughput as observed by the end users.

Further Reading

There are literally thousands of research papers on all aspects of WiFi, from performance to security, from new protocols to hardware experimentation. This is in part due to the availability of simulation and experimentation platforms for unlicensed band transmissions. Almost every research group on wireless communications and networking in the world has published some papers on WiFi.

1. The primary source of the past and ongoing standards can be found on the IEEE website:
[802] IEEE 802.11 Wireless Local Area Networks Working Group
`www.ieee802.org/11`

2. A well written and comprehensive book on WiFi protocol details can be found at:
[Gas05] M. S. Gast, *802.11 Wireless Networks*, 2nd Ed., O'Reilly, 2005.

3. The following textbook on wireless communications has extensive coverage of the physical layer technologies, including OFDM and MIMO, in the latest 802.11 standards:
[Mol11] A. F. Molish, *Wireless Communications*, 2nd Edition, Wiley, 2011.

4. A shorter derivation of a widely cited model on modeling of DCF performance can be found in the following paper, which we also have followed in the example in chis Chapter:
[BT05] G. Bianchi and I. Tinnirello, "Remarks on IEEE 802.11 DCF performance evaluation," *IEEE Communication Letters*, vol. 9, no. 8, pp. 765-767, August 2005.

5. Part of the discussions in Advanced Material were motivated from a recent experimental evaluation of CSMA enhancement:

[Nar11] B. Nardelli, J. Lee, K. Lee, Y. Yi, S. Chong, E. W. Knightly, and M. Chiang, "Experimental evaluation of optimal CSMA," *Proceedings of IEEE INFOCOM*, April 2011.

Problems

18.1 *Hidden nodes* ⋆

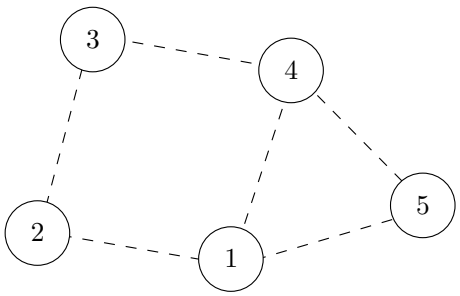


Figure 18.15 A simple network to illustrate hidden nodes. A dashed edge between two stations indicate the stations can transmit to and interfere with each other.

Consider the network in Figure 18.15.

(a) Suppose station 1 is transmitting to station 2. Which station(s) can cause the hidden node problem?

(b) What about station 1 transmitting to station 4?

(c) What about station 1 transmitting to station 5?

18.2 *Flow in the middle* ⋆

Consider the "flow in the middle" topology in Figure 18.12(a), which has three sessions A, B, and C.

(a) Figure 18.16 shows the activity of sessions A and C in time. Draw on the figure the range of time when session B can transmit without colliding with the other sessions.

(b) Figure 18.17 shows the activity of sessions B and C in time. Draw on the figure the range of time when session A can transmit without colliding with the other sessions.

(c) Explain why session B is disadvantaged.

18.3 *Information asymmetry* ⋆

Figure 18.16 Session activity. Grey areas indicate the time when a session is transmitting a frame.

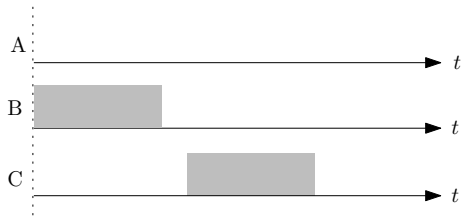


Figure 18.17 Session activity. Grey areas indicate the time when a session is transmitting a frame. Note the transmissions of B and C do not overlap in time.

Consider the "asymmetric sensing" topology in Figure 18.12(b), assuming RTS/CTS is enabled.

(a) Figure 18.18 shows the activity of session B. At time t_1 station 1 senses the channel to be idle and sends an RTS frame to station 2 in an attempt to transmit data. What will happen?

(b) Suppose station 1 sends RTS frames at times t_2, t_3 and t_4. What will happen? Roughly speaking, what is the relationship between the time differences $t_2 - t_1$, $t_3 - t_2$ and $t_4 - t_3$?

(c) Explain why it is difficult for session A to transmit successfully.

(d) Suppose we reverse the positions of sessions A and B, *i.e.*, session A is transmitting and station 3 sends an RTS frame to initiate data transfer in session B. Explain why the problem in part (c) disappears.

Figure 18.18 Session activity. Grey areas indicate the time when a session is transmitting a frame.

18.4 *Aloha* ⋆⋆

There is a simpler random access protocol than CSMA. It is called **Aloha**, as it was invented in Hawaii in the early 1970s, and further lead to the development of packet radio technologies from ARPA. The operation of (the slotted time version of) Aloha is easy to describe. During each time slot, each of a given set of N users chooses to transmit a packet with probability p. We assume that if two or more users transmit at the same time slot, all packets are lost. Each lost packet is retransmitted with probability p too. We assume this process continues until a packet is eventually transmitted successfully.

(a) Assume the channel supports 1 unit of capacity (measured in Mbps), when there is a successful transmission. What is the throughput S as a function of N and p?

(b) What is the optimal p, as a function of N, to maximize the throughput?

(c) As the network becomes large and $N \to \infty$, what is the maximized throughput? You will see it is not a big number, which is intuitive since slotted Aloha described above has neither the listen-and-wait nor exponential backoff features. Aloha takes the least amount of communication and coordination and it does not even use carrier sensing. It also has a low throughput. CSMA leverages implicit message passing through carrier sensing but requires no further explicit coordination. Its throughput can be high for a very small number of users but drops as the crowd gets larger. A centralized scheduler would have taken the most coordination overhead, and in turn provide the best performance. But in many networks, it is infeasible to afford a centralized scheduler.

18.5 *Alternative backoff rules* ⋆⋆⋆

Suppose there are two stations in a CSMA network attempting to transmit a data frame. The two stations start at *stage* 1 with some contention window size w_1, and each station chooses a time slot, within the contention window, uniformly at random. If the chosen time slots collide, then the stations proceed to stage 2 with an updated contention window size w_2, and so on. Transmission completes at some stage i, if during this stage the two stations choose different time slots. We are interested in the expected number of time slots elapsed before transmission completion. This expected number is a measure of how efficient the transmission is (the smaller the better).

To simplify the upcoming analysis, we assume there is no limit to the number of stages, *i.e.*, the contention window size is unbounded from above.

(a) Suppose the two stations are in stage i with contention window size w_i.

What is the probability that the stations choose the same time slot? Conditioning on the two stations having chosen the same time slot, what is the expected value of the time slot chosen, given they are indexed from 1 to w_i?

(b) What is the probability that the transmission completes at stage i?

(c) Given the transmission completes at stage i, what is the expected number of time slots elapsed?

(Hint: it is the sum of the expected values of time slots chosen in previous stages (see part (a)), plus the expected value of the maximum of the two (different) time slots chosen at stage i, which is $2(w_i + 1)/3$.)

(d) What is the expected number of time slots elapsed before transmission completion?

(Hint: apply the law of total expectation.)

(e) Now we plug in different contention window update rules. Consider the following three rules:

(i) Binary exponential backoff: $w_i = 2^i$;

(ii) Additive backoff: $w_i = i$;

(iii) Super-binary exponential backoff: $w_i = 2^{2i}$.

Compute the expected number of time slots in part (d) for the three cases. What do you observe? How does that match the intuition that the best backoff policy should be neither too slow nor too aggressive?

19 Why am I only getting a few % of advertised 4G speed?

By the end of this chapter, you will count yourself lucky to get as much as a few % of the advertised speed. Where did the rest go?

19.1 A Short Answer

First of all, the term 3G is confusing: there is one track following the standardization body 3GPP called **UMTS** or **WCDMA**, and another track in 3GPP2 called **cdma2000**. Each also has several versions in-between 2G and 3G, often called 2.5G, such as EDGE, EVDO, etc. Then 4G has an even more confusing terminology: the main track is called LTE, with variants such as LTE light and LTE advanced. Another competing track is called WiMAX. Some refer to evolved versions of 3G, such as HSPA+ as 4G too. All these have created quite a bit of confusion on a consumer's mind as to what really is a 3G technology and what really is a 4G technology.

You might have read that 3G downlink speed for stationary users should be 7.2 Mbps. But when you try to download an email attachment of 3 MB, it often takes as long as one and half minutes. You get around 267 kbps, 3.7% of what you might expect.

Many countries are moving towards 4G wireless called Long Term Evolution (**LTE**) networks now. They use a range of techniques to increase the **spectral efficiency**, defined as the number of bits per second that each Hz of bandwidth can support. It uses methods like OFDM mentioned at the end of Chapter 18, multiple antennas capturing different signals from different paths in the air, and splitting a large cell into smaller ones.

But the user observed throughput, while much higher than 3G, still falls short of the advertised numbers we often hear in the neighborhood of 300 Mbps. Why is that? There are two main reasons: non-ideal network conditions and overheads.

Many parts of the wireless network exhibit non-ideal conditions, including both the air interface and the backhaul network. Furthermore, networks, like our lives, are dominated by overheads, such as the overhead of network management in the form of control bits in packets or control sequences in protocols.

This chapter is in some sense the "overhead" of this book: there are no further "deep" messages other than the importance of overhead: networking is not just

about maximizing performance metrics like throughput, but also the necessary "evils" of managing the network.

Let us go into a little bit of detail on three major sources of "speed reduction," or more accurately, reduction in **useful throughput** in a session from a sender to the receiver, defined as the number of bits of actual application data received, divided by the time it takes to get the data through. This is what you "feel" you are getting in your service, but may not be what advertisements talk about or what speed tests measure.

19.1.1 Air-interface

1. *Propagation channel*: The wireless channels suffer from various types of degradation, including **path loss** (signal strength drops as the distance of propagation increases), **shadowing** (obstruction by objects), and multipath **fading** (each signal bounces off of many objects and is collected at the receiver from multiple paths). A user standing at the cell edge, far away from the base station and blocked by many buildings, will receive a lower rate than another user standing right under a base station. These factors come into play even if there is only one user in the whole world.
2. *Interference*: There are also many users, and they interfere with each other. As mentioned in Chapter 1, if there are few strong interferers, or if the interferers are weak but there are many of them, the received Signal-Interference-Ratio (SIR) will be low. At some point, it will be so low that the order of modulation needs to be toned down and transmission rate reduced so that the receiver can accurately decode. A typical instance of the problem is the *near far problem*. Even power control in Chapter 1 cannot completely resolve this problem.

19.1.2 Backhaul

There can be more than 10 links traversed from the base station to the actual destination on the other side of a wireless session of, say, YouTube. The session first goes through the radio access network, then the cellular core network also owned by the cellular provider, then possibly some long distance providers' links, then possibly multiple other ISPs composing the rest of the Internet, and finally to Google's data center network.

1. *Links*: Users' traffic competes with each other on the links behind the air interface in the cellular network. As explained in more detail in the next section, many wireless networks actually have most of their links in wireline networks. Congestion happens on these links and the resulting queuing delay reduces throughput. Plus there is also propagation delay simply due to the distance traversed. An increase in delay reduces throughput, since it is defined as the number of bits that can be communicated from the source to the destination per second.

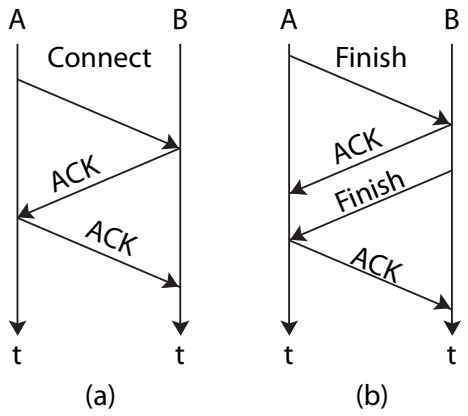


Figure 19.1 (a) Three way session establishment and (b) four way session tear down in TCP. (a) After A initiates a connection with B, B sends an acknowledgement, and A acknowledges the acknowledgement, so that B knows that A knows there is now a connection established. (b) After A initiates a session tear-down, B first acknowledges that. Then B sends a tear-down message right afterwards, since TCP connections are bidirectional: A having no more messages for B does not mean B has no more messages for A.

2. *Nodes*: These links are connected through nodes: gateways, switches, routers, servers, etc. Some of these, like routers, store packets while waiting for the egress links to be ready, increasing packet delay. Others, like servers, have processing power limitations, and can become heavily congested when they are in popular demand. For example, a popular web server or video server may become so congested that it cannot process all the requests. This has nothing to do with the rest of the network, just a server that cannot handle the demand. Yet it does reduce the throughput for the session.

19.1.3 Protocols

1. *Protocol semantics*: Many functionalities require sequences of message passing. For example, in TCP, each session needs to be set up and torn down, through a 3-way handshake and a 4-way tear down, respectively. This process is illustrated in Figure 19.1. Why does the network protocol designer bother to create such a complicated procedure just for setting up and terminating a session? Well, because in this way, for session establishment, both the sender and receiver know that there is a session and that the other knows it too. And for session tear down, 4-way handshake ensures there is no dangling state of connection in either direction of a full-duplex connection (*i.e.*, a bidirectional path where both ways can be carried out at the same time). Obviously, for

shorter sessions, these overheads occupy a larger fraction of the capacity used, leaving less for the application data.

2. *Packet headers*: As explained in Chapter 13, each layer adds a header to carry control information, such as address, protocol version number, quality of service, error check, etc. These headers also leave some space for flexible future use too. These headers add up, especially if the packet payload is small and the fraction of header becomes larger. Some protocols also specify a packet fragmentation threshold, so bigger packets are divided into smaller ones, adding to the fraction of header overhead.
3. *Control plane signaling*: Think about an air transportation network. The actual traffic of people and cargo is carried out by airplanes between airports following some routes. But the routing decision and many other control signals traverse entirely different networks, possibly the Internet or the telephone network. The data plane is separated from the control plane. On the Internet, the actual data traffic flows on **data channels** (a logical concept, rather than physical channels), while control signals travel on **control channels**. These signaling channels take portions of the available data rate and reserve them for control purposes. While most people realize the importance of data channels, control channel is just as critical. In 3G and 4G standards, a great deal of effort is put into designing control channels. Sometimes they are not sized right, causing extra delay and reducing throughput further.

In general, there are five main functionalities of network management:

- *Performance*: monitor, collect, and analyze performance metrics.
- *Configuration*: update configuration of the control knobs in different protocols.
- *Charging*: maintain the data needed to identify how to charge each user, e.g., when a user uses the network in time dependent pricing.
- *Fault-management*: monitor to see if any link or node is down, and then contain, repair and root-cause diagnose the fault.
- *Security*: run authentication, maintain integrity, and check confidentiality.

The messages of these functionalities sometimes run on channels shared with the actual data (in-band control), and sometimes run on dedicated control channels (out-of-band control). Collectively, they form the control plane. Protocols running network management include examples like Simple Network Management Protocol (**SNMP**) for the Internet.

19.2 A Long Answer

The speed of your wireless (or wireline) Internet connection is not one number, but *many* numbers, depending on the answers to the following four questions.

First, speed as measured in *which layer*? This is often a primary reason for the confusion on speed test results. For example, wireless standardization bodies

often quote physical layer speeds, but users only experience application layer speed directly. Depending on which layer we are talking about, it also changes which parts of backhaul and protocols are involved. The more links traversed in the backhaul network, the more chances to run into congestion. The more protocols involved, the more overheads.

In Chapter 17, we saw a few key protocols often used in each of the layers. As another example, in LTE, just the link layer (layer 2) consists of 3 sublayers, each with its own overhead:

- *MAC layer*: medium access control. It controls the multiplexing of data in different logical channels, and decides the scheduling of the packets.
- *RLC layer*: radio link control. It controls the segmentation and reassembly of packets to fit the size that can be readily transmitted on the radio link. It also locally handles the retransmission of the lost packets.
- *PDCP layer*: packet data convergence protocol. It processes header compression, security, and handover.

Second, speed as measured *where*? Depending on the locations of the two end points of the speed measurement, different parts of the backhaul networks and different protocols are involved. It is the weakest link that matters when it comes to a speed test. For example, speed is often measured between a mobile device and the base station in the cellular industry. But as shown in Figure 19.2, the base station is followed by a mobile switching center and a couple of gateways before reaching the IP Internet backbone and eventually reaching the other end, *e.g.*, another mobile device or a server. The speed measured will be different as we put the destination at each of these possible points along the path. If the speed test is between the smart phone and the base station, only the air-interface and physical and link layers' protocols are involved. If the speed test is between the smart phone and the content server, then the cellular backhaul, the backbone IP network, and the content server's congestion conditions will be relevant. So will network layer, transport layer, and application layer protocols.

Third, speed as measured *when*? At different times of the day, we see different amounts of congestion in the air interface and in the backhaul. Traffic intensity during different hours of the day often exhibits a repetitive pattern, especially at higher aggregation levels. More user activity translates into more bits to be transmitted, causing

- *interference* (a multiplicative behavior) as we saw in Chapter 1: $x/y \geq s$ where s is some target SIR,
- *congestion* (an additive behavior) in Chapter 14: $x + y \leq c$ where c is some capacity limit,
- and *collision* (a binary behavior) in Chapter 18: $x, y \in \{0, 1\}$ if sessions x and y cannot transmit at the same time without colliding.

Fourth, speed as measured for *what application*? This matters for two reasons. Different traffic runs different sets of protocols, some with more overhead than

others. For example, texting may only take a little capacity on the data channel traffic but it does require a lot of control channel traffic. Email download and web traffic can be less overhead-heavy compared to voice over IP or video conferencing. User utility and expectation also differ a lot in different applications. Interactive gaming has very stringent demands on delay and jitter, while file download does not. Some applications can tolerate a longer initialization latency if the throughput is consistently high once it starts. Other applications are more sensitive to time shifts but can accommodate throughput fluctuations gracefully. Any objective measure of speed eventually must translate into subjective user experience.

19.3 Examples

We will now walk through some numerical examples of non-ideal network condition, before turning to a further discussion of protocol overhead in Advanced Material.

19.3.1 An air interface example

Here is an example for 4G LTE. The throughput, at the physical layer and over the air-interface only, can be estimated as follows.

For each subframe of data (one unit of time lasting 1 ms), we count how many bits are sent. LTE physical layer uses OFDM, dividing the spectrum into blocks and running signal processing on each of them. The number of bits transmitted equals (a) the number of symbols per frequency block multiplies (b) the number of frequency blocks, multiplies (c) bits per symbol per frequency block, multiplies (d) coding and multi-antenna gain.

For part (a) above, the number of symbols is (a1) the number of symbols per frequency carrier, times (a2) the number of carriers per frequency block, with (a3) control overhead deducted.

Therefore, we have the following formula to count bits sent in a subframe:

[(symbols/carrier - control overhead) × number of carriers per frequency block - channel estimation overhead] × bits/symbol × number of frequency blocks × coding and multi-antenna gain.

So, what are these factors's numerical values?

- Symbols/carrier: this is typically 12-14 as used in LTE.
- Control overhead/carrier: at least 1 but sometimes 2 or 3 symbols per carrier in a subframe is for control signaling.
- Carriers/frequency block: usually 12 carriers per frequency block of 180 kHz.
- Channel estimation overhead per frequency block: the overhead spent on sending pilot symbols to estimate the channel is usually 20 symbols for 4 by 4 multi-antenna systems.

- Bits/symbol: this depends on the modulation, the received SIR, and the decoder capability. Ideally it can be 6 for LTE, using $2^6 = 64$ QAM modulation. But often it is 4, using the lower order of $2^4 = 16$ QAM modulation, when the channel condition is not good enough (*e.g.*, when there is too much shadowing, fading, or the distance to the base station is long). If the channel is even worse, it can become 2.
- Number of frequency blocks: Suppose we can use the entire 20 MHz frequency band for an LTE channel, with 1 MHz guard band on each side. Since each frequency block in OFDM is 180 kHz, we have 100 blocks. But in reality the two way communication is carried out by either frequency division duplex (FDD), and we only get 10 MHz, *i.e.*, 50 blocks, or time division duplex (TDD), and we get 40% of the time using the frequency for uplink and 60% of the time for downlink.
- Coding rate: Coding rate is the overhead due to channel coding, which adds redundancy to protect the transmitted bits against channel distortion. The higher the coding rate, the less redundancy is added. Ideally we want it to be close to 1, but it can be lower for certain codes. The higher protection required and the less efficient a code, the lower this factor becomes.
- Multi-antenna gain: ideally for 4 by 4 multi-antenna systems, it should be a factor of 4. But due to the limitation of devices (sometimes only 2 antennas can be installed) and the channel correlation in space (when antennas are too close to each other, they do not experience very different channels), it will be more like a factor of 2.

Therefore, the number of bits transmitted over a 1 ms subframe timeslot, in the *best case*, will be

$$[(14-1)\times 12-10]\times 6\times 100\times 0.9\times 4 = 315,360 \text{ bits},$$

which translates into $315,360/0.001 = 315$ Mbps.

But in reality it is more likely going to be

$$[(12-2)\times 12-20]\times 4\times 50\times 0.7\times 2 = 28,000 \text{ bits},$$

which translates into $28000/0.001 = 28$ Mbps, *i.e.*, 8.8% of the original expectation.

If the MAC layer retransmission and overhead is counted: PDCP has 3.5 bytes of header plus sequence number, RLC has 5.5 bytes of header plus sequence number, MAC has 1 byte and CRC 3 bytes, and there is at least another factor of 0.9 loss, dropping the rate to about 25 Mbps.

This is only the PHY layer channel degradation and the MAC layer overhead. It is already less than 8% of the ideal number. If we count interference among users, and the upper layer protocols, and the backhaul network congestion, the throughput number can easily drop by another factor of 2-5.

Now, 5-10 Mbps on a mobile device is still quite impressive and can enable a lot of applications, especially when most people's home WiFi experience is about

5-20 Mbps (as constrained by either the 802.11b speed or the residential gateway backhaul speed). But you should not be surprised if you do not see 300 Mbps on your LTE smart phone.

19.3.2 Backhaul examples

The cellular backhaul consists of some links (*e.g.*, microwave links, free space optical links, satellite links) that connect the air-interface with the rest of the end-to-end path in Figure 19.2 , and then the cellular core network, and finally the public IP network.

Here is another example for transport protocol, *e.g.*, TCP, on wireline networks. The TCP throughput can be estimated as follows:

$$\text{TCP throughput} = \text{TCP-Window-Size/RTT},$$

where RTT denotes the Round Trip Time. The maximum window size is $2^{16}-1 = 65,535$ bytes, due to the receiver window's 16-bit field in the TCP header. The purpose of the receiver window is to prevent a fast sender from overwhelming a slow receiver.

Suppose you are connected to a 1 Gbps Ethernet link and transmit a file to a destination with a round trip latency of 100ms. In this case, the maximum TCP throughput that can be achieved is

$$65535 * 8/0.1 \approx 5.24 \text{ Mbps}.$$

Therefore, even if you are connected to a 1Gbps Ethernet link, you should not expect any more than 5.24Mbps when transferring the file, given the TCP window size and the round trip time.

In practice, TCP may not even attain the maximum window size because of the congestion control mechanism which prevents the sender from overloading the network. The TCP window is min{congestion window, receiver window}.

The congestion window is reduced when congestion is detected inside the network, *e.g.*, a packet loss. The TCP throughput for long distance WAN links can be estimated below, as shown in a homework problem in Chapter 14:

$$\text{TCP throughput} \leq \text{MSS}/(\text{RTT} * \sqrt{p}),$$

where MSS is the Maximum Segment Size (fixed for TCP/IP protocol, typically 1460 bytes), and p is the packet loss rate. Consider a packet loss rate of 0.1%, which gives the following throughput upper bound:

$$1460 * 8/(0.1 * \sqrt{0.001}) \approx 3.63 \text{ Mbps}.$$

Finally, consider the problem of *flash crowds*, *i.e.*, a large number of visits to a popular web server, all within a short amount of time. The server's application may not generate the data fast enough because of server bottlenecks (on CPU, memory, or network bandwidth). For example, the sender may write a small amount of data, triggering **Nagle's algorithm** that delays sending the data: it

combines small data together into large packets for better network utilization, but the delay goes up. This is a typical tradeoff in managing overhead by aggregating frames: you might be able to reduce the amount of overhead, but at the expense of an increase in latency. A similar tradeoff shows up in aggregating acknowledgements: if the aggregated acknowledgement is lost, the sender would have to retransmit all the packets not acknowledged, an effect that is similar to losing an I frame in a GOP in Chapter 17.

Suppose that, due to the reasons above, it takes 4 RTTs for the server to send out 1 MSS of data. The effective throughput becomes extremely small:

$$1460 * 8/(0.1 * 4) \approx 29.2 \text{ kbps}.$$

This may explain why you often have to wait a long time for a simple webpage to show up on your browser. If the bottleneck is at the server as stated above, it does not matter if your smart phone is on 4G LTE or the old 2G GSM.

19.4 Advanced Material

We delve deeper into control protocol overhead in this section, with three cases on cellular core network management, mobility support, and local switching. In a homework problem, we will also explore the overhead associated with network security.

19.4.1 Cellular core network

A wireless network, cellular or WiFi, actually consists mostly of *wired* links. Usually only the link between the end user device and the base station (or access point in WiFi's case) is wireless. So how does traffic go from the base station through the rest of the Internet and reach the destination? If the destination is another wireless device, there will be another wireless leg there, but otherwise it is all wired links now.

A packet traverses through two segments of wireline networks. One is the cellular core network, backhauling the traffic from the base station through a set of servers and links carefully designed and managed by the cellular service provider. Then there is the public IP network, run by either one or a federation of ISPs. We have seen some features of the public IP network already, and now we sketch the cellular core network. We only discuss a tiny fraction of the cellular core that allows us to understand where the overheads might arise, and to lead into the discussion of mobility management and network management.

Why do we even need a core network to support the radio air-interface of a wireless network? That is because there are many tasks beyond sending and receiving signals over the air. For example, some systems are needed to take care of billing, to support mobility of users, to monitor traffic, and to provision

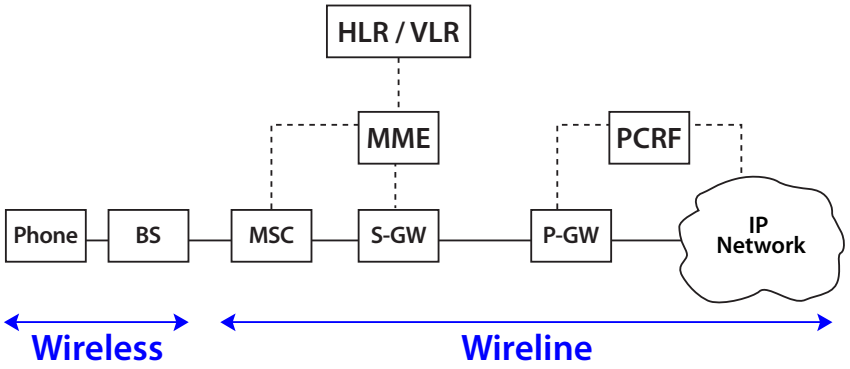


Figure 19.2 Main components of the Evolvable Packet Core in LTE: the air interface (between a phone and the BS), the public IP network, and the rest of the wireline network in between called the cellular core network. The solid lines represent physical connections, with the one between the phone and the BS the only wireless link. The dotted lines represent control plane channels. The BS passes through stages of gateways in the cellular core network before reaching the public IP network.

quality of service, and to ensure inter-operability between different types and generations of networks.

Each generation of cellular standards has a different design of the core network. For 4G LTE, it is called the Evolvable Packet Core (**EPC**) network. The starting point of the EPC is the base station, also called the **eNB**, the evolvable Node B, and the associated Mobile Switching Center (**MSC**), each controlling a set of base stations. The end point is an IP router. In-between there are a few things going on:

- *Hardware*: There are multiple gateways filled with servers and switches. The two main ones are the Serving Gateway and the PDN Gateway.
- *Software*: There is a suite of software in charge of accounting and billing, monitoring and configuration, multimedia signal processing, mobility support, security and privacy, quality of service control, IP address allocation, *etc.*
- *Control channels*: The control signaling is sometimes carried out in-band: together with the actual data, and sometimes out-of-band: over special channels dedicated to control signals.

As shown in Figure 19.2, the main logical components of the EPC include the following:

- *PCRF*: Policy Control and charging Rules Function. It is responsible for policy control decision making, *e.g.*, authorization to differentiate the quality of service.

- *MME*: Mobility Management Entity. It controls the signaling between user devices and the EPC.
- *HLR*: Home Location Register. This is a database to support mobility, to be explained next.
- *S-GW*: Server GateWay. It counts the number of bytes for charging purposes. It serves as the local mobility anchor, buffering data while the MME initiates a handoff. It also interfaces with some 3G cellular standards like UMTS and GPRS for a smooth integration. It processes at the link layer.
- *P-GW*: PDN GateWay. It allocates IP addresses, controls quality of service, and enforces other rules from the PCRF. It also interfaces with other 3G or 4G standards like cdma2000 and WiMax. It processes at the network layer running the IP.

19.4.2 Mobility management: mobile IP and cellular handoff

Your device's IP address, or your phone number, is a type of unique ID. But for mobile devices, the problem is that the ID must be *decoupled* from a fixed location. When your laptop moves in a building, you may be communicating with a different AP. When you drive around, the base station serving you changes whenever the channel between a new base station and you is better than that between the current one and you. When your plane lands at the airport and you turn your iPhone back on, all your emails and voicemails must locate where you are now.

The above cases have different velocities of mobility and different spatial spans of change in location, but they all need to take care of mobility management. The key point in all solutions is to have an *anchor*, a point where others can reach you no matter where you are, and that point keeps track of you, or who can find you. This is similar to the scenario when an elementary schoolmate wants to find you after you go to a different city for college. She may contact your parents' home, the "home agent," trusting that your parents will know how to contact your college dorm, the "foreign agent," and in turn reach you.

Consider a device A with a fixed IP address and a home network. There is a **home agent** in the home network. As the device moves into a foreign network as a visitor, it contacts an agent there in charge of taking care of these visitors. It tells that foreign agent its fixed IP address and home agent ID. Then the **foreign agent** will inform the home agent that the device A is now residing in its network.

Now when someone, say, Bob, wants to send a message to device A, the message goes to the home agent. Bob does not know where device A might actually be, but he knows that the home agent would know. The home agent looks up a table that matches this device's IP address with the current foreign agent's ID, contacts that agent, and forwards the message to it. The foreign agent then forwards the message to the device. This process of **indirect forwarding** is

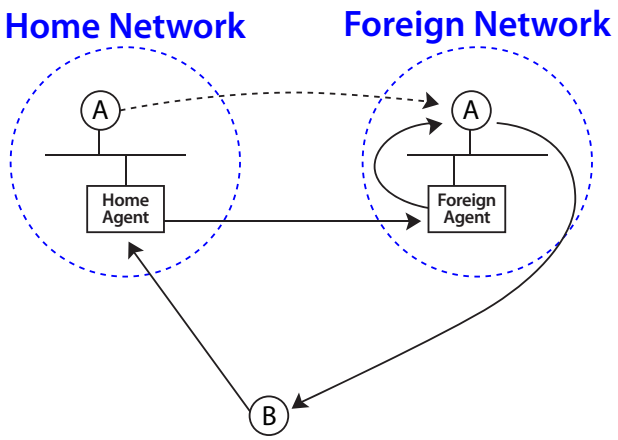


Figure 19.3 Indirect forwarding for mobility support. Device A moved from the location on the left in the homework network to a new location on the right in a foreign network. The foreign agent informs the home agent. So when another device B calls A, it first contacts the home agent who can forward the call to the foreign agent, who in turn forwards the call to A. It is simpler for A to communicate back to B.

shown in Figure 19.3. If the device keeps moving, the first foreign agent becomes the **anchor foreign agent** and keeps forwarding the packets onward.

Adding the foreign agent address to the message makes it clear how the message has been handled. If you want to keep it completely transparent to device A, the home agent can *encapsulate* the actual message and *tunnel* it through to the foreign agent, who can then strip the encapsulation.

The principle of having a home agent as the permanent address anchor and having a continuously updated foreign agent is the key idea behind both mobile IP protocols and cellular handoff procedures.

Figure 19.4 shows the handoff procedure in mobile networks. Each cell phone has an entry in the permanent address database, called **Home Location Register** (HLR). As it moves to foreign networks, a care-of-address is entered into the dynamic **Visitor Location Register** (VLR). The calling procedure then follows the above indirect forwarding method in Figure 19.3.

As to the actual resource allocation for handoff, it depends on whether the phone is moving across boundaries of an MSC or not. If moving across to a different BS but still with the same MSC, that MSC can manage the handoff. If it is also crossing to a new MSC, as shown in Figure 19.5, then the moment the current base station detects that the SIR is low and a handoff is needed, it asks its MSC to notify a nearby MSC, which in turn asks the right BS to prepare for a handoff and to allocate the necessary radio resources. Then the "ready to go" signal is sent all the way back to the current base station, which then tells the phone to shift to the new BS.

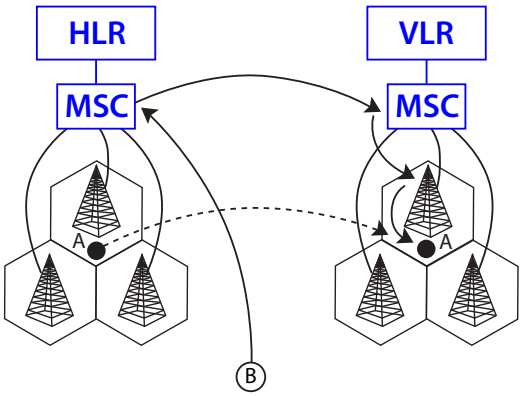


Figure 19.4 Handoff in wireless cellular networks with indirect forwarding. The solid undirected links indicate the connections between the MSC and the BSs. The dotted line indicates the movement by phone A from one cell to another, controlled by a different MSC. The solid arrowed lines represent the communication path from a caller B, first to the MSC in A's home network, then to the MSC in A's visiting network, and finally to A.

19.4.3 Protocol overhead in switching and routing

In Chapter 13, we discussed routing based on IP addresses. But each device's network adaptor actually recognizes only MAC addresses. Unlike the 32 bit, sometimes dynamically assigned IPv4 address (*e.g.*, 64.125.8.15), each MAC address is a 48 bit, hard coded number, often expressed in the format of 6 segments of 8 bits each (*e.g.*, 00-09-8D-32-B2-21).

When we discussed the DHCP's dynamic assignment of IP addresses in Chapter 13, we skipped the detail of how that is done, when a device has neither the source IP address (that is precisely what needs to be done via DHCP) nor the destination address (it does not know where the local DHCP server is).

Now we briefly go through these to highlight that distributed protocols carry a price of overhead.

First, the DHCP takes in a MAC address and returns an IP address, together with other local network information such as the DNS server and gateway router. The dynamic IP address is leased for a period of time before it must be renewed. This is an example of **soft state**.

In soft state, a configuration needs to be periodically refreshed. If not, it is erased, *i.e.*, the state disappears unless instructed otherwise. In contrast, in **hard state**, a configuration stays there forever until an explicit "tear down" command is issued, *i.e.*, the state remains the same unless instructed otherwise. Many Internet protocols use soft state because of the risk of leaving dangling states when control packets get lost in the network. This is similar to mobile devices

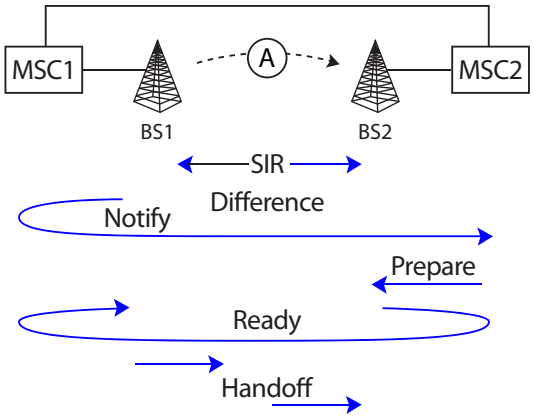


Figure 19.5 Handoff in wireless cellular networks with resource reservation. As phone A moves to some point between BS1 and BS2, and detects that the received SIR from BS2 is starting to be better than that from BS1, it initiates a handoff. MSC1 notifies MSC2, which asks a right BS to prepare the necessary radio resources, then notifies back to MSC1 that BS2 is ready to receive A. Finally, MSC1 tells BS1 to let A be handed off to BS2.

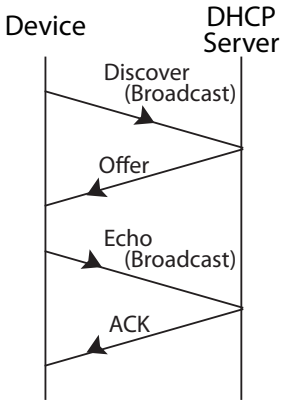


Figure 19.6 The basic protocol overhead associated with a device getting connected with a DHCP server so that a dynamic IP address can be obtained from the server. It illustrates the principle of "when in doubt, shout," as the device cold-starts with a broadcast message to discover the DHCP servers it can reach.

using soft state for keeping the screen on: unless you use the phone before the timer is up, it will go to some type of sleep mode to conserve energy.

Back to the DHCP. But how does a new device on a local network know how to contact the DHCP server? It does not. So it sends a **broadcast** message

of "DHCP server discovery" to solve this bootstrapping problem. Some communication media, like wireless, are also intrinsically broadcast in nature. Then the DHCP servers hearing this message each send a "DCHP offer" (which includes an IP address, a DNS server ID, a gateway router ID, and the lease time). The device then broadcasts an echo of the parameters from the DHCP server it chooses. Finally, the chosen DHCP server realizes it has been chosen and sends an acknowledgement. This four way handshake is illustrated in Figure 19.6.

Our second challenge is to determine how to deliver a message to the network adaptor of the destination device. This requires the help of another translator: Address Resolution Protocol (**ARP**). It is the opposite of DHCP: given an IP address, the ARP provides the MAC address.

Suppose A (an iPhone) wants to communicate with B (*e.g.*, a `www.bbc.com` server). To simplify the flow, we assume B is connected to A via just the gateway router.

First, A gets B's IP address via the DNS as explained in Chapter 13. Now, A needs to translate the destination IP address to the corresponding MAC address, since that is what device drivers understand. To accomplish this, first A gets the gateway router's MAC address from the ARP. (It already knows that router's IP address from the DHCP server, as shown in Figure 19.6.) This establishes a link layer communication path between A and the router. A encapsulates the IP packet into a MAC frame. Upon receiving it, the router extracts the IP packet from the MAC frame, and reads the destination IP address. Now the router gets B's MAC address from this IP address by another ARP, and can send it to B. This process is illustrated in Figure 19.7.

Across Figures 19.6 and 19.7, 1 DHCP server, 1 DNS server, and 2 ARP servers are used, and many rounds of control messages are signalled. All these overheads take up capacity and introduce delay.

Both protocols above may sound excessively complicated. But if you want to have distributed actions in a layered architecture, some message passing among the network elements is a price you have to pay.

Further Reading

Non-ideal network conditions' impact on speed and the overhead of network protocols are relatively under-explored subjects.

1. A famous debate in network management is on the end to end principle, started with the following classic paper:

[SRC81] J. H. Saltzer, D. P. Reed, and D. D. Clark, "End to end arguments in system design," *Proceedings of IEEE International Conference on Distributed Computing Systems*, 1981.

2. On the wireless side, there are very few well written texts on the cellular

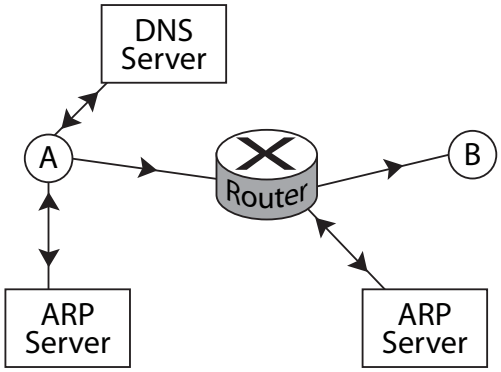


Figure 19.7 The source device with a network adaptor A wants to reach the destination device with a network adaptor B. It first needs to contact the DNS server to get the IP address of B. Then it needs to translate the IP address to the corresponding MAC address that the device driver can understand. Then, the gateway router reads B's IP address from A's packet, gets B's MAC address from another ARP server, and finally finds a way to communicate with B directly.

core networks, relative to air-interface. The following recent book is one of the few:

[Ols+09] M. Olsson, S. Sultanan, S. Rommer, L. Frid, and C. Mulligan, *SAE and the Evolved Packet Core*, Academic Press, 2009.

3. A classic graduate textbook on data networks from twenty years ago still contain some key message relevant in today's study of networks:

[BG92] D. P. Bertsekas and R. Gallager, *Data Networks*, 2nd Ed., Prentice Hall, 1992.

4. Here is an interesting book discussing the underlying reasons and historical accidents behind why the Internet protocols operate the way they do today:

[Day08] J. Day, *Patterns in network architecture: A return to fundamentals*, Prentice Hall, 2008.

5. Here is a unique, recent book in print on the actual practice of designing and managing a global network:

[Cam12] G. K. Cambron, *Engineering and Operation in Global Network*, 2012.

Problems

19.1 *RTS/CTS overhead* ⋆

In the Examples section, we estimated T_s of 802.11g for the case in which RTS/CTS is disabled. Here we estimate T_s for the other case. Given RTS/CTS is enabled, a successful transmission consists of the sequence: [RTS frame] + SIFS + [CTS frame] + SIFS + [data frame] + SIFS + [ACK frame] + DIFS. The only addition is the RTS/CTS handshake that occurs before data transmission.

You are also given: (1) the time to transmit an RTS frame is 23.25μs, and (2) the time to transmit a CTS frame is 22.37μs.

(a) If $L = 8192$ bits, calculate T_s for both the cases of RTS/CTS being enabled and disabled. Calculate the effective throughput as L/T_s.

(b) If $L = 320$ bits, calculate T_s and L/T_s for both cases.

(c) In most home networks, RTS/CTS is disabled. Can you see why?

19.2 *Header overhead* ⋆

A typical IEEE 802.3 Ethernet packet structure is illustrated in Figure 19.8, with the terminologies described below:

- Preamble: Use 64 bits to synchronize with the signal's frequency before transmitting the real data.
- MAC Dest/Src: Record the destination and source MAC addresses of the packet, each with 6 bytes.
- Length: Specify the length of the IP packet with 2 bytes.
- Frame check sequence(CRC): Contains 32-bit cyclic redundancy check which enables detection of corrupted data within the packet.
- Interframe gap: After a packet has been sent, transmitters are required to transmit a total of 96 bits of idle line state before transmitting the next packet.
- IPv6 header: The header of IPv6 packet. 40 bytes in total.
- TCP header: The header of TCP packet. 20 bytes in total.

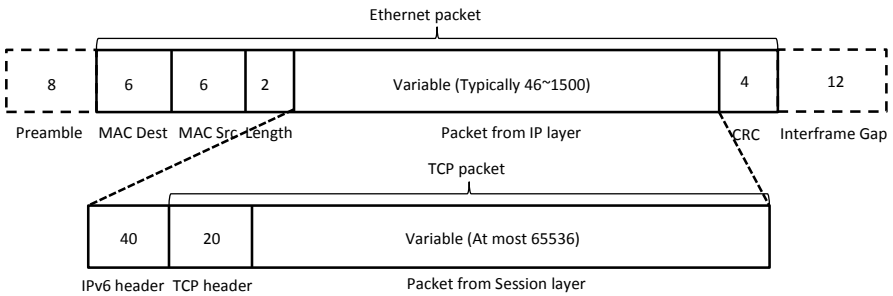


Figure 19.8 IEEE 802.3 packet structure.

What is the percentage of payload data rate if we are to send a 250-byte packet from session layer?

19.3 *Slow start phase's throughput* ⋆⋆

As mentioned in Chapter 14, TCP starts with a small congestion window, which is initially set to 1 MSS (Maximum Segment Size, typically 1460 Bytes), and go through the slow start phase. The congestion window increases multiplicatively, *e.g.*, 2 MSS, 4 MSS, 8 MSS, ..., for every round trip time, until the slow start threshold is reached and the congestion avoidance phase is entered. Suppose you open a web browser and try to download a webpage of 70 kB.

(a) Assume there is no packet lost, how many RTTs are required to download the webpage? Remember to add up 1 RTT of handshake to set up the TCP connection.

(b) If RTT = 100 ms, what is the average throughput? Can you see the impact of slow start on a short-duration session's throughput?

19.4 *Alternatives to indirect forwarding* ⋆⋆

Section 19.4.2 mentioned the process of indirect forwarding in mobility management. Can you think of an alternative forwarding process?

19.5 *Overhead associated with security* ⋆⋆⋆

We have not got a chance to talk about **network security** so far, a significant subject with many books written about it. There are several meanings to the word "security", and many famous methods to ensure or to break security in a network. We will simply walk through the main steps involved in ensuring confidentiality in Secure Shell (**SSH**), an application-layer security protocol heavily used in remote login to ensure that the server is the right one, the client is who it claims to be, and the communication between this client and the server is confidential. Along the way, we will see the amount of overhead involved in providing SSH's secure service.

There are numerous books and papers written about encrypting texts. We will only need the following notion in this homework problem: **public key cryptography**. This is based on mathematical operations that are easy to run one way but very difficult the other way around, *e.g.*, multiplying two large prime numbers is easy, but factoring a large number into two large prime numbers is very difficult. This enables the creation of a pair of keys: a public encryption key (known to anyone who wants to send a message to, say, Alice), and a private decryption key (known only to those who are allowed to decrypt and read the original message).

Now consider a client trying to remotely login a server. If you are the inventor of a secure remote login protocol using public key cryptography, what are the steps you would design? What would the biggest vulnerability of your design?

20 Is it fair that my neighbors iPad downloads faster?

We have come to the last chapter, on a touchy subject that we touched upon many times in the previous chapters: quantifying fairness of resource allocation.

20.1 A Short Answer

20.1.1 Thinking the problem aloud

A naive (and problematic) view is "equality is fairness." If you have to choose from an allocation of $[1,\ 1]$ (of some resource) between two users, and an allocation of $[100,\ 101]$, many people would choose $[100, 101]$ even though it deviates from an equal allocation. Magnitude matters. Part of Rawls' theory of justice is the Difference Principle that we will discuss in Advanced Material, which prefers a less equal allocation where everyone's allocation has a higher absolute value. Of course, a more challenging choice would have been between $[1,\ 1]$ and $[1,\ 2]$.

Another objection to marking equal allocations as the most fair stems from the differences in the contributions by and the needs of different users. If a user in a social network glues the entire network together, her contribution is higher than that of a "leaf node" user. If one works twice as hard or twice as efficiently as another, these two people should not be treated as if they were the same. If a professor assigns A grade to all students no matter their performance, that will be neither providing the right incentive for learning nor deemed fair by an objective person.

And yet most people would agree that a slower worker does not deserve to starve to death simply because she works slower. There are some basic allocations that should be provided to everyone. The debate surrounds the definition of "basic." Different notions of fairness define what is "basic" differently.

Throughout this chapter, we will examine some approaches to debating the above points in unambiguous ways.

20.1.2 Fairness measures from axioms

Given a vector $\mathbf{x} \in \mathbb{R}_+^n$, where x_i is the resource allocated to user i, *how fair* is it? This question is a special case of the general question on fairness.

Consider two feasible allocations, $\mathbf{x}$ and $\mathbf{y}$, of iPhone download speeds among

three users: $\mathbf{x} = [1,\ 2,\ 3]$ Mbps and $\mathbf{y} = [1,\ 10,\ 100]$ Mbps. Among the large variety of choices we have in quantifying fairness, we can get fairness values, such as 0.33 for $\mathbf{x}$ and 0.01 for $\mathbf{y}$, or 0.86 for $\mathbf{x}$ and 0.41 for $\mathbf{y}$: $\mathbf{x}$ is viewed as 33 times more fair than $\mathbf{y}$, or just twice as fair as $\mathbf{y}$.

How many such "viewpoints" are there? What would *disqualify* a quantitative metric of fairness? Can they all be constructed from a set of **axioms**: simple statements taken as true for the sake of subsequent inference?

One existing approach to quantifying fairness of $\mathbf{x}$ is through a function f that maps $\mathbf{x}$ into a real number. These fairness measures are sometimes referred to as **diversity indices** in statistics. These range from simple ones, *e.g.*, the ratio between the smallest and the largest entries of $\mathbf{x}$, to more sophisticated functions, *e.g.*, Jain's index and the entropy function. Some of these fairness measures map $\mathbf{x}$ to a normalized range between 0 and 1, where 0 denotes the minimum fairness, 1 denotes the maximum fairness, and a larger value indicates more fairness. How are these fairness measures related? Is one measure "better" than any other? What other measures of fairness may be useful?

An alternative approach is the optimization-theoretic approach of α-fairness and the associated utility maximization problem. Given a set of feasible allocations, a maximizer of the α-fair (or, isoelastic) utility function satisfies the definition of α-fairness. We have seen two well-known examples: a maximizer of the log utility function ($\alpha = 1$) is proportionally fair, and a maximizer of the α-fair utility function as $\alpha \to \infty$ is max-min fair. It is often believed that $\alpha \to \infty$ is more fair than $\alpha = 1$, which is in turn more fair than $\alpha = 0$. But it remains unclear what it means to say, for example, that $\alpha = 2.5$ is more fair than $\alpha = 2.4$.

Clearly, these two approaches for quantifying fairness are different. One difference is the treatment of efficiency, or magnitude, of resources. On the one hand, α-fair utility functions are continuous and strictly increasing in each entry of $\mathbf{x}$, thus its maximization results in Pareto optimal resource allocations. On the other hand, scale-invariant fairness measures (ones that map $\mathbf{x}$ to the same fairness value as a normalized $\mathbf{x}$) are unaffected by the magnitude of $\mathbf{x}$, and $[1,\ 1]$ is as fair as $[100,\ 100]$. Can the two approaches be unified?

To address the above questions, we discuss an *axiomatic* approach to fairness measures. There is a set of five axioms, each of which is simple and intuitive, thus accepted as true for the sake of subsequent inference. They lead to a useful family of fairness measures. As explained in Advanced Material, the axioms are: the axiom of continuity, of homogeneity, of saturation, of partition, and of starvation. Starting with these five axioms, we can *generate* fairness measures. We derive a unique family of **fairness functions** f_β that include many known ones as special cases, and reveals new fairness measures corresponding to other ranges of β.

While we start with the approach of the fairness measure rather than the optimization objective function, it turns out that the latter approach can also be recovered from f_β. For $\beta \geq 0$, α-fair utility functions can be factorized as the product of two components: (a) our fairness measure with $\beta = \alpha$, and (b) a

function of the total throughput that captures the scale, or efficiency, of $\mathbf{x}$. Such a factorization quantifies a tradeoff between fairness and efficiency, addressing questions like what is the maximum weight that can be given to fairness while still maintaining Pareto efficiency. It also facilitates an unambiguous understanding of what it means to say that a larger α is "more fair" for general $\alpha \in [0, \infty)$.

20.2 A Long Answer

20.2.1 Constructing the fairness function

We are again condensing a vector into a scalar, a task we faced in opinion aggregation in Chapter 6. We first present a unified representation of the fairness measures constructed from five axioms (in Advanced Material), and provably the only family of fairness measures that can satisfy all the axioms. It is a family of functions parameterized by a real number β:

$$f_\beta(\mathbf{x}) = \text{sign}(1-\beta) \cdot \left[\sum_{i=1}^{n} \left(\frac{x_i}{\sum_j x_j} \right)^{1-\beta} \right]^{\frac{1}{\beta}}. \tag{20.1}$$

We see that only the distribution matters but not the magnitude of $\mathbf{x}$. This is due to one of the axioms, the Axiom of Homogeneity that says the fairness function f should be a homogeneous function where scaling of the arguments does not matter. In the rest of this section, we will show that this unified representation leads to many implications.

We first summarize the special cases in Table 20.1, where β sweeps from $-\infty$ to ∞ and $H(\cdot)$ denotes the entropy function:

$$H(\mathbf{x}) = -\sum_i x_i \log x_i.$$

For some values of β, known approaches to measure fairness are recovered, *e.g.*, Jain's index:

$$J(\mathbf{x}) = \frac{(\sum_i x_i)^2}{n \sum_i x_i^2}.$$

For $\beta \in (0, -1)$ and $\beta \in (-1, -\infty)$, new fairness measures are also revealed.

As an illustration, for two resource allocation vectors $\mathbf{x} = [1,\ 2,\ 3,\ 5]$ and $\mathbf{y} = [1,\ 1,\ 2.5,\ 5]$, we plot fairness $f_\beta(\mathbf{x})$ and $f_\beta(\mathbf{y})$ for different values of β in Figure 20.1. It is not trivial to decide which of these two vectors is more fair. Different values of β clearly change the fairness comparison ratio, and may even result in different fairness orderings: $f_\beta(\mathbf{x}) \geq f_\beta(\mathbf{y})$ for $\beta \in (-\infty, 4.6]$, and $f_\beta(\mathbf{x}) \leq f_\beta(\mathbf{y})$ for $\beta \in [4.6, \infty)$.

A person's fairness parameter β can be reverse-engineered, through a series of questions asking the person to pick what she perceives as a fairer allocation between two choices. Different people will have different β in their minds, but

Value of β	Fairness Measure	Known Names
$\beta \to \infty$	$-\max_i \left\{ \frac{\sum_i x_i}{x_i} \right\}$	Max ratio
$\beta \in (1, \infty)$	$-\left[(1-\beta) U_{\alpha=\beta} \left(\frac{\mathbf{x}}{w(\mathbf{x})} \right) \right]^{\frac{1}{\beta}}$	α-fair utility
$\beta \in (0, 1)$	$\left[(1-\beta) U_{\alpha=\beta} \left(\frac{\mathbf{x}}{w(\mathbf{x})} \right) \right]^{\frac{1}{\beta}}$	α-fair utility
$\beta \to 0$	$e^{H\left(\frac{\mathbf{x}}{w(\mathbf{x})}\right)}$	Entropy
$\beta \in (0, -1)$	$\left[\sum_{i=1}^n \left(\frac{x_i}{w(\mathbf{x})} \right)^{1-\beta r} \right]^{\frac{1}{\beta}}$	No name
$\beta = -1$	$\frac{(\sum_i x_i)^2}{\sum_i {x_i}^2} = n \cdot J(\mathbf{x})$	Jain's index
$\beta \in (-1, -\infty)$	$\left[\sum_{i=1}^n \left(\frac{x_i}{w(\mathbf{x})} \right)^{1-\beta r} \right]^{\frac{1}{\beta}}$	No name
$\beta \to -\infty$	$\min_i \left\{ \frac{\sum_i x_i}{x_i} \right\}$	Min ratio

Table 20.1 Some known fairness metrics are recovered as special cases of an axiomatic construction in (20.1). For $\beta \in (0, -1)$ and $\beta \in (-1, -\infty)$, new fairness measures of the Generalized Jain's Index are revealed. For $\beta \in [0, \infty]$ the fairness component of α-fair utility function is recovered. In particular, proportional fairness at $\alpha = \beta = 1$ is obtained from $\lim_{\beta \to 1} \frac{|f_\beta(\mathbf{x})|^\beta - n}{|1-\beta|}$, as we will verify in a homework problem.

we would hope that the same person will be self-consistent and keep the same β whether she is evaluating fairness of allocation to friends or foes.

Majorization is a partial order over vectors to study if the elements of vector $\mathbf{x}$ are less spread out than the elements of vector $\mathbf{y}$. Vector $\mathbf{x}$ is majorized by $\mathbf{y}$, and we write $\mathbf{x} \preceq \mathbf{y}$, if $\sum_{i=1}^n x_i = \sum_{i=1}^n y_i$, and

$$\sum_{i=1}^{d} x_i^{\uparrow} \leq \sum_{i=1}^{d} y_i^{\uparrow}, \text{ for } d = 1, \ldots, n, \tag{20.2}$$

where $x_i^{\uparrow}$ and $y_i^{\uparrow}$ are the ith elements of $\mathbf{x}^{\uparrow}$ and $\mathbf{y}^{\uparrow}$, sorted in ascending order. According to this definition, among the vectors with the same sum of elements, the one with the equal elements is the most majorizing vector. For example, $[1,\ 2,\ 3,\ 4] \preceq [1,\ 1,\ 2,\ 6]$.

Intuitively, $\mathbf{x} \preceq \mathbf{y}$ can be interpreted as $\mathbf{y}$ being a fairer allocation than $\mathbf{x}$. However, majorization alone cannot be used to define a fairness measure since it is only a partial order and may fail to compare vectors. Still, if resource allocation $\mathbf{x}$ is majorized by $\mathbf{y}$, it is desirable to have a fairness measure f such that $f(\mathbf{x}) \leq f(\mathbf{y})$. A function satisfying this property is known as **Schur-concave**. In statistics and economics, many measures of statistical dispersion or diversity

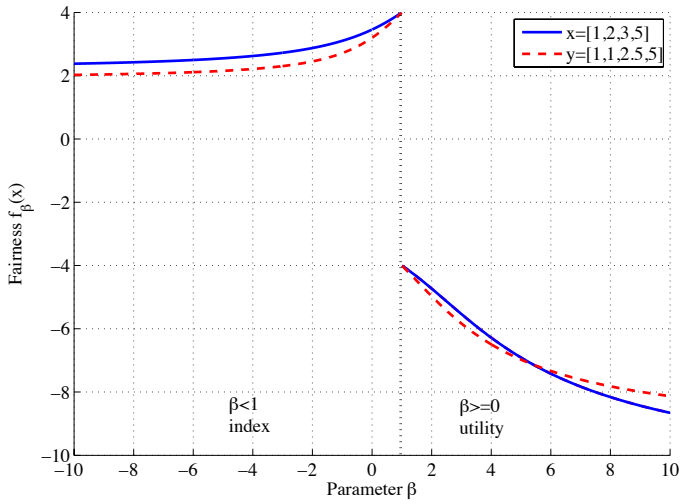


Figure 20.1 An example of fairness evaluation for two vectors of resource allocation over different β. It is not obvious which of these two four-user allocations $\mathbf{x}$ and $\mathbf{y}$ is more fair than the other. Indeed, the order is different depending on the value of β defining the exact shape of fairness measure.

are known to be Schur-concave, *e.g.*, the **Gini Coefficient** that we will see in a homework problem. Fairness measure (20.1) also is Schur-concave.

There are many other properties that can be proved about this unique family of axiomatically constructed fairness measures. One good use of axiomatic construction is that if a conclusion is undesirable, we often have a guess as to which axioms need to be perturbed. For example, it is obvious that equal allocation maximizes our fairness measure. There are two ways to avoid this naive view of fairness:

- Add user weights $\{q_1, q_2, \ldots, q_n\}$ in the axiomatic construction, which leads to another fairness measure that depends on both $\mathbf{x}$ and $\mathbf{q}$. We will largely skip this presentation, except in discussing Rawls' theory in Advanced Material.

- Incorporate efficiency into the picture, which can be carried out by deleting the axiom that states fairness does not depend on magnitude of the resource allocation vector. We will follow this path now.

20.2.2 What Does "Larger α is More Fair" Mean?

To answer this question, we first show a factorization of the α-fair utility function U_α. Re-arranging the terms, we have

$$\begin{aligned} U_{\alpha=\beta}(\mathbf{x}) &= \frac{1}{1-\beta} \left| f_\beta(\mathbf{x}) \right|^\beta \left(\sum_i x_i \right)^{1-\beta} \\ &= \left| f_\beta(\mathbf{x}) \right|^\beta \cdot U_\beta \left(\sum_i x_i \right), \end{aligned} \tag{20.3}$$

where $U_\beta \left(\sum_i x_i \right)$ is the uni-variate version of the α-fair utility function with $\alpha = \beta \in [0, \infty)$.

Equation (20.3) demonstrates that the α-fair utility functions can be factorized as the product of two components:

- A fairness measure, $\left| f_\beta(\mathbf{x}) \right|^\beta$
- An efficiency measure, $U_\beta \left(\sum_i x_i \right)$.

The fairness measure $\left| f_\beta(\mathbf{x}) \right|^\beta$ only depends on the normalized distribution, $\mathbf{x}/(\sum_i x_i)$, of resources, while the efficiency measure is a function of only the sum resource $\sum_i x_i$.

Allocation:	$\mathbf{x}$	
	↙	↘
Factorize:	$\mathbf{x}/\sum_i x_i$	$\sum_i x_i$
	↓	↓
Measure:	$f_\beta \left(\mathbf{x}/\sum_i x_i \right)$	$U_\beta \left(\sum_i x_i \right)$
	↘	↙
Combine:	$U_{\alpha=\beta}(\mathbf{x})$	

Table 20.2 An illustration of the factorization of the α-fair utility functions into a fairness component of the normalized resource distribution and an efficiency component of the sum resource.

The factorization of α-fair utility functions is illustrated in Table 20.2. Decoupling into these two components helps understand issues such as fairness-efficiency tradeoff and the feasibility of $\mathbf{x}$ under a given constraint set. For example, an allocation vector that maximizes the α-fair utility with a larger α may not be less efficient, because the α-fair utility incorporates both fairness and efficiency at the same time.

Guided by the product form of (20.3), we consider the following **welfare function**: a scalarization of the maximization of two objectives: fairness *and*

efficiency:

$$\Phi_\lambda(\mathbf{x}) = \lambda \ell\left(f_\beta\left(\mathbf{x}\right)\right) + \ell\left(\sum_i x_i\right), \tag{20.4}$$

where $\beta \in [0, \infty)$ is fixed, and $\lambda \in [0, \infty)$ absorbs the exponent β in the fairness component of (20.3) and is a weight specifying the relative emphasis placed on the fairness. And ℓ is just a shorthand notation:

$$\ell(y) = \text{sign}(y) \log(|y|). \tag{20.5}$$

The use of the log function later recovers the product in the factorization of (20.3) from the sum in (20.4).

Remember that an allocation vector $\mathbf{x}$ is said to be **Pareto dominated** by $\mathbf{y}$ if $x_i \leq y_i$ for all i and $x_i < y_i$ for at least some i. An allocation is called **Pareto optimal** if it is not Pareto dominated by any other feasible allocations. All the Pareto optimal points form a Pareto optimal tradeoff curve, a concept we mentioned many times before and now defined in a rigorous way. To preserve the meaning of Pareto optimality, we require that if $\mathbf{y}$ Pareto dominates $\mathbf{x}$, then $\Phi_\lambda(\mathbf{y}) > \Phi_\lambda(\mathbf{x})$. If the relative emphasis on efficiency is sufficiently high, Pareto optimality of the solution can be maintained.

The necessary and sufficient condition on λ, such that $\Phi_\lambda(\mathbf{y}) > \Phi_\lambda(\mathbf{x})$ if $\mathbf{y}$ Pareto dominates $\mathbf{x}$, is that λ must be no larger than a threshold $\bar{\lambda}$, which turns out to be:

$$\bar{\lambda} = \left|\frac{\beta}{1-\beta}\right|. \tag{20.6}$$

A different notion of fairness, *i.e.,* a different β, leads to a different threshold $\bar{\lambda}$.

Consider the set of maximizers of (20.4) for λ in the range of (20.6):

$$\mathbb{P} = \left\{\mathbf{x} : \mathbf{x} = \arg\max_{\mathbf{x}\in\mathbb{R}} \Phi_\lambda(\mathbf{x}),\ \forall \lambda \leq \left|\frac{\beta}{1-\beta}\right|\right\}. \tag{20.7}$$

When weight $\lambda = 0$, the corresponding points in $\mathbb{P}$ are the most efficient. When weight $\lambda = \left|\frac{\beta}{1-\beta}\right|$, it turns out that the factorization in (20.3) becomes exactly the same as (20.4).

What we have shown is the following: α-fairness corresponds to the solution of an optimization that places the *maximum emphasis* on the fairness measure parameterized by $\beta = \alpha$ while preserving Pareto optimality. With this property, it indeed makes sense to say that larger α is more fair.

20.2.3 Fairness-efficiency unification

Now we return to the Axiom of Homogeneity that says f needs to be a homogeneous function, *e.g.*, $f_\beta([1,\ 1]) = f_\beta([a,\ a])$ for any $a > 0$. This axiom clearly takes efficiency out of the picture altogether. If fairness $F(\mathbf{x})$ satisfies a new set

of axioms that removes the Axiom of Homogeneity, we can show that it must be of the form

$$F_{\beta,\lambda}(\mathbf{x}) = f_\beta(\mathbf{x}) \cdot \left(\sum_i x_i \right)^{\frac{1}{\lambda}}, \tag{20.8}$$

where $\frac{1}{\lambda} \in \mathbb{R}$ is the "degree of homogeneity", the same as weighting the fairness component $f_\beta(\mathbf{x})$ by λ, and $f_\beta(\mathbf{x})$ is the fairness function (20.1). There are now two parameters for fairness measures: a real number β and a positive number λ.

The new fairness measure F may not be Schur-concave, and equal allocation may not be fairness-maximizing. For example, you can easily verify that $F(1,1) < F(0.2,10)$ for $\beta = 2$ and $\lambda = 0.1$.

This family of fairness measures unifies a wide range of existing fairness indices and utilities in diverse fields such as computer science, economics, sociology, psychology, and philosophy, including

- Jain's index
- Jasso index
- Theil index
- Atkinson index
- Shannon entropy
- Renyi entropy
- α fairness
- Foster-Sen welfare function
- p-norm.

These are all fairness measures that are global (*i.e.,* mapping a given allocation vector in a system to a single scalar) and decomposable (*i.e.,* subsystems' fairness values can be somehow collectively mapped into the overall system's fairness value).

20.3 Examples

20.3.1 Capacity allocation

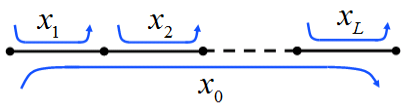


Figure 20.2 A linear network with L links and $n = L + 1$ sessions. All links have the same capacity of 1 unit. The long session is indexed as session 0. The short sessions are indexed by $i = 1, 2, \ldots, L$. Allocating capacity between the long session and the short sessions can strike different tradeoffs between fairness and efficiency, as driven by the choice of objective function.

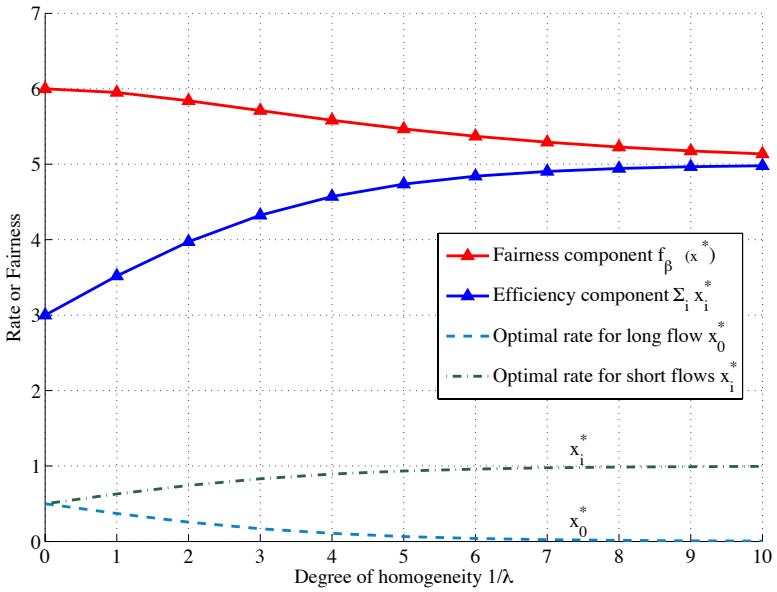


Figure 20.3 An example of generalized objective for capacity allocation. As the degree of homogeneity increases, λ and the emphasis on fairness component drops while the efficiency component rises. The capacity allocated to the long session becomes smaller.

As discussed in Chapter 14, the typical optimization problem modeling capacity allocation is as follows:

$$\begin{array}{ll} \text{maximize} & U(\mathbf{x}) = \sum_{i=1}^{n} U(x_i) \\ \text{subject to} & \mathbf{x} \in \mathcal{R} \\ \text{variables} & \mathbf{x}, \end{array} \tag{20.9}$$

where U is the utility function for each session, and $\mathcal{R}$ is a set of all feasible resource allocation vectors.

We now consider the classic example of a linear network with L links, indexed by $l = 1, \ldots, L$, and $n = L + 1$ sessions, indexed by $i = 0, 1, \ldots, L$, shown in Figure 20.2. Session $i = 0$ goes through all the links and sources $i \geq 1$ go through links $l = i$. All links have the same capacity of 1 unit. We denote by x_i the rate of session i.

We will illustrate two points: how a given $\mathbf{x}$ can be evaluated by different fairness measures, and how $F_{\beta,\lambda}(\mathbf{x})$ acting as the objective function broadens the range of trade-off between efficiency and fairness than just $U(\mathbf{x})$.

We formulate a generalized NUM problem in this linear network: maximization of $F_{\beta,\lambda}(\mathbf{x})$, a generalization of the α-fair utility function, under link capacity

constraints.

$$\begin{array}{ll} \text{maximize} & F_{\beta,\lambda}(\mathbf{x}) = f_\beta(\mathbf{x}) \cdot \left(\sum_i x_i\right)^{\frac{1}{\lambda}} \\ \text{such that} & x_0 + x_i \le 1, \ \ \forall i \\ \text{variables} & x_i \ge 0, \ \ \forall i. \end{array} \tag{20.10}$$

For $\beta \ge 0$ and $\lambda = (1-\beta)/\beta$, the optimal rate allocation maximizing (20.10) achieves α-fairness if we just take β in our fairness function to be α in isoelastic utility function. For $1/\lambda \ge \beta/(1-\beta)$, problem (20.10) is also concave after a logarithm change of objective function, *i.e.,* $\log F_{\beta,\lambda}(\mathbf{x})$.

Let us fix $\beta = 1/2$ and solve (20.10) for different values of $1/\lambda$. Figure 20.3 plots optimal rate allocations $\mathbf{x}^*$ (in terms of x_0^* and x_i^* for $i \ge 1$, since all x_i^* are the same for $i = 1, 2, \ldots, L$, by symmetry), their fairness components $f_\beta(\mathbf{x}^*)$, and their efficiency components $\sum_i x_i^*$, all against $1/\lambda$. As $1/\lambda$ (the weight on the efficiency component) grows, the efficiency component's value $\sum_i x_i^*$ increases and skews the optimal rate allocation away from the equal allocation: the long session x_0 in the linear network gets penalized, while short sessions x_i are favored. At the same time, the fairness component of the objective function decreases.

20.3.2 Taxation fairness

An interesting application of fairness is evaluating the fairness of taxation schemes. Let us denote the pre-tax personal income in the population by vector $\mathbf{x}$, and the tax amount by vector $\mathbf{c}$. We can evaluate taxation fairness in two ways. An obvious one is by comparing the fairness of the after-tax income distribution to that of the pre-tax income distribution:

$$\frac{f_\beta(\mathbf{x} - \mathbf{c}(\mathbf{x}))}{f_\beta(\mathbf{x})}. \tag{20.11}$$

But this is not the *only* question of fairness involved. If a few people shoulder most of the tax burden, and many people pays no tax at all, that can be unfair too. This second view looks at the fairness of the tax distribution itself:

$$\frac{f(\mathbf{c}(\mathbf{x}))}{f_\beta(\mathbf{x})}. \tag{20.12}$$

If you read the editorials of the *The New York Times*, it is often the first metric that is mentioned: more redistribution of wealth is needed by taxing the high income population more, for otherwise it is too far from equal distribution. If you read the editorials in *The Wall Street Journal*, it is often the second one mentioned: almost half of the U.S. population does not even pay any federal income tax at all (and some pay negative amount through benefits checks), while the top earners pay a disproportionately large share of the overall income tax.

To visualize the tradeoff between these (20.11) and (20.12), we can put them on two axes and look at the impact of different personal income tax rates (on the highest income bracket) on both. We use the US income and tax distributions detailed in Tables 20.3 and 20.4. Of course, the tradeoff curves are different for

Tax rate (%)	Single Bracket ($)	Married Bracket ($)
10	0-8375	0-16750
15	8376-34000	16751-68000
25	34001-82400	68001-137300
28	82401-171850	137301-209250
33	171851-373650	209251-373650
35	373651+	373651+

Table 20.3 U.S. federal income tax rates in 2010.

Bracket ($)	Users
0-4000	12214
4000-8375	270475
8376-34000	426869
34001-82400	221351
82401-171850	41703
171851-373650	16746
373651+	10642
Total	1000000

Table 20.4 Distribution of incomes by tax bracket, based on 2007 U.S. income distribution (source: IRS).

different β. We show a typical curve for $\beta \in [0, 1]$ in Figure 20.4, since most of the common fairness metrics concentrate around that range of β.

The slope over the range of 25% – 50% for the highest tax bracket in Figure 20.4, *i.e.,* the "exchange rate" between these two axes, is around 1. This illustrates one reason why this debate is hard: for each 1% increase in income distribution fairness, there is a 1% decrease in tax revenue distribution fairness.

Another reason is the scale of the two axes: the y-axis value is small: more than half of the fairness in tax revenue contribution is lost, yet the x-axis value is also small. It is difficult to improve fairness in income distribution through taxation.

Of course, this discussion is inadequate for the purpose of assessing taxation fairness, for many reasons:

- There are other taxes intertwined with personal income tax's impact, *e.g.*, payroll tax, corporate tax, alternative minimum tax, sales tax, property tax, estate tax, and the various deductions that can be taken. Some of these are taxed multiple times too, *e.g.*, investment income in corporations is taxed twice at the federal level in the US. Some countries also impose

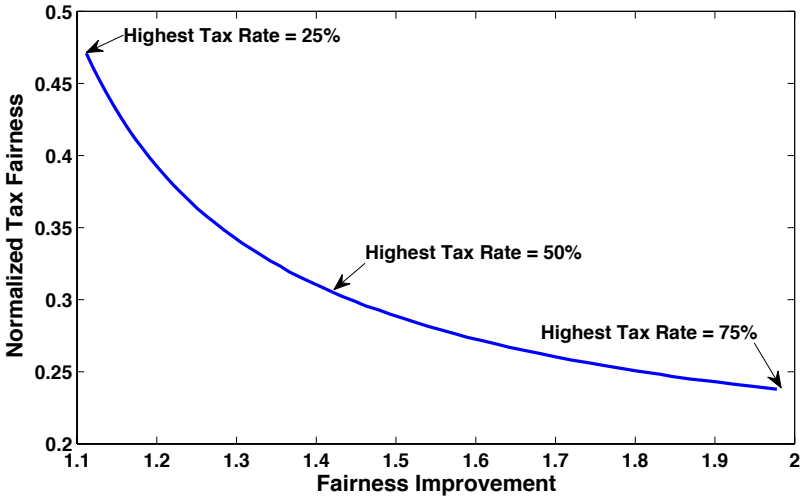


Figure 20.4 Fairness of tax vs. fairness of post-tax income for $\beta = 0.4$. The x-axis is $\frac{f_\beta(\mathbf{x}-\mathbf{c}(\mathbf{x}))}{f_\beta(\mathbf{x})}$. The y-axis is $\frac{f(\mathbf{c}(\mathbf{x}))}{f_\beta(\mathbf{x})}$. Each point on the curve corresponds to a different tax rate on the highest bracket. The slope between the point of 25% tax rate and that of 50% tax rate is about 1, highlighting one of the challenges in this debate: for one percent of income distribution fairness improvement, there needs to be one percent of reduction in the taxation fairness. The small values on both axes further underscores that the debate is touchy.

sales taxes of many kinds. For example, in parts of Canada, salted peanuts are taxable food even though neither peanuts nor salts have sales tax.

- Fairness of taxation should be positioned in a feedback control loop with people's incentives and reactions considered. Raising the tax rate beyond a certain point *reduces* the tax revenue.

- How the collected tax is spent is as important as how much can be collected, including questions on the effectiveness, efficiency, accountability, and flexibility of individual choices in the spendings by government bureaucrats.

- Fairness is often tied to the difference (and the perception of such difference) in the society between income derived from merits and income derived from corruption. It is tied to the upward mobility in the society by one's own effort to realize each person's different potential to the fullest.

- An even deeper debate is on the roles of government versus individuals (and private institutions) in providing solutions to society's problems. Tax dollars increase the power of government decisions and reduce the self-reliance of individuals.

20.4 Advanced Material

Across many disciplines, fairness has been extensively studied by raising different kinds of questions and answering them with different methodologies. For example:

- Different indices, from the Atkinson index to the Gini index, have been studied at the intersection of economics, statistics, and sociology.
- Bargaining has been studied at the intersection of economics, sociology, and political philosophy.
- The ultimatum game, dictator game, divide a dollar game, and their extension of fair cake cutting has been studied at the intersection of economics, computer science, and psychology. This subject, together with opinion aggregation and voting theory form the field known as **social choice theory**.
- Fairness is not just about the outcome, but also the process. Procedural fairness has been studied extensively in law and psychology. Dictatorship and arbitrariness have long been practiced in the Orwellian name of fairness, and individual choices limited by governments under the disguise of social welfare maximization.

We will see some examples of the above list in homework problems. Below we focus on a brief discussion of Rawls' theory of justice as distributive fairness.

20.4.1 Rawls' theory of justice and distributive fairness

In the 20th century political philosophy, Rawls' work on fairness has been one of the most influential since its original publication in 1971. It starts with the "original position" behind the "veil of ignorance," where each person does not know where in the society she will land. This is similar to the approach of "one cuts the cake, the other selects a slice first" in the problem of fair cake-cutting that we will see in a homework problem. You can probably also sense the connection of this assumption to the maxmin fairness already.

The arguments posed by Rawls are based on two fundamental principles (*i.e.,* axioms stated in English rather than mathematics), as described in his 2001 restatement:

1. "Each person is to have an equal right to the most extensive scheme of equal basic liberties compatible with a similar scheme of liberties for others."
2. "Social and economic inequalities should be arranged so that they are both (a) to the greatest benefit of the least advantaged persons, and (b) attached to offices and positions open to all under conditions of equality of opportunity."

The first principle governs the distribution of *liberties* and has priority over the second principle. But suppose we interpret it as a principle of distributive fairness in allocating limited *resources* among users. It can now be captured as a theorem (rather than an axiom) that says any fairness measures satisfying Axioms 1-5

will satisfy the following property: adding an equal amount of resource c to each user will only increase the fairness value, *i.e.,*

$$f_\beta\left(\mathbf{x}+c\cdot\mathbf{1}_n,\mathbf{q}\right)\geq f_\beta\left(\mathbf{x},\mathbf{q}\right),\ \text{for } q_i=\frac{1}{n},\ \forall c\geq 0,\ \forall\beta,$$

where equal weights $q_i=\frac{1}{n}$ can be viewed as a quantification of "equal right" in the first principle of Rawls' theory. Of course, for a general weight vector $\mathbf{q}$ or if the λ weight between fairness and efficiency is introduced, the above property only holds for c up to some upper limit.

The second part is the celebrated **difference principle**: "to the greatest benefit" rather than "to the greatest relative benefit". So $[100,\ 101]$ is more fair than $[1,\ 1]$. It is an approach different from strict **egalitarianism** (since it is on the absolute value of the least advantaged user rather than the relative value) and **utilitarianism** (when narrowly interpreted where the utility function does not capture fairness).

Now the Difference Principle can be axiomatically constructed as a special case of a continuum of generalized notions trading off fairness with efficiency. This is best illustrated by annotating Rawls' own graph in Figure 20.5. The point representing the difference principle is the consequence of concatenating two steps of pushing to the *extremum* on both β and λ in (20.1): let $\beta\to\infty$, and make λ as large as possible while retaining Pareto efficiency, *i.e.,* $\lambda\to\bar{\lambda}$. If either β is finite (*e.g.*, $\lambda=1$), or λ is smaller than $\bar{\lambda}$ (more emphasis on efficiency), we will have a fairness notion that is not as strong as Rawls' difference principle.

Now, back to the introductory discussion at the beginning of this chapter. What would be your threshold x above which $[x,\ x+1]$ is more fair than $[1,\ 1]$? Questions like this can help reverse engineer your β and λ.

20.4.2 Axioms

Let $\mathbf{x}$ be a resource allocation vector with n non-negative elements. A fairness measure is a sequence of mapping $\{f^n(\mathbf{x}),\forall n\in\mathbb{Z}_n\}$ from n-dimensional vectors $\mathbf{x}$ to real numbers, called fairness values, *i.e.,* $\{f^n:\mathbb{R}_+^n\to\mathbb{R},\forall n\in\mathbb{Z}_+\}$. To simplify the notation, we suppress n in $\{f^n\}$ and denote them simply as f. We introduce the following set of axioms about f, whose explanations are provided after the statement of each axiom.

1). *Axiom of Continuity.* Fairness measure $f(\mathbf{x})$ is continuous on $\mathbb{R}_+^n$, for all $n\in\mathbb{Z}_+$.

Axiom 1 is intuitive: A slight change in resource allocation shows up as a slight change in the fairness measure.

2). *Axiom of Homogeneity.* Fairness measure $f(\mathbf{x})$ is a homogeneous function:

$$f(\mathbf{x})=f(t\cdot\mathbf{x}),\quad\forall\ t>0. \tag{20.13}$$

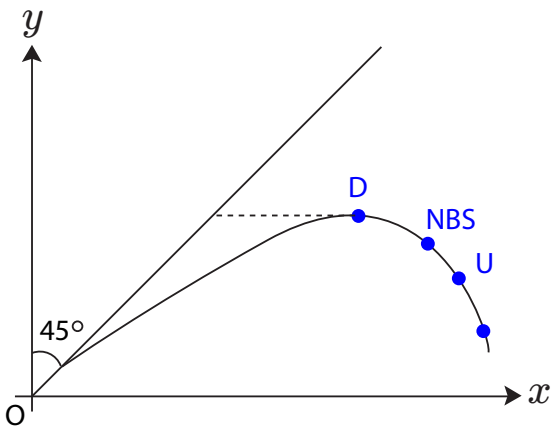


Figure 20.5 An annotated version of the only quantitative graph in Rawls 2001 book. Since the efficiency frontier between a "more advantaged group" user x and a "less advantaged group" user y does not pass through 45 degree line, strict egalitarian point is the origin. Difference principle generates a point D on the efficient frontier, whereas sum utility maximization generates point U. Nash Bargaining Solution (as discussed in Chapter 6) point N may lie between D and U. Fairness function $F_{\beta,\lambda}(\mathbf{x}, \mathbf{q})$ can generate any point on the efficient frontier when $\lambda \leq \bar{\lambda}$.

Without loss of generality, for $n = 1$, we take $|f(x_1)| = 1$ for all $x_1 > 0$, *i.e.*, fairness is a constant for a one-user system.

Axiom 2 says that the fairness measure is independent of the unit of measurement or magnitude of the resource allocation. Therefore, for an optimization formulation of resource allocation, the fairness measure $f(\mathbf{x})$ alone cannot be used as the objective function if efficiency (which depends on magnitude $\sum_i x_i$) is to be captured. As we saw in this chapter, this axiom can be readily removed.

3). *Axiom of Saturation.* Equal allocation's fairness value is independent of number of users as the number of users becomes large, *i.e.,*

$$\lim_{n\to\infty} \frac{f(\mathbf{1}_{n+1})}{f(\mathbf{1}_n)} = 1. \tag{20.14}$$

This axiom is a technical condition used to help ensure the *uniqueness* of the fairness measure. Note that it is *not* stating that equal allocation is the most fair.

A primary motivation for quantifying fairness is to allow a comparison of fairness values. Therefore, we must ensure well-definedness of the ratio of fairness measures, as the number of users in the system increases. Axiom 4 states that fairness comparison is independent of the way the resource allocation vector is reached as the system grows.

4). *Axiom of Partition.* Consider a partition of a system into two sub-systems.

Let $\mathbf{x} = [\mathbf{x}^1, \mathbf{x}^2]$ and $\mathbf{y} = [\mathbf{y}^1, \mathbf{y}^2]$ be two resource allocation vectors, each partitioned and satisfying $\sum_j x_j^i = \sum_j y_j^i$ for $i = 1, 2$. There exists a *mean function* h such that their fairness ratio is the mean of the fairness ratios of the subsystems' allocations, for all partitions such that the sum resources of each subsystem are the same across $\mathbf{x}$ and $\mathbf{y}$:

$$\frac{f(\mathbf{x})}{f(\mathbf{y})} = h\left(\frac{f(\mathbf{x}^1)}{f(\mathbf{y}^1)}, \frac{f(\mathbf{x}^2)}{f(\mathbf{y}^2)}\right). \tag{20.15}$$

According to the axiomatic theory of mean function, a function h is a mean function if and only if it can be expressed as follows:

$$h = g^{-1}\left(\sum_{i=1}^{2} s_i \cdot g\left(\frac{f(\mathbf{x}^i)}{f(\mathbf{y}^i)}\right)\right), \tag{20.16}$$

where g is any continuous and strictly monotonic function, referred to as the Kolmogorov-Nagumo generator function, and $\{s_i\}$ are the positive weights such that $\sum_i s_i = 1$.

5). *Axiom of Starvation.* For $n = 2$ users, we have $f(1, 0) \leq f(\frac{1}{2}, \frac{1}{2})$, *i.e.,* starvation is no more fair than equal allocation.

Axiom 5 is the only axiom that involves a *value* statement on fairness: starvation is no more fair than equal distribution for two users. It specifies an increasing direction of fairness and is used to ensure uniqueness of $f(\mathbf{x})$. We could have picked a different axiom to achieve similar effects, but the above axiom for just 2 users and involving only starvation and equal allocation is the weakest such statement, thus the "strongest axiom."

With the five axioms presented, we now discuss some implications of Axiom 4. This axiom can construct fairness measure f from lower dimensional spaces. If we choose $\mathbf{y} = [w(\mathbf{x}^1),\ w(\mathbf{x}^2)]$ (where $w(\mathbf{x}^i) = \sum_j x_j^i$ is the sum of the resource in sub-system i) in Axiom 4 and use the fact that $|f(w(\mathbf{x}^i))| = 1$ for scalar inputs as implied by Axiom 2, we can show that Axiom 4 implies a hierarchical construction of fairness. This in turn allows us to derive a fairness measure $f : \mathbb{R}_+^n \to \mathbb{R}$ of n users recursively (with respect to a generator function $g(y)$) from lower dimensions, $f : \mathbb{R}_+^k \to \mathbb{R}$ and $f : \mathbb{R}_+^{n-k} \to \mathbb{R}$ for integer $0 < k < n$.

The functions g giving rise to the same fairness measures f may not be unique, *e.g.*, logarithm and power functions. The simplest case is when g is identity and $s_i = 1/n$ for all i. A natural parameterization of the weight s_i in (20.15) is to choose the value proportional to the sum resource of sub-systems:

$$s_i = \frac{w^\rho(\mathbf{x}^i)}{\sum_j w^\rho(\mathbf{x}^j)},\ \forall i, \tag{20.17}$$

where $\rho \geq 0$ is an arbitrary exponent. It turns out that the parameter ρ can be chosen such that the hierarchical computation is independent of the partition as mandated in Axiom 4.

By definition, a set of axioms is true, as long as the axioms are consistent. As we saw in this chapter, there exists a fairness measure $f(\mathbf{x})$ satisfying Axioms 1–5. Furthermore, *uniqueness* results contain two parts. First, we show that, from any generator function $g(y)$, there is a unique $f(\mathbf{x})$ thus generated. Such an $f(\mathbf{x})$ is a well-defined fairness measure if it also satisfies Axioms 1–5. We can further show that only *logarithm* and *power* functions are possible generator functions. Therefore, we find, in closed-form, all possible fairness measures satisfying axioms 1-5, as shown in (20.1).

Further Readings

Perhaps no chapter in this book has as much intellectual diversity as this one. There are literally tens of thousands of papers on the subject of fairness from different disciplines.

1. In political philosophy, the classic and influential book by Rawls has a second edition:

[Raw01] J. Rawls, *Justice as Fairness: A Restatement*, Harvard University Press, 2001.

2. In computer science and economics, cake cutting has been the standard problem in the study of fairness. A comprehensive survey, including connections to auction and voting, is provided in the following book:

[BT96] S. J. Brams and A. D. Taylor, *Fair Division: From Cake-cutting to Dispute Resolution*, Cambridge University Press, 1996.

3. More than just the outcome, fairness is also about the process. In psychology, sociology, and legal study, a historical account and survey of procedural fairness can be found in:

[LT88] E. A. Lind and T. R. Tyler, *The Social Psychology of Procedural Justice*, Springer, 1988.

4. On taxation and reaction to taxes as a dynamic system, the following paper discusses the impact of social perception and presents a differential equation model of the feedback loop between taxation by the government and reaction by the people:

[AA05] A. Alesina and G. M. Angeletos, "Fairness and redistribution," *American Economic Review*, 2005.

5. The axiomatic development follows this paper, which also discusses the connections with other branches on the study on fairness:

[LC11] T. Lan and M. Chiang, "An axiomatic theory of fairness," *Princeton*

University Technical Report, 2011.

Problems

20.1 *Fairness-efficiency unification* ⋆⋆

The definition of fairness measure is given in Section 20.2.1 as follows:

$$f_\beta(\mathbf{x}) = \begin{cases} \text{sign}(1-\beta)\Big(\sum_{i=1}^n \big(\frac{x_i}{w(\mathbf{x})}\big)^{1-\beta}\Big)^{\frac{1}{\beta}} & \text{, if } \beta \neq 0 \\ \exp\Big(-\sum_{i=1}^n \frac{x_i}{w(\mathbf{x})} \log \frac{x_i}{w(\mathbf{x})}\Big) & \text{, if } \beta = 0. \end{cases}$$

where $w(x) \equiv \sum_{j=1}^n x_j$.

(a) Prove that $\lim_{\beta\to 0} f_\beta(\mathbf{x}) = f_0(\mathbf{x})$.

(b) Consider two allocation vectors $\mathbf{x} = [0.1,\ 0.2,\ 0.3,\ 0.6], \mathbf{y} = [0.2, 0.2, 0.8, 0.9]$. Plot $f_\beta(\mathbf{x}), f_\beta(\mathbf{y})$ as functions of $-10 \leq \beta \leq 10$ in one figure.

(c) Fix $\beta = 0.5$, plot fairness-efficiency measures $F_{\beta,\lambda}(\mathbf{x}), F_{\beta,\lambda}(\mathbf{y})$, as functions of $0 \leq \frac{1}{\lambda} \leq 1$, in one figure.

20.2 *Atkinson index and Gini coefficient* ⋆

According to U.S. census bureau, the distribution of income of households in U.S in 2009 is as follows

Income of households

Income	Percentage
\$0 to \$14,999	10.6
\$15,000 to \$24,999	11.0
\$25,000 to \$34,999	10.3
\$35,000 to \$49,999	14.0
\$50,000 to \$74,999	18.8
\$75,000 to \$99,999	12.4
\$100,000 to \$149,999	13.4
\$150,000 to \$199,999	5.4
\$200,000 and over	4.4

Assume uniform distribution within each income interval, except for the interval of income over \$200,000, where we assume all houses in this interval have the same income of \$200,000.

(a) Plot the Atkinson index w.r.t. $0 \leq \epsilon \leq 10$. The Atkinson index of allocation vector $\mathbf{x} = (x_1, \cdots, x_N)$ is defined as follows:

$$A_\epsilon(x_1, \cdots, x_N) = \begin{cases} 1 - \frac{1}{\mu}\left(\frac{1}{N}\sum_{i=1}^{N} x_i^{1-\epsilon}\right)^{1/(1-\epsilon)} & \text{, if } \epsilon \in [0,1) \cup (1,\infty) \\ 1 - \frac{1}{\mu}\left(\prod_{i=1}^{N} x_i\right) & \text{, if } \epsilon = 1. \end{cases}$$

(b) Calculate the Gini coefficient.

20.3 *Multi resource fairness* ⋆⋆⋆

The fairness-efficiency functions introduced in the lecture notes have the form

$$F_{\beta,\lambda}(\mathbf{x}) = \operatorname{sign}(1-\beta)\left(\sum_{i=1}^{n}\left(\frac{x_i}{\sum_{j=1}^{n} x_j}\right)^{1-\beta}\right)^{1/\beta}\left(\sum_{i=1}^{n} x_i\right)^{\lambda}, \tag{20.18}$$

where $\mathbf{x}$ is an n-dimensional resource allocation vector (*i.e.,* each entry x_i is the amount of a resource given to person i, $i = 1, 2, \ldots, n$). But in some contexts, the allocation of one resource is not enough. For instance, consider a datacenter utilized by two people. Each user runs one type of job, and the two types of jobs have different resource needs. Both require memory and CPU (processing power), but user A's jobs require 1 GB of memory and 2 MIPS of CPUs per job, while user B's jobs require 3 GB of memory and 1.5 MIPS CPUs per job. These resources are limited: there are 8 GB of available memory and 12 MIPS of CPUs. Clearly, just allocating memory or just allocating CPUs is not enough: we need to consider the fairness of *both* resource allocations.

This homework question introduces two different ways of measuring the fairness of multi-resource allocations, as in the datacenter example above. First, one could just measure the fairness of the number of jobs allocated to each user. Each user is allocated enough resources to complete this number of jobs–for instance, if user A is allocated 2 jobs, she receives 2 GB of memory and 4 MIPS of CPUs. The resource allocation vector in (20.18) is just the number of jobs assigned to each user–for instance, if user A is assigned 2 jobs and user B 3 jobs, the fairness of this allocation is

$$\operatorname{sign}(1-\beta)\left(\left(\frac{2}{2+3}\right)^{1-\beta} + \left(\frac{3}{2+3}\right)^{1-\beta}\right)^{1/\beta}(2+3)^{\lambda}. \tag{20.19}$$

But this approach misses the heterogeneity of the resource requests among the users. The second way of measuring multi-resource fairness involves *dominant shares.* These are defined as the maximum fraction of each resource received by the user. For instance, if user A gets 2 jobs, she receives 2 GB of memory and 4 MIPS of CPUs. Then A receives 1/4 of the available memory but 1/3 of the CPUs. User A's *dominant resource* is CPUs, and her *dominant share* is 1/3. These dominant shares are then taken as the resource allocation vector–if user

A's dominant share is 1/3 and user B's is 2/3, the allocation vector $\mathbf{x}$ in (20.18) is $[1/3,\ 2/3]$.

(a). What is user B's dominant resource? Calculate the dominant shares for users A and B in terms of x_A and x_B, the number of jobs allocated to users A and B respectively.

(b). Formulate the maximization problem for multi-resource fairness in the data-center example above, using both fairness on jobs and fairness on dominant shares (*i.e.,* write down two formulations, one for fairness on jobs and one for fairness on dominant shares). Use your answers to part (a) and (20.18) to write down the objective function for maximizing fairness on dominant shares. (Hint: The constraints in both optimization problems are resource capacity constraints, and the optimization variables are x_A and x_B.)

(c). Numerically solve for the optimal resource allocation according to your formulations in part (b), with the resource requirements given above and $\beta = 0.5$ and $\lambda = 1$. (Non-integer numbers of jobs are allowed). Are the optimal allocations the same? Which do you think is more "fair"?

20.4 *Cake-cutting fairness* ⋆⋆⋆

"One cuts, the other selects" is a well-known procedure of dividing a cake for two people such that both value their own share to be at least half of the whole cake. Alice first cut the cake into two pieces with each piece having the same value to her, and then Bob selects among the two pieces.

Can you come up with a procedure of dividing a cake for three participants so that they all value their own share to be at least one third of the whole cake?

(Hint: First divide the cake into three pieces whose values are equal for one participant.)

(More detail can be found at the following survey: S. J. Brams, M. A. Jones, and C. Klamler, "Better ways to cut a cake", *Notice of the American Mathematics Society*, vol. 53, no. 11, pp. 1314-1321, December 2006.)

20.5 *The ultimatum game* ⋆⋆

The ultimatum game is a game where two players interact to decide how to divide a sum of money between them. The first player proposes how to divide and the second player can either accept or reject this proposal. If the second player rejects, neither player receives anything; If the second player accepts, the money is split according to the proposal. The game is played only once so that reciprocation is not an issue.

Consider an ultimatum game where the proposer, Alice and the accepter, Bob, are to divide a one-foot long sandwich. Alice knows that Bob will not accept offer less than x foot, however she is not certain about x and only has the belief about

the probability density function of x:

$$f(x) = \begin{cases} 4x & \text{if } x < 0.5 \\ 4(1-x) & \text{if } x \geq 0.5 \end{cases}$$

How will Alice propose to split the sandwich, and what is the expected share she receives?

Notes

Index