Title
Toward cheat-proof networking

Author
Raghavan, Barath

Publication Date
2009

Peer reviewed|Thesis/dissertation

UNIVERSITY OF CALIFORNIA, SAN DIEGO

TOWARD CHEAT-PROOF NETWORKING

A Dissertation submitted in partial satisfaction of the

requirements for the degree Doctor of Philosophy

in

Computer Science

by

Barath Raghavan

Committee in charge:

Professor Alex C. Snoeren, Chair
Professor Mihir Bellare
Professor Rene Cruz
Professor Stefan Savage
Professor Scott Shenker

2009

The Dissertation of Barath Raghavan is approved, and it is acceptable in quality and form for publication on microfilm and electronically:

Chair

University of California, San Diego

2009

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGMENTS

Alvin AuYoung, Neil Alldrin, Jeannie Albrecht, Chris Tuttle, Jay Chen, Yu-Chung Cheng, Cristian Estan, Sumeet Singh, Ramana Kompella, Justin Ma, Michael Vrable, Ryan Braud, Patrick Verkaik, Chris Kanich, Marti Motoyama, and Tom Ristenpart were co-conspirators in this experiment that was grad school, and made the journey worthwhile. I'm lucky to have had the opportunity to collaborate with and learn from Kashi Vishwanath, Ken Yocum, Saurabh Panjwani, Anton Mityagin, Eike Kiltz, John McCullough, Mohammad Al-Fares, and Siva Radhakrishnan.

George Varghese, Geoff Voelker, Stefan Savage, Amin Vahdat, Mihir Bellare, Rene Cruz, Scott Shenker, and David Wetherall always had an open door and taught me so much in every discussion. I'm grateful to Alex Snoeren who took a chance on me, put up with me over the years, and instilled in me his passion for rigor in research. Mariel Vazquez and Javier Arsuaga inspired me to do research when I was a wayward undergrad, an inspiration that has yet to let up. I have looked to the wisdom and humility of Travis Newhouse, Sriram Ramabhadran, and Yoshi Kohno and hope to continue to learn from their example.

The angular riffs and odd time signatures of Jay Byron, Tim McDaniel, Adam Streed, and Jörg Rieckermann were an indispensible part of my graduate education.

And most important of all, Jenny Trujillo and my parents and sister have provided me with the love, encouragement, and helpful nudges without which I couldn't have made it through.

Chapter 3 is in part a reprint of material that appears in the Proceedings of the ACM SIGCOMM Conference, 2007 by Barath Raghavan, Kashi Vishwanath, Sriram Ramabhadran, Kenneth Yocum, and Alex Snoeren. Chapter 4 is in part a reprint of material that appears in the Proceedings of the ACM SIGCOMM Workshop on Hot Topics in Networks, 2006 by Barath Raghavan and Alex Snoeren. The dissertation author was the primary investigator and author of these works.

VITA

2009	Doctor of Philosophy, Computer Science and Engineering University of California, San Diego La Jolla, CA
2006	Master of Science, Computer Science and Engineering University of California, San Diego La Jolla, CA
2002	Bachelor of Science, Electrical Engineering and Computer Science University of California, Berkeley Berkeley, CA

PUBLICATIONS

B. Raghavan, P. Verkaik, and A. Snoeren. Secure and Policy-Compliant Source Routing. *IEEE/ACM Transactions on Networking*, Aug. 2009.

B. Raghavan, T. Kohno, A. C. Snoeren, and D. Wetherall. Enlisting ISPs to Improve Online Privacy: IP Address Mixing by Default. In *Proceedings of the Privacy Enhancing Technologies Symposium*, 2009.

X. Hua, D. Nguyen, B. Raghavan, J. Arsuaga, and M. Vazquez. Random State Transitions of Knots: A First Step Towards Modeling Unknotting by Type II Topoisomerases. *Topology and its Applications*, 154(7), 2007.

B. Raghavan, K. Vishwanath, S. Ramabhadran, K. Yocum, and A. C. Snoeren. Cloud Control with Distributed Rate Limiting. In *Proceedings of ACM SIGCOMM*, 2007.

B. Raghavan, S. Panjwani, and A. Mityagin. Analysis of the SPV Secure Routing Protocol: Weaknesses and Lessons. *ACM SIGCOMM Computer Communication Review*, Apr. 2007.

B. Raghavan and A. C. Snoeren. Decongestion Control. In *Proceedings of ACM SIGCOMM Workshop on Hot Topics in Networks*, 2006.

E. Kiltz, A. Mityagin, S. Panjwani, and B. Raghavan. Append-Only Signatures. In *Proceedings of ICALP*, 2005.

B. Raghavan and A. C. Snoeren. A System for Authenticated Policy-Compliant Routing. In *Proceedings of ACM SIGCOMM*, 2004.

A. C. Snoeren and B. Raghavan. Decoupling Policy from Mechanism in Internet Routing. In *Proceedings of ACM SIGCOMM Workshop on Hot Topics in Networks*, 2003.

B. Raghavan and A. C. Snoeren. Priority Forwarding in Ad Hoc Networks with Self-Interested Parties. In *Proceedings of the Workshop on Economics of Peer-to-Peer Systems*, 2003.

FIELDS OF STUDY

Major Field: Computer Science

Studies in Systems and Networking
Professor Alex C. Snoeren

ABSTRACT OF THE DISSERTATION

TOWARD CHEAT-PROOF NETWORKING

by

Barath Raghavan

Doctor of Philosophy in Computer Science and Engineering

University of California, San Diego, 2009

Professor Alex C. Snoeren, Chair

Over the last three decades the Internet has evolved from a network of a dozen academics to one that spans the globe and is the primary medium of electronic communication. When the two most fundamental Internet protocols—IP and TCP—were designed, they were intended to address the problems of the day: addressing, routing, reliable delivery, and, eventually, congestion. This dissertation studies how these protocols may be augmented to adjust to today's reality and cope with the possibility of cheating—the overuse of network resources—by network users.

We study cheating in the Internet in two contexts. First, we examine strictly limiting the resources a user or entity can consume in the network. Second, we study self-interested, greedy users who want to consume as many resources as possible. The key challenge in both of these contexts is not primarily in designing cheat-proof mechanisms, but in doing so while avoiding unwanted network or architectural overhead.

For the former, we develop the notion of Distributed Rate Limiting, which enables a network service provider to cap the aggregate bandwidth consumed by a user at different locations in the network. Distributed Rate Limiters operate at the network layer and aim to emulate the behavior of today's centralized limiters with low inter-limiter communication overhead. For

the latter—for developing a transport layer that not only copes with the vicissitudes of network application traffic, but also with the desires and motivations of network users—we develop the notion of Decongestion Control, a congestion control paradigm in which users attempt to maximize their individual throughput in the course of normal operation. In networking canon, dropped packets represent wasted resources, and thus traditional network congestion control protocols aim to avoid sending at a rate that induces packet loss. We study whether the benefits of a transport layer that embraces—rather than avoids—widespread packet loss and user self interest outweigh the potential loss in efficiency.

For both of these systems, we identify numerous potential benefits to and applications for both the network provider and network user alike, and develop a framework in which such systems can subsequently be evaluated. For Distributed Rate Limiting we identify two important metrics—the inter-flow fairness and the rate that a limiter can deliver under shifting traffic patterns—and evaluate how communication overhead impacts the algorithms we present with respect to these metrics. For Decongestion Control, we similarly identify and examine the principal challenges—that the protocol must provide performance not worse than TCP and that its widespread use must not cause congestion collapse.

Chapter 1

Introduction

While the protocols we use today in the Internet have remained essentially unchanged since their first standardization in the early 1980s, the Internet itself has changed dramatically. In 1981, when the current versions of TCP and IP were specified, there were 231 hosts in the Internet [119], connected by slow dedicated links between mainframes at a handful of select academic, research, government, and military installations. The users of the network were highly knowledgeable about the network itself, used it for non-profit purposes and in a spirit of community. Today's Internet is the boiler room of the world economy—a communication medium without which most major corporations could not function. This decades-long evolution of the Internet has left an important piece behind—the underlying protocols. Indeed, we still use IP and TCP today essentially unchanged from their specification in the original IETF Requests for Comment from September 1981 [152, 153]; we are driving Model Ts on the Autobahn. It is a testament to the vision shown by the designers of these original protocols that they have served us so well for so many years, but the world has changed sufficiently to warrant a re-evaluation.

The most glaring changes to the Internet over the last decade are ones of capacity and scale. The network now touches every remote region of the globe at remarkable bandwidths. Perhaps more impressively, the number of users on the network is in the billions. Beyond simply

the network's physical and geographic structure, the philosophy of the network has changed. The notion of the Internet as a community with shared goals, values, and interests is long gone. In its place, we have a communication medium, not a community, and as with any shared medium, there need be common interests and self-enforcement or, alternatively, exogenous constraints.

Many other mediums fit this paradigm. For example, the radio spectrum is a shared medium—broadcasters compete in auctions for frequency licences and are held to their allocations by the FCC. However, not all of the usable electromagnetic spectrum is strictly enforced—some, such as the 2.4 GHz region, are unlicensed. As a result, all users of unlicensed frequency spectrum must share the medium with each other; this requires cooperation or coordination. Without such cooperation, entities can negatively interfere with each other such as in the case of 802.11 wireless devices and microwave ovens.

When IP and TCP were designed, they were intended to address the problems of the day: addressing, routing, reliable delivery, and, eventually, congestion. At the time, network users were cooperative (even incidents such as the Internet worm of 1988 were not borne of malicious intent [174]); this was an *assumption* that underpins protocol designs from that era. Today's challenges constanly remind us that the underlying assumptions of yesteryear may no longer apply. Instead, we must capture today's reality: in a shared medium, some users may not cooperate.

We study this lack of cooperation in two respects. The first is that of the microwave oven—failure or malice—in which the entity is, intentionally or not, stopping normal progress for others. The second is one that is more natural in a modern network, and can be designed for: self-interested, greedy users who want to consume as many resources as possible. We use the term "cheating" narrowly to define these behaviors: obstructive, excessive, or greedy consumption of network resources by an entity at one or more locations in the network.

The goal of this dissertation is to answer the following question: *what changes can be made at the network layer and the transport layer to effectively limit the ability of individuals in*

the Internet to unilaterally impose their own will upon the network? In other words, can we stop cheating—in this context the consumption of resources above and beyond that allocated for a single entity—by making changes at the network and transport layers of the Internet?

We explore two paths to this destination. The first views the Internet's architecture from the perspective of a network provider, and examines what changes can be made to ensure that cheating parties can be held to their aforementioned resource limit. Since a provider cannot make changes at the host, all changes must be imposed upon already-existing traffic running today's protocols such as TCP and UDP. The second path views the Internet holistically, allowing for changes at the end host, and examines how to design for self interest. As a result, we study what changes can to be made both by Internet providers and end hosts to mitigate cheating and improve operation of the network.

1.1 Approach

To understand the consequences of our designs at both the network layer and the transport layer, we view the network holistically. First, to select the right metrics for study we must understand the goals of Distributed Rate Limiting and Decongestion Control. Broadly, we aim to prevent entities from consuming more than their share of resources—for some externally defined notion of "share"—in the network. At the same time, our goal is to maintain some consistency of behavior with respect to the current network—we want to ensure that the changes are deployable. Our goal is not to merely design systems to be cheat proof, but rather to design practical, efficient systems that happen to also be cheat proof. As such, for each system we require definitions of constraints that tie the hands of the user of the network and the operator of the network alike.

1.1.1 Evaluation

In the context of Distributed Rate Limiting we define that the behavior of the rate limiter must be, from the perspective of traffic traversing that limiter, the same as that of a

single token bucket limiter. This constrains the user, since each user receives a limited share of bandwidth, and the designer, since the system must provide fairness and demand responsiveness similar to a single token bucket. Thus we judge each DRL algorithm based upon its ability to deliver the desired rate limit and its ability to emulate an ideal, centralized token bucket and the fairness such a token bucket would deliver; we measure its fairness using Jain's fairness index [91]. To evaluate each algorithm with respect to these properties, we use emulation, testbed deployment, and analytic study. We use emulation to study each algorithm's behavior at scale, since it is difficult to provide such large-scale repeatability in the wide-area. We use testbed deployment to understand the potential behavior of our algorithms in a real-world deployment. Finally, we use analytic study to better understand the stability of the algorithms in the abstract.

In the context of Decongestion Control, we assume that the protocol must present the same flow-based interface at the transport layer that applications rely upon TCP for. Inherent in this interface—typically presented to applications as sockets—is inter-flow fairness and data loss-free communication. Thus we design Decongestion Control to ensure fairness between flows and provide reliable delivery of data via a socket-based interface. We also incentivize users to cooperatively participate in the use of the protocol. In addition, the protocol must deliver bandwidth at least comparable to that of TCP, and as a result must not cause *wasteful* overload of the network.

To evaluate these properties, we study the protocol in two types of simulation. First, we aim to understand whether the protocol will cause undesirable overload of the network—a decline in aggregate end-to-end goodput for end hosts—and as a result, be unable to provide high bandwidth communication. To this end, we simulate the behavior of the protocol for a variety of large, ISP-like topologies and evaluate the degree to which overload-induced packet loss causes wastage of bandwidth within the network itself. Second, to understand the behavior of the protocol from the perspective of an end-host we simulate it using `ns2` to determine whether it can deliver TCP-like responsiveness to new traffic demands and low jitter.

1.1.2 Principles

Over the years researchers have studied how to secure networks from both selfish and malicious behavior; still, there are few guiding principles—unlike for other areas of network protocol design [12, 52, 54, 177]—that one can follow when designing new protocols. To this end, we highlight several broad principles for cheat-proof networks: conflicting but necessary requirements for networks that both operate smoothly and are free from cheating by user or provider alike. We intentionally avoid defining what constitutes "cheating" as it is almost always a matter of perspective, and present these principles merely as rough guidelines. Each component, each protocol, and each architecture must have system-wide goals that identify desirable and undesirable system operation.

First, there exists a necessary tradeoff between inter-user fairness guarantees and the needs and demands of operators, those with reserved resources, and other privileged users. This tradeoff manifests itself in the context of congestion control and traffic policing—ordinary users should receive their fair share of network bandwidth, but operators should be able to restrict particular flows or traffic classes.

Second, users should be allowed to send traffic as they want, to anyone they want but no user should have a strong incentive to deviate from a system-specified desired behavior. This highlights the need for user freedom and autonomy in the use of a network, but that "rational" parties will not have an incentive to behave in a way that is against their own interests. This tradeoff stems from the need to allow for innovation, but the need to ensure that anarchy is not the result of that innovation. Indeed, many important new technologies, such as peer to peer protocols, have leveraged this freedom.

While these principles are by no means comprehensive, they serve as rough guidelines that help in the design of Distributed Rate Limiting and Decongestion Control.

1.1.3 Scope

In this dissertation we focus our efforts upon only the network and transport layers—the layers currently governed by IP and TCP. However, "cheating"—both broadly defined and as we define it in this dissertation—can exist at any layer of the network. For example, hosts using non-switched Ethernet [86] or IEEE 802.11 [85] can cheat by acquiring their shared communication medium faster or more often than allowed, disobeying the MAC-layer specification. Similarly, users can cheat Web download systems to consume more than such services define as a user's fair share.

While the possibility of cheating at these layers exists today, just as they did for their counterparts of yesteryear, only IP and TCP have remained in use in largely their original manner for decades. Two decades ago, in 1989, neither the Web—a higher layer—nor IEEE 802.11—at a lower layer—existed, yet TCP and IP were in use for the protocols of the day such as SMTP [154], NNTP [94], FTP [155], and Telnet [156]. Since that time, hosts have gone from connecting via modems or shared 10Mbps Ethernet to connecting via numerous wireless protocols and 10Gbps switched Ethernet. We believe that the longevity of use of IP and TCP indicates the fundamental value they deliver as protocols and as abstractions, and therefore believe that efforts to cheat-proof networking protocols must begin with the network and transport layers.

1.2 Contributions

In Chapter 3 we approach cheating in the Internet at the network layer, from the perspective of a network service provider. Specifically, today's cloud-based services integrate globally distributed resources into seamless computing platforms. Accounting for the resource usage of these Internet-scale applications presents a challenging technical problem, especially given that users of these services can consume these distributed resources at many locations simultaneously.

We present the design and implementation of Distributed Rate Limiters, which work

together to enforce a global rate limit across traffic aggregates at multiple sites, enabling the coordinated policing of a cloud-based service's network traffic. Our system enables us to limit the potential cheating of such a service's users, and to do so at the network layer. However, our abstraction not only enforces a global limit, but also ensures that congestion-responsive transport-layer flows behave as if they traversed a single, shared limiter.

We present two primary designs—one general purpose, and one optimized for TCP—that allow service operators to explicitly trade off between communication costs and system accuracy, efficiency, and scalability. Both designs are capable of rate limiting thousands of flows with negligible overhead. We demonstrate that our TCP-centric design is scalable to hundreds of nodes while robust to both loss and communication delay, making it practical for deployment in nationwide service providers. More concretely, we make the following contributions:

- We introduce the network primitive of Distributed Rate Limiting (DRL), which can contain misbehavior by the selfish users of a distributed service.
- We introduce a series of metrics that define what it means for a Distributed Rate Limiting system to effectively limit network traffic.
- We develop several DRL algorithms that expose inherent design tradeoffs in the design of Distributed Rate Limiters.
- We build a DRL system and evaluate these algorithms according to our metrics and present our findings.

In Chapter 4 we approach cheating in the Internet at the transport layer, and consider the case of self-interested end hosts. We propose an alternative approach to Internet congestion control called Decongestion Control, one which presumes that end hosts aim to maximize their own network goodput, no matter the impact on others. In a departure from traditional approaches, end hosts using decongestion control strive to transmit packets faster than the network can deliver them,

leveraging end-to-end erasure coding—to ensure that recieved packets convey useful data—and in-network fairness enforcement—to ensure fairness between mutually uncooperative end hosts.

While loss avoidance has long been central to the Internet architecture, we study whether the benefits of a network architecture that embraces—rather than avoids—widespread packet loss outweigh the potential loss in efficiency. We argue that such an approach may decrease the complexity of routers while increasing stability, robustness, and tolerance for misbehavior. While a number of important design issues remain open, we show that our approach may avoid congestion collapse and may deliver high steady-state goodput for a variety of traffic demands in three different backbone topologies. Specifically, we make the following contributions:

- We introduce a new approach to Internet congestion control that aims to address cheating at end hosts. Our approach is contrary to conventional wisdom on the subject of congestion control: instead of limiting congestion, we embrace packet loss and operate the network at capacity.

- We evaluate the structure of the Internet to determine whether it is appropriate to operate the network under conditions heretofore considered "overload."

- We present the design of a decongestion control protocol that aims to cope with cheating at end hosts and packet loss in the network by erasure coding transport streams.

Chapter 2

Background

The subject of cheat-proof networking has been approached over the years from a variety of angles. In this chapter we examine prior research on congestion control, traffic policing, network incentives, queue management, fairness, Internet architecture, distributed algorithms, and reliable data delivery as it relates to cheat-proofing the network and transport layers. The works we highlight cover a broad swath of networking research, as cheat-proofing requires subtle changes of both protocols and assumptions across the network. Each of the algorithms, protocols, and systems that we overview in this chapter represents an attempt by network designers to reign in the Internet, to maximize the utility of the network for its users, to decrease overhead for network operators, to simplify the network's operation, or to codify exogenous policy desires using network mechanisms. Since the topics covered in this chapter are interrelated and cannot be strictly ordered, we present them iteratively, progressing from the broadest, most basic, or most fundamental work toward more specialized or advanced work.

The early approach to design of Internet protocols still influences the evolution of protocols and systems on the Internet [52]. Clark described how the Internet is built on "rough consensus and running code"—a philosophy that has held true over the decades for everything from TCP SACK to peer-to-peer networking. The most successful protocol or architectural

changes to the network have occurred organically, because end users, software vendors, or network providers found individual benefit. This approach has made fundamental architectural change difficult in the Internet—indeed, while the Internet's architecture has not changed radically in over a decade, the telephone network, which is far more authoritarian, has transparently but fundamentally shifted its architecture in recent decades towards using digital switching and, in some cases, packet-switched networks to carry voice traffic.

Indeed, one of the primary challenges in Internet architecture design is accommodating for incremental deployment. Not only do new protocols, systems, and architectures need to be able to deliver the same service that existing ones do, but they need to provide some incentive for the user and the network provider alike to make the switch. At the same time, they need to be technologically feasible for a large enough population in order for the upgrade to make a difference. Where applicable, we highlight the inroads researchers have made to successfully change the Internet's architecture.

2.1 Internet architecture

The Internet aims to help people and systems communicate; consequently, it must be designed to enable the best and withstand the worst of user behavior. Economists, philosophers, and myrmecologists have known for ages that societies must be built upon rules that work for the individual as well as the whole. Today, the Internet is in need of a new design ethos, one that takes into account all types of user behavior. Before we can examine what new directions are possible, we must examine how the Internet has been designed and refined.

TCP was originally designed to provide reliable, in-order delivery of a data stream across a best-effort, IP packet-switched network. Consequently, TCP must cope with the fact a) that the network does not guarantee end-to-end delivery of data, b) that all data streams are divided into datagrams (the generation of which is the responsibility of end hosts), c) that the end hosts are likely to have numerous flows between them, not just one, and d) that almost all network

applications need to deal with these same challenges. To address these challenges, Cerf and Kahn introduced TCP/IP as a combined protocol in 1974 [48] (that combined the functionality of both the network and transport layers [217]), which a) used acknowledgments to determine whether a packet successfully reached its destination and retransmitted those that did not while assuming no aid from the network [177], b) divided data streams into a sequence of packets that were tagged with a sequence number designating the packet's position in the byte stream, c) used port numbers to allow the operating system to effectively multiplex and demultiplex flows, and d) used a simple abstraction of a reliable data stream which enabled easy integration with existing operating system primitives [135].

TCP's creators were visionary—they designed a protocol that was general enough to be adapted to many unforeseen scenarios, and yet was able to be fixed through modification at end hosts. Over the years, researchers and engineers have identified and addressed numerous problems with TCP. Chief among these was the need for congestion control. A series of "congestion collapses" in the 1980s exposed the problem: there was a need not only for a system that helped *use* the network's resources, but also for a system that helped equitably manage and share consumption of the network's resources—that with freedom also came responsibility.

The solution of the day, and today, was for hosts to participate in a voluntary congestion control protocol that was signaled on top of TCP's existing packet headers and that would govern the rate of transmission by a TCP sender. This has proved to work remarkably well. However, this approach to congestion control itself made many implicit assumptions about the network and its users, including a) that users will always use a "correct" implementation of the protocol, b) that all hosts in the network use the same congestion control protocol, and c) that hosts have numerous users and thus fairness in the network should be defined on the granularity of a flow, not a host or user [169]. Indeed, these assumptions have held true for the most part until recently, but now a) users have an incentive to disobey the protocol to obtain superior performance, b) every operating system vendor uses a different variant of TCP congestion control, and many

applications ignore TCP in its entirety and implement their own transport protocol, and c) very few end hosts today are used simultaneously by multiple people or entities—instead, most machines are used by a single person at a time.

Researchers have not been caught by surprise by the gradual shift in computing and networks that has upturned old assumptions. Indeed, over the years numerous alternative designs have been proposed in an attempt to cope with changing assumptions or cope with vexing problems; we examine many of these protocols and systems in this chapter.

2.2 Traffic policing and provisioning

Since the Internet is a shared network operated and used by parties with disparate interests, network providers require a means for network traffic control. Since their resources are limited, they require means of provisioning—for ensuring that there exist resources in the network to let desirable traffic through and drop undesirable traffic. This traffic control can be either be enforced beforehand, through reservations or admission control, or after the fact, through rate limiting and packet dropping.

2.2.1 Local rate limiters

We begin at a high level and consider the problem of policing user behavior. Indeed, the network layer has a high-level view of packet transmission—sources send packets; destinations receive packets; neither is typically held to account for the bandwidth consumed. Coalitions of hosts, human users with multiple IP addresses, classes of application traffic, and other logical groupings of hosts are irrelevant to the largely stateless, packet-switching core of the Internet.

The simplest manner in which a network provider can control traffic is by selectively forwarding or dropping packets. There are two classic mechanisms—token buckets and leaky buckets—that accomplish this task, and while they are very similar, we review them both due to their popularity. A token bucket represents metaphorical bucket that is used to make decisions

to forward or drop packets based on traffic rate. Upon packet arrival, if a token exists in the virtual bucket, it is removed and the packet is forwarded; if no tokens remain, the packet is dropped. A token bucket allows for packet bursts of a fixed maximum size—the bucket depth—while maintaining a fixed long term rate, governed by the fill rate—the rate at which new tokens are added to the bucket.

A leaky bucket represents a similar concept in which the bucket fills as packets are sent, rather than emptied. When a packet arrives, it is added to the bucket. Any packets that arrive when the bucket is already full are dropped, otherwise they are forwarded. The bucket is also set to "leak" at a fixed rate, thereby ensuring a fixed long-term average.

Over the years, both token buckets and leaky buckets have been used to achieve a variety of traffic shaping goals. For example, Sahu *et al.* analyzed token buckets for achieving differentiated service of TCP flows with packet marking [176]; Liu and Towsley and Rexford *et al.* considered uses of leaky buckets [115, 171].

Several recent studies have attempted to first identify and quantify, and then limit, network traffic. Estan *et al.* have developed algorithms to identify high-bandwidth flows in limited space [67] and Kumar *et al.* have devised an approach to do approximate per-flow counting [109]. After performing estimation, several techniques can be used to perform rate limiting. For example, Mahajan *et al.* showed how to efficiently identify overly aggressive aggregate flows [126]. Active queue management algorithms such as CHOKe [149] and RED-PD [127] can also be used to perform local rate limiting of misbehaving flows. To prevent misbehavior and enforce desirable fairness properties, researchers have taken two basic approaches: explicitly enforcing fairness [26, 27, 59, 76, 150, 189, 196] and limiting aggressive traffic [126, 149, 210]. While neither of these approaches has seen widespread deployment, they form the foundation upon which we build in this dissertation.

2.2.2 Admission control

Admission control is an old idea—one that dates to the era of circuit switching on telephone networks. The idea is simple: admit traffic based on capacity or fairness criteria and deny resources to traffic not admitted. Though the systems we propose in this dissertation are designed for the Internet's best-effort, packet-switched architecture, both DRL and Decongestion Control fit the model of admission control [33]. DRL can be viewed as a real-time admission control system that aims to provide a global notion of fairness and a hard limit on resources. Decongestion is a modern incarnation of the concept of isarithmic networks [57]—where the network is always saturated with packets and where hosts must remove a packet from the network to send a packet—which at its heart is an admission control system.

While there has been considerable work to determine the optimal allocation of bandwidth between end-point pairs in virtual private networks (VPNs), the goal is fundamentally different. In the VPN hose model [108], the challenge is to meet various quality-of-service guarantees [56] by provisioning links in the network to support any traffic distribution that does not exceed the bandwidth guarantees at each end point in the VPN [62]. DRL, on the other hand, is a traffic policing problem where the goal is to emulate the behavior of a single virtual pipe. In contrast, in a VPN, the traffic aggregate is constrained by a single link at the ingress or egress point. Zhang *et al.* introduce an algorithm for admission control that can guarantee network properties along a whole path [214] and for scalable bandwidth management [215].

2.3 Reliable transport and congestion control

While traffic policing and auditing can aid in detecting and preventing misbehavior, today's Internet works because end hosts are cooperative. If we look at large Autonomous Systems (ASes), we find well-connected, over-provisioned backbone networks connected to constrained edge networks. Most routers within these ASes perform FIFO forwarding, leaving them vulnerable to attacks well documented in the literature. Unlike in such attacks, however, users can be

greedy without being directly malicious: they can send packets to legitimate recipients as fast as they want. Since the network lacks mechanisms to prevent such behavior, users can improve their individual throughputs. In aggregate, however, this behavior can lead to Internet-wide congestion collapse [187]. This scenario is a classic tragedy of the commons; individual selfish behavior can drive the system to a globally pessimal state, yet there is no incentive for any user to unilaterally back off.

2.3.1 TCP congestion control

After the congestion collapses of the 1980s, Jacobsen developed a series of congestion control mechanisms for TCP [88]. Concurrently, Ramakrishnan and Jain developed a more theoretically refined "binary feedback" congestion avoidance algorithm, though ultimately this congestion control approach was not widely deployed [167].

To remedy Jacobson's congestion control's inability to recover quickly, researchers developed fast retransmit and fast recovery [195], and later selective acknowledgements (SACK) [132]. Mathis and Mahdavi improved upon SACK without requiring additional protocol changes with the ingenious FACK algorithm [131]. While these modifications of TCP helped cope with packet reordering, TCP was still unable to cope with large-scale reordering; Blanton and Allman worked to improve TCP's ability to handle reordering [31] and Zhang *et al.* designed a new TCP variant, RR-TCP, to make TCP robust to packet reordering [212]. Mogul and Minshall studied the classic TCP Nagle algorithm and provided recommendations on its use in modern networks [139].

Despite TCP's flexibility and its ever-improving performance, researchers have found that TCP faces fundamental performance bottlenecks in numerous environments. For example, Balakrishnan *et al.* found that path asymmetry can severely impact TCP performance [20]. More fundamental is its inability to respond quickly to packet loss in high bandwidth, high delay environments, an issue which we discuss subsequently.

Prior work [179, 187] has also established that congestion-control misbehavior is both

easy and individually profitable, but can have serious effects on the network as a whole. Since no prevention mechanisms exist today, it may only be a matter of time before users take advantage of this vulnerability. For example, the release of a greedy UDP-based Bittorrent client could easily drive the Internet to congestion collapse in a matter of days. Lopez *et al.* analyzed the potential game-theoretic behavior of a fountain-based protocol that uses FEC for transmissions [116, 117]. Their initial investigations show that there is potential for such protocols, which they term "fountain-based protocols". This line of research has also led to more theoretical analyses of TCP congestion control in a competitive setting [6, 96] and the study of the effects of partially cooperative or slow-responding congestion control protocols [22].

2.3.2 Network-centric approaches

One prominent approach to managing network congestion is to apply enforcement in the network. This approach comes with significant history: the "Plain Old Telephone System" (POTS), which is the most similar historical antecedent of the Internet, is designed around the idea of smart network switches that reserve bandwidth for individual flows or sessions. Since the Internet is a best-effort packet-switched network, providing POTS-like reservations and smart sharing is inherently difficult. Specifically, with regard to congestion control, there have been several efforts in the past decade to use router-based information to better guide congestion control decisions by end hosts. Hosts running the XCP and RCP protocols, for example, could receive information about available path bandwidth from routers, and then be able to quickly adjust their send rates [63, 97]. Since it is often the case that determining available path bandwidth is most useful at the beginning of a flow, QuickStart for TCP aims to provide and use that information within the constraints of TCP, easing deployment [71, 178]. An earlier approach—Explicit Congestion Notification (ECN)—ensured that hosts would learn of congestion (at a coarse granularity) [166]; subsequent work has ensured that hosts cooperate with such a mechanism [38, 39, 64].

2.3.3 Endpoint approaches

In addition to the numerous alterations to base-line TCP congestion control (including mechanisms such as fast retransmit, fast recovery, and SACK), researchers have proposed numerous more aggressive end host-based improvements to TCP. Many of these algorithms, such as HighSpeed TCP [70] and Scalable TCP [101], aim to improve TCP's performance in high bandwidth-delay product networks while making as small changes to the protocol itself. Indeed, HighSpeed simply uses alternative window decrease constants when TCP's congestion window is large, thereby ensuring that TCP avoids precipitously backing off upon experiencing a packet loss in a high-bandwidth flow; Scalable TCP alters the AIMD constants for all scenarios, to similar effect. TCP BIC [208] and CUBIC [80] improve upon TCP's ability to converge upon the available bandwidth on its path, while simultaneously allowing for flows to recognize when the available bandwidth has increased significantly.

Along another line of thinking beginning with TCP Vegas [34] and more recently FAST TCP [93, 207], researchers have aimed to use the round-trip delay experienced by a TCP flow's packets to infer when the flow is experiencing queuing, and thus, congestion. Compound TCP aims to leverage information gleaned both from packet drops and from round-trip delay to adjust a flow's congestion window [199].

In addition to TCP-compatible congestion control protocols, researchers have suggested alternatives with more radical changes, such as PCP [11] which paces its packets [3] and performs in-line packet-pair tests to determine the available bandwidth along a path without router assistance. Some protocols have also targeted specific application domains. TFRC [74], for example, ensures that a flow's packets aren't subjected to the bursts of ordinary TCP congestion control while still ensuring that the flow competes fairly for network bandwidth with existing TCP flows; this can prove useful for streaming flows that do not require reliable delivery guarantees [107].

2.3.4 Coded transport

As an alternative to automatic-repeat request (ARQ) reliable transport, researchers have considered using erasure codes and forward-error correction (FEC) to mask losses experienced by transport protocols [114]. There are numerous ways to integrate TCP and forward-FEC [122, 173]; FEC can be placed below, inside, or above TCP—thus, FEC can hide losses from TCP, be used to prevent retransmissions, or be applied to application-layer datagrams. Fountain codes [121, 133] enable senders to send a near-infinite stream of coded packets, and to do so with low encoding and decoding complexity, thereby aiding in the design of non-ARQ based transport [43, 44] for broadcast and bulk data transmission. Others have considered using erasure codes to aid multipath transport protocols [184]. None of these schemes, however, have explored the practicality and ramifications of an entirely FEC-based congestion control on network design.

2.3.5 Fairness and incentives

TCP's notion of fairness—that each flow, irrespective of source or destination, is equal—has become so central to the Internet's architecture that new protocols often must demonstrate their TCP friendliness. However, some have questioned whether TCP friendliness is always an ideal goal [36, 37, 140]. Indeed, even without any new protocols, it is easy to break TCP's notion of fairness: parallel TCP sockets allow a single user or host to capture more resources than their evident fair share [8].

Over the years, researchers have examined whether pricing can suitably alter the incentive structures of the network, and thereby influence the way we think about network fairness. In two fundamental works, Cocchi *et al.* established some foundations on network pricing [55] and Kelly studied charging and rate control [99]. To better understand the consequences of such greedy behavior, researchers have also applied game-theoretic analyses to congestion control [6, 116, 117, 186, 190, 211].

2.4 Distributed coordination and control

The challenges of designing a cheat-proof network architecture are fundamentally distributed—hosts, routers, and other systems must be involved in a series of distributed coordinations. Next we examine several more basic but important distributed algorithms and protocols for coordination, counting, admission control, and group communication. We draw upon the designs and lessons of this area of research in our work on DRL and Decongestion Control.

2.4.1 Distributed counting

Today's routers must maintain counters for many accounting tasks, but to do so is difficult given the high speeds at which packets must be counted and the limited amount of SRAM. To address this issue, Shah *et al.* proposed LCF, a counting algorithm that uses an optimal amount of SRAM [182], but has a fundamental bottleneck that was subsequently eliminated by Ramabhadran *et al.* with the algorithm LR(T) [165]. The direct relevance of these counting algorithms may not be clear at first, but their approach to updating the values of multiple counters in SRAM is the same as recently suggested approaches to updating distributed counters [90]. The general problem of using and efficiently computing aggregates across a distributed set of nodes has been studied in a number of other contexts. These include distributed monitoring [60], triggering [90], counting [185, 206], and data stream querying [19, 129].

A natural extension of local counting algorithms are distributed counters. Viewed abstractly, a distributed counter is a global data structure over n nodes such that any node may perform operations such as incrementing the counter or getting the counter's current value. Wattenhofer *et al.* proved that there is a fundamental bound of $\Omega(\log n / \log \log n)$ on the number of messages required to update the global state of a distributed counter for n operations over n nodes, and also present an algorithm which achieves this bound [206]. In their work, they view the network as a graph of nodes each of which must be able to increment the counter and find its

correct current value. Shavit *et al.* proposed randomized data structures called diffracting trees that similarly implement efficient distributed counters [185]. Distributed counters are analogous to counting networks [15], which were proposed to alleviate coordination overhead in interconnection networks when performing synchronization. Counting networks themselves are related to sorting networks as used in the analysis of parallel sorts [106].

2.4.2 Distributed admission control

Unlike ordinary admission control, most distributed admission control systems consider allocations independent from the edge router at which the request originates; these same approaches are also used to protect against denial-of-service attacks [10]. While it appears that centralized and distributed bandwidth broker architectures provide a mechanism to enforce fair allocations across edge routers, they only consider basing admission on flow types, such as for "large" or "small" flows [215].

Distributed rate limiting can be viewed as a continuous form of distributed admission control. Distributed admission control allows participants to test for and acquire capacity across a set of network paths [92, 215]; each edge router performs flow-admission tests to ensure that no shared hop is over-committed. While our limiters similarly "admit" traffic until the virtual limiter has reached capacity, they do so in an instantaneous, reservation-free fashion. Jamin *et al.* described a measurement-based admission control system [92] while Goel *et al.* proved theoretical bounds for specific approaches to theoretical admission control [78]. To understand the performance impact of end-point vs. router-based admission control, Breslau *et al.* performed simulations under a variety of conditions [35]. Abraham and Kumar presented algorithms for rate control of flows in IntServ networks [1, 53] whereas Kelly *et al.* presented an alternative framework for distributed admission control [100].

2.4.3 Distributed monitoring

Recently, Babcock *et al.* have explored computing Iceberg queries on distributed data streams [19], naturally extending the problem finding "elephants" in local data streams. Simply, they aim to find the k most common elements appearing in distributed streams. Manjhi *et al.* improve upon this result by enabling time-sensitive Iceberg queries on distributed data streams [129].

To find aggregate values over a distributed set of nodes that exceed some threshold, Dilman *et al.* proposed efficient reactive monitoring, in which nodes monitor their local rates and signal a global controller (which polls all nodes) when they exceed some threshold [60]. Their primary goal is to detect such events with minimal inter-node communication. More recently, Jain *et al.* have proposed exploring the same problem from a database-inspired perspective—distributed triggering [90]. A distributed trigger is set to notify of some aggregate constraint being met, in the vein of database triggers. In these contexts, such events are expected to be the uncommon case, as opposed to ours, in which they are the common case; triggers, for example, are proposed as an aid to applications such as distributed debugging. In concept, we view reactive monitoring and distributed triggers as complementary to DRL—our mechanisms can be used, as in our deployment scenarios, when enforcing limits is the common case, and reactive monitoring (or distributed triggers) can be used when it is uncommon, and only notification, not enforcement, is needed.

2.4.4 Distributed traffic management

Zhao *et al.* studied a similar problem of resource sharing in distributed clusters [216]. In their setting, the resources in question are available subject to redirectors, which control user demands and where they are sent. To express resource sharing agreements, they used currencies as done with lottery scheduling. Much of their work focused on enforcement through a single redirector, which made enforcement and redirection decisions for the whole system.

Bhatnagar *et al.* have examined distributed admission control for QoS guarantees in core-stateless networks [30]; their approach used a token that is passed between nodes to provide mutual exclusion in admitting traffic and anticipatory reservation to improve response. Finally, in a more theoretical approach, Reif *et al.* examined a similar problem, in which the goal of the system is to satisfy unknown user demands to distributed resources with minimal response time [170].

2.4.5 Group communication

To enable the distributed algorithms examined in this section, the nodes involved in each system must communicate. As an alternative to traditional distributed computation approaches that use simple communication substrates such as fully-connected meshes, several researchers have proposed using gossip-based approaches, in which information flows semi-randomly through the system from node to node [103, 203]. Gossip protocols lend themselves to theoretical analysis, and often can be proved to converge reliably to a correct state. In doing so, nodes can compute distributed aggregate functions without centralized control. For example, Mehyar *et al.* developed algorithms for asynchronous, distributed averaging [136].

Chapter 3

Distributed Rate Limiting

Today, many distributed systems operate on the scale of the Internet, serving the same set of users at many physically distinct locations. As a result, notions of resource control—particularly, rate limiting—that make sense in centralized systems become difficult to apply. Constraining a self-interested user to a fixed utilization is relatively easy when all decisions and data needed to make decisions are local; when multiple distant nodes must coordinate to do so, it becomes complex and costly.

In the systems of interest, a set of users has access to resources at distributed resource providers. Users may choose to consume resources at any of the providers at any time, with the constraint that, over the whole system, each user does so without exceeding some agreed upon aggregate rate. However, since the providers are distributed, they are immediately aware of only the local rate consumed by each user, not the aggregate of a user's usage over all nodes. Thus, a greedy user has not only the incentive but the means to consume bandwidth—to cheat—far beyond what was possible in a centralized context.

Aside from the functionality and management benefits distributed services afford the end user, today's hosted platforms present significant benefits to the service provider as well. Rather than deploy each component of a multi-tiered application within a particular data center,

so-called "cloud-based services" can transparently leverage widely distributed computing infrastructures. Google's service, for example, reportedly runs on hundreds of thousands of servers distributed around the world [41]. Potential world-wide scale need not be limited to a few large corporations, however. Recent offerings like Amazon's Elastic Compute Cloud (EC2) promise to provide practically infinite resources to anyone willing to pay [9].

A key barrier to moving traditional applications to the cloud is the loss of control of bandwidth, and thus cost. A cheating user has the potential to consume resources in a cloud environment far beyond that of their network uplink. Ultimately service providers only provide services they get paid for. In the cloud-based services model, cost recovery is typically accomplished through metered pricing. For example, Amazon's EC2 charges incrementally per gigabyte of traffic consumed [9]. Experience has shown, however, that ISP customers prefer flat fees to usage-based pricing [146], and in a distributed environment, greedy users are apt to consume far greater resources than they had access to in years past. At a corporate level, IT expenditures are generally managed as fixed-cost overheads, not incremental expenses [81].

A flat-fee model requires the ability for a provider to limit consumption to control costs. Limiting global resource consumption in a distributed environment, however, presents a significant technical challenge. Ideally, resource providers would not require services to specify the resource demands of each distributed component *a priori*; such fine-grained measurement and modeling can be challenging for rapidly evolving services. Instead, they should provide a fixed price for an aggregate, global usage, and allow services to consume resources dynamically across various locations, subject to the specified aggregate limit.

In this chapter, we focus on a specific instance of this problem: controlling the aggregate network bandwidth consumed by a selfish user or cloud-based service, or distributed rate limiting (DRL). Our goal is to allow a set of distributed traffic rate limiters to collaborate to subject a class of network traffic (for example, the traffic of a particular user, group of users, or cloud-based service itself) to a single, aggregate global limit. While traffic policing is common in data centers

and widespread in today's networks, such limiters typically enforce policy independently at each location [147]. For example, a resource provider with 10 hosting centers may wish to limit the total amount of traffic it carries for a particular service to 100 Mbps. Its current options are to either limit the service to 100 Mbps at each hosting center (running the risk that they may all use this limit simultaneously, resulting in 1 Gbps of total traffic), or to limit each center to a fixed portion (i.e., 10 Mbps) which over-constrains the traffic aggregate and is unlikely to allow the user or service to consume its allocated budget unless traffic is perfectly balanced across the cloud.

The key challenge of distributed rate limiting is to allow individual flows to compete dynamically for bandwidth not only with flows traversing the same limiter, but with flows traversing other limiters as well. Thus, flows arriving at different limiters should achieve the same rates as they would if they all were traversing a single, shared rate limiter. Fairness between flows inside a traffic aggregate depends critically on accurate limiter assignments, which in turn depend upon the local packet arrival rates, numbers of flows, and up/down-stream bottleneck capacities. We address this issue by presenting the illusion of passing all of the traffic through a single token-bucket rate limiter, allowing flows to compete against each other for bandwidth in the manner prescribed by the transport protocol(s) in use. For example, TCP flows in a traffic aggregate will share bandwidth in a flow-fair manner [28]. The key technical challenge to providing this abstraction is measuring the demand of the aggregate at each limiter, and apportioning capacity in proportion to that demand. We present three primary contributions:

Rate Limiting Cloud-based Services. We identify a key challenge facing the practical deployment of cloud-based services and identify the chief engineering difficulties: how to effectively balance *accuracy* (how well the system bounds demand to the aggregate rate limit), *responsiveness* (how quickly limiters respond to varying traffic demands), and *communication* between the limiters. A distributed limiter cannot be simultaneously perfectly accurate and responsive; the communication latency between limiters bounds how quickly one limiter can adapt to changing

demand at another.

Distributed Rate Limiter Design. We present the design and implementation of two distributed rate limiting algorithms. First, we consider an approach, *global random drop* (GRD), that approximates the number, but not the precise timing, of packet drops. Second, we observe that applications deployed using Web services will almost exclusively use TCP. By incorporating knowledge about TCP's congestion control behavior, we design another mechanism, *flow proportional share* (FPS), that provides improved scalability.

Evaluation and Methodology. We develop a methodology to evaluate distributed rate limiters under a variety of traffic demands and deployment scenarios using both a local-area testbed and an Internet testbed, PlanetLab. We demonstrate that both GRD and FPS exhibit long-term behavior similar to a centralized limiter for both mixed and homogeneous TCP traffic in low-loss environments. Furthermore, we show that FPS scales well, maintaining near-ideal 50-Mbps rate enforcement and fairness up to 490 limiters with a modest communication budget of 23 Kbps per limiter.

3.1 Problem space

We consider the specific case of class-based rate limiting, where the goal is to restrict the total bandwidth consumed by a given class of traffic. A variety of well-understood mechanisms exist to implement per-class limiting, including class-based queuing, token buckets, and a variety of active queue management (AQM) schemes. Traditionally, however, these schemes have considered limiting the traffic that arrives at a particular network bottleneck. In distributed rate limiting, the goal is to enforce a similar, aggregate per-class rate limit, but do so simultaneously across a set of distinct network locations. Packets from each class of traffic may arrive at any location at any time; since the limiters are topologically distributed, they are only instantaneously aware of the local rate consumed by each traffic class, not the aggregate of a class's usage at all

limiters.

3.2 Applications

Cloud-based services come in varying degrees of complexity; as the constituent services become more numerous and dynamic, resource provisioning becomes more challenging. We observe that the distributed rate limiting problem arises in any service composed of geographically distributed sites. In this section we describe three increasingly mundane applications, each illustrating how DRL empowers service providers to enforce heretofore unrealizable traffic policies, and how it offers a new service model to customers.

Beyond the cloud-based services described earlier, we also discuss DRL in the context of content distribution networks (CDNs), perhaps today's canonical example of distributed resource consumption. And while, we unfortunately do not have access to these large application scenarios, DRL may also be deployed across Internet testbeds, such as PlanetLab, to control resource consumption for particular users or applications.

3.2.1 Limiting cloud-based services

Cloud-based services promise a "utility" computing abstraction in which clients see a unified service and are unaware that the system stitches together independent physical sites to provide cycles, bandwidth, and storage for a uniform purpose. In this context, we are interested in rate-based resources that clients source from a single provider across many sites or hosting centers.

For clouds, distributed rate limiting provides the critical ability for resource providers to control the use of network bandwidth as if it were all sourced from a single site. A provider runs DRL across its sites, setting global limits on the total network usage of particular traffic classes or clients. Providers are no longer required to migrate requests to accommodate static bandwidth limits. Instead, the available bandwidth gravitates towards the sites with the most

demand. Alternatively, clients may deploy DRL to control aggregate usage across providers as they see fit. DRL removes the artificial separation of access metering and geography that results in excess cost for the client and/or wasted resources for service providers.

3.2.2 Content distribution networks

While full-scale cloud-based computing is in its infancy, simple cloud-based services such as content-distribution networks (CDNs) are prevalent today and can benefit from distributed rate limiting. CDNs such as Akamai provide content replication services to third-party Web sites. By serving Web content from numerous geographically diverse locations, CDNs improve the performance, scalability, and reliability of Web sites. In many scenarios, CDN operators may wish to limit resource usage either based on the content served or the requesting identity. Independent rate limiters are insufficient, however, as content can be served from any of numerous mirrors around the world according to fluctuating demand.

Using DRL, a content distribution network can set per-customer limits based upon service-level agreements. The CDN provides service to all sites as before, but simply applies DRL to all out-bound traffic for each site. In this way, the bandwidth consumed by a customer is constrained, as is the budget required to fund it, avoiding the need for CDNs to remove content for customers who cannot pay for their popularity.[1] Alternatively, the CDN can use DRL as a protective mechanism. For instance, the CoDeeN content distribution network was forced to deploy an ad-hoc approach to rate limit nefarious users across proxies [205]. DRL makes it simple to limit the damage on the CDN due to such behavior by rate limiting traffic from an identified set of users. In summary, DRL provides CDNs with a powerful tool for managing access to their clients' content.

[1]For example, Akamai customers are typically not rate limited and billed in arrears for actual aggregate usage, leaving them open to potentially large bills. If demand dramatically exceeds expectation and/or their willingness to pay, manual intervention is required [5].

3.2.3 Internet testbeds

PlanetLab supports the deployment of long-lived service prototypes. Each PlanetLab service runs in a *slice*—essentially a fraction of the entire global testbed consisting of $1/N$ of each machine. Currently PlanetLab provides work-conserving bandwidth limits at each individual site, but the system cannot coordinate bandwidth demands across multiple machines [84].

DRL dynamically apportions bandwidth based upon demand, allowing PlanetLab administrators to set bandwidth limits on a per-slice granularity, independent of which nodes a slice uses. In the context of a single PlanetLab service, the service administrator may limit service to a particular user. In Section 3.5.9 we show that DRL provides effective limits for a PlanetLab service distributed across North America. In addition, while we focus upon network rate limiting in this chapter, we have begun to apply our techniques to control other important rate-based resources such as CPU.

3.2.4 Wireless metropolitan-area networks

In recent years, many cities have begun to deploy wireless Internet service in many public spaces such as outdoor areas and airports. Indeed, many are considering (and a few already have) open city-wide deployments [193]. As with all free, public resources, access must be controlled to ensure fair access to all users. In particular, we can consider independent traffic classes such as Web downloads or peer-to-peer file sharing. Because users may actually be communicating with other hosts inside the access network itself (with peer-to-peer file sharing software, for example), limiting at the gateway to the Internet is unlikely to effectively manage network utilization. Instead, by limiting the total usage by particular aggregates—all peer-to-peer traffic receives a maximum aggregate service rate regardless of location or access point, for example—the city can appropriately apportion capacity across traffic classes.

3.2.5 VPN service

Today, large, multinational businesses often purchase network access from ISPs at each physical location. In such an access model, each location leases a solitary Internet connection of some bandwidth—this capacity is independent of the business's other locations and their usage. Had the company been physically centralized, it likely would have preferred to lease a single line and had its usage metered in aggregate. The artificial separation of the access metering due to geography may result in excess cost and/or wasted resources: if half the company is working and the other half is sleeping, the bandwidth from one location cannot be used at another. This is especially true if the ISP provides a *burstable* charging model, as the increased multiplexing on the single link is likely to decrease the traffic variance.

DRL allows an ISP to logically join such physically dispersed resources, treating them as a single pipe leased to the customer. Thus, the company can purchase fixed network bandwidth world-wide; the ISP can both meet and constrain the demand at that limit. This arrangement is mutually beneficial: the company can use network resources irrespective of geography as its needs change and the ISP can support more customers while provisioning, in aggregate, less bandwidth.

Such a physical separation should not require a logical separation of resources. Instead, with DRL, the company can purchase some fixed network bandwidth world-wide; the ISP can both meet and constrain the demand at that limit. This arrangement is mutually beneficial: the company can use network resources irrespective of geography as its needs change and the ISP can gain customers with this attractive access model while guaranteeing, in aggregate, less bandwidth.

3.2.6 Assumptions and scope

Like centralized rate limiting mechanisms, distributed rate limiting does not provide QoS guarantees. Thus, when customers pay for a given level of service, providers must ensure

the availability of adequate resources for the customer to attain its given limit. In the extreme, a provider may need to provision each limiter with enough capacity to support a service's entire aggregate limit. Nevertheless, we expect many deployments to enjoy considerable benefits from statistical multiplexing. Determining the most effective provisioning strategy is worthy of future study.

The goal of this work is to discover DRL mechanisms that are both *accurate* and *responsive*, allowing the identity to attain the limit just as it would were it crossing a single pipe. Furthermore, we assume that mechanisms are already in place to quickly and easily identify traffic belonging to a particular service [124]. In many cases such facilities, such as simple address or protocol-based classifiers, already exist and can be readily adopted for use in a distributed rate limiter. In others, we can leverage recent work on network capabilities [161, 210] to provide unforgeable means of attribution. Finally, without loss of generality, we discuss our solutions in the context of a single service; multiple services can be limited in the same fashion.

3.3 Algorithms

We are concerned with coordinating a set of topologically distributed limiters to enforce an aggregate traffic limit while retaining the behavior of a centralized limiter. That is, a receiver should not be able to tell whether the rate limit is enforced at one or many locations simultaneously. Specifically, we aim to approximate a centralized token-bucket traffic-policing mechanism. We choose a token bucket as a reference mechanism for a number of reasons: it is simple, reasonably well understood, and commonly deployed in Internet routers. Most importantly, it makes instantaneous decisions about a packet's fate—packets are either forwarded or dropped—and so does not introduce any additional queuing.

We do not assume anything about the distribution of traffic across limiters. Thus, traffic may arrive at any or all of the limiters at any time. We use a peer-to-peer limiter architecture: each limiter is functionally identical and operates independently. The task of a limiter can be

split into three separate subtasks: estimation, communication, and allocation. Every limiter collects periodic measurements of the local traffic arrival rate and disseminates them to the other limiters. Upon receipt of updates from other limiters, each limiter computes its own estimate of the global aggregate arrival rate that it uses to determine how to best service its local demand to enforce the global rate limit.

We do not innovate in estimation or communication; we focus on the appropriate traffic metrics to estimate and the allocation mechanism to deploy at each limiter.

3.3.1 Estimation and communication

We measure traffic demand in terms of bytes per unit time. Each limiter maintains an estimate of both local and global demand. Estimating local arrival rates is well-studied [76, 196]; we employ a strategy that computes the average arrival rate over fixed time intervals and applies a standard exponentially-weighted moving average (EWMA) filter to these rates to smooth out short-term fluctuations. The estimate interval length and EWMA smoothing parameter directly affect the ability of a limiter to quickly track and communicate local rate changes; we determine appropriate settings in Section 3.5.2.

To separate the rate of smoothing from the actual estimate interval, our EWMA weight is defined per unit time as opposed to per estimate interval. Given this parameter and an estimate interval, we can compute the actual EWMA weight as $\sqrt[k]{\alpha}$ where k is the number of estimate intervals per second and α is the per-second EWMA weight.

At the end of each estimate interval, local changes are merged with the current global estimate. In addition, each limiter must disseminate changes in local arrival rate to the other limiters. The simplest form of communication fabric is a broadcast mesh. While fast and robust, a full mesh is also extremely bandwidth-intensive (requiring $O(N^2)$ update messages per estimate interval). Instead, we implement a gossip protocol inspired by Kempe *et al.* [103]. Such "epidemic" protocols have been widely studied for distributed coordination; they require little to

no communication structure, and are robust to link and node failures [58]. Our gossip algorithm computes global sums: at every estimate interval, each limiter sends its current information about the average (which is represented as a sum and a weight—each gossip update contains 20 bytes of message data).

At the end of each estimate interval, limiters select a fixed number of randomly chosen limiters to update; limiters use any received updates to update their global demand estimates. The number of limiters contacted—the gossip branching factor—is a parameter of the system. We communicate updates via a UDP-based protocol that is resilient to loss and reordering; for now we ignore failures in the communication fabric and revisit the issue in Section 3.5.8. Each update packet is 48 bytes, including IP and UDP headers.

Limiters may lack sufficient bandwidth to communicate rate updates. More sophisticated communication fabrics may reduce coordination costs using structured approaches [79]; we defer an investigation to future work. Another option is to locally filter updates or apply hysteresis to reduce communication costs. In this case, a limiter generates a new update only if it differed from its last advertised value by more than a certain factor[2]. Thus the system does not communicate when the arriving demand is stable. We have tested this approach, and have found that this optimization significantly reduces cost with only a small compromise in accuracy.

3.3.2 Allocation

Having addressed estimation and communication mechanisms, we now consider how each limiter can combine local measurements with global estimates to determine an appropriate local limit to enforce. A natural approach is to build a global token bucket (GTB) limiter that emulates the fine-grained behavior of a centralized token bucket. Recall that arriving bytes require tokens to be allowed passage; if there are insufficient tokens, the token bucket drops packets. The rate at which the bucket regenerates tokens dictates the traffic limit. In GTB, each

[2]This is straightforward to apply in the case of a full mesh. In the case of gossip, a limiter must still propagate an update on behalf of other limiters for a certain number of rounds. This could be resolved by associating a TTL with each update.

```
GTB-Update-Token-Bucket(P : Packet)
    elapsed ← (lastupdate − now)
    demand ← Σ_{j≠i}^{n} r_j
    tokencount ← tokencount + (limit · elapsed)
    tokencount ← Min(bucketsize, tokencount)
    tokencount ← tokencount − (demand/2 · elapsed)
    if tokencount ≥ length(P)
        tokencount ← tokencount − length(P)
        forward P
    else
        drop P
    tokencount ← tokencount − (demand/2 · elapsed)
    tokencount ← Max(0, tokencount)
    bytecount ← bytecount + length(P)
GTB-Handle-Packet(P: Packet)
    GTB-Update-Token-Bucket(P)
GTB-Recv-Update-Msg(M : Msg)
    GTB-Update-Token-Bucket(NULL)
    store M
Estimate()
    est ← bytecount / (now − lasttime)
    lasttime ← now
    bytecount ← 0
    r_i ← (r_i · α) + (est·(1 − α))
    propagate r_i
```

Figure 3.1 Pseudocode for GTB. We denote the local limiter as i; r_j is the local knowledge of demand at limiter j, so r_i corresponds to the local demand estimate. α as the EWMA weight. We save tokencount and lasttime between calls to GTB-Update-Token-Bucket.

```
GRD-Handle-Packet(P: Packet)
    demand ← Σ_{i=1}^{n} r_i
    bytecount ← bytecount+Length(P)
    if demand > limit then
        dropprob ← (demand − limit) / demand
        if Rand() < dropprob then
            Drop(P)
            return
    Forward(P)
```

Figure 3.2 Pseudocode for GRD. Each value r_i corresponds to the current estimate of the rate at limiter i.

limiter maintains its own global estimate and uses reported arrival demands at other limiters to estimate the rate of drain of tokens due to competing traffic.

Figure 3.1 shows the pseudo-code of a GTB limiter. Specifically, each limiter's token bucket refreshes tokens at the global rate limit, but removes tokens both when bytes arrive locally and to account for expected arrivals at other limiters. At the end of every estimate interval, each limiter computes its local arrival rate and sends this value to other limiters via the communication fabric. Each limiter sums the most recent values it has received for the other limiters and removes tokens from its own bucket at this "global" rate until a new update arrives.

As shown in Section 3.5.1, however, GTB is highly sensitive to stale observations that continue to remove tokens at an outdated rate, making it impractical to implement at large scale or in lossy networks.

Global random drop

Instead of emulating the precise behavior of a centralized token bucket, we observe that one may instead emulate the higher-order behavior of a central limiter. For example, we can

ensure the rate of drops over some period of time is the same as in the centralized case, as opposed to capturing the burstiness of packet drops—in this way, we emulate the rate enforcement of a token bucket but not its burst limiting. Figure 3.2 presents the pseudocode for a global random drop (GRD) limiter that takes this approach. Like GTB, GRD monitors the aggregate global input demand, but uses it to calculate a packet drop probability. GRD drops packets with a probability proportional to the excess global traffic demand in the previous interval (line 4). Thus, the number of drops is expected to be the same as in a single token bucket; the aggregate forwarding rate should be no greater than the global limit.

GRD somewhat resembles RED queuing in that it increases its drop probability as the input demand exceeds some threshold [75]. Because there are no queues in our limiter, however, GRD requires no tuning parameters of its own (besides the estimator's EWMA and estimate interval length). In contrast to GTB, which attempts to reproduce the packet-level behavior of a centralized limiter, GRD tries to achieve accuracy by reproducing the number of losses over longer periods of time. It does not, however, capture short-term effects. For inherently bursty protocols like TCP, we can improve short-term fairness and responsiveness by exploiting information about the protocol's congestion control behavior.

Flow proportional share

One of the key properties of a centralized token bucket is that it retains inter-flow fairness inherent to transport protocols such as TCP. Given the prevalence of TCP in the Internet, and especially in modern cloud-based services, we design a flow proportional share (FPS) limiter that uses domain-specific knowledge about TCP to emulate a centralized limiter without maintaining detailed packet arrival rates. To do so, we compute the demand at each limiter in terms of the number of TCP flows and set limits at all limiters proportionally.

Each FPS limiter uses a token bucket for rate limiting—thus, each limiter has a *local* rate limit. Unlike GTB, which renews tokens at the global rate, FPS dynamically adjusts its local rate limit in proportion to a set of weights computed every estimate interval. These weights

are based upon the number of live flows at each limiter and serve as a proxy for demand; the weights are then used to enforce max-min fairness between congestion-responsive flows [28]. While this approach should be fair for any congestion-responsive protocols that approximate max-min fairness (such as equation-based congestion control [74]), we describe FPS in the context of TCP.

The primary challenge in FPS is estimating TCP demand. In the previous designs, each rate limiter estimates demand by measuring packets' sizes and the rate at which it receives them; this accurately reflects the byte-level demand of the traffic sources. In contrast, FPS must determine demand in terms of the number of TCP flows present, which is independent of arrival rate. Furthermore, since TCP always attempts to increase its rate, a single flow consuming all of a limiter's rate is nearly indistinguishable from 10 flows doing the same. There is a slight difference between these scenarios: larger flow aggregates have smaller demand oscillations when desynchronized [13]. Since TCP is periodic, we considered distinguishing TCP flow aggregates based upon the component frequencies in the aggregate via the FFT. However, we found that the signal produced by TCP demands is not sufficiently stationary. Nevertheless, we would like that a 10-flow aggregate receive 10 times the weight of a single flow.

Our approach to demand estimation in FPS is shown in Figure 3.3. Flow aggregates are in one of two states. If the aggregate under-utilizes the allotted rate (local limit) at a limiter, then all constituent flows must be *bottlenecked*. In other words, the flows are all constrained elsewhere. On the other hand, if the aggregate either meets or exceeds the local limit, we say that one or more of the constituent flows is *unbottlenecked*—for these flows the limiter is the bottleneck.

We calculate flow weights with the function FPS-Estimate. If flows were max-min fair, then each unbottlenecked flow would receive approximately the same rate. We therefore count a weight of 1 for every unbottlenecked flow at every limiter. Thus, if all flows were unbottlenecked, then the demand at each limiter is directly proportional to its current flow count. Setting the local weight to this number results in max-min fair allocations. We use the computed weight on

```
FPS-Estimate()
1   for each flow f in sample set
2       Estimate(f)
3   localdemand ← r_i
4   if localdemand ≥ locallimit then
5       maxflowrate ← MaxRate(sample set)
6       idealweight ← locallimit / maxflowrate
7   else
8       remoteweights ← Σ_{j≠i}^{n} w_j
9       idealweight ← (localdemand·remoteweights) / (limit−localdemand)
10  locallimit ← (idealweight·limit) / (remoteweights+idealweight)
11  Propagate(idealweight)

FPS-Handle-Packet(P: Packet)
1   if Rand() < resampleprob then
2       add Flow(P) to sample set
3   Token-Bucket-Limit(P)
```

Figure 3.3 Pseudocode for FPS. w_i corresponds to the weight at each limiter i that represents the normalized flow count (as opposed to rates r_i as in GRD).

line 10 of FPS-Estimate to proportionally set the local rate limit.

A seemingly natural approach to weight computation is to count TCP flows at each limiter. However, flow counting fails to account for the demands of TCP flows that are bottlenecked: 10 bottlenecked flows that share a modem do not exert the same demands upon a limiter as a single flow on an OC-3. Thus, FPS must compute the equivalent number of unbottlenecked TCP flows that an aggregate demand represents. Our primary insight is that we can use TCP itself to estimate demand: in an aggregate of TCP flows, each flow will eventually converge to

its fair-share transmission rate. This approach leads to the first of two operating regimes:

Local arrival rate $\geq$ local rate limit. When there is at least one unbottlenecked flow at the limiter, the aggregate input rate is equal to (or slightly more than) the local rate limit. In this case, we compute the weight by dividing the local rate limit by the sending rate of an unbottlenecked flow, as shown on lines 5 and 6 of FPS-Estimate. Intuitively, this allows us to use a TCP flow's knowledge of congestion to determine the amount of competing demand. In particular, if all the flows at the provider are unbottlenecked, this yields a flow count without actual flow counting.

Thus, to compute the weight, a limiter must estimate an unbottlenecked flow rate. We can avoid per-flow state by sampling packets at a limiter and maintaining byte counters for a constant-size flow set. We assume that the flow with the maximum sending rate is unbottlenecked. However, it is possible that our sample set will contain only bottlenecked flows. Thus, we continuously resample and discard small flows from our set, thereby ensuring that the sample set contains an unbottlenecked flow. We have empirically determined settings for re-sampling in our implementation; the parameters seem to work irrespective of flow properties. We leave a further exploration of this parameter to future work.

It is likely that we will select an unbottlenecked flow in the long run for two reasons. First, since we uniformly sample packets, an unbottlenecked flow is more likely to be picked than a bottlenecked flow. Second, a sample set that contains only bottlenecked flows results in the weight being overestimated, which increases the local rate limit, causes unbottlenecked flows to grow, and makes them more likely to be chosen subsequently.

To account for bottlenecked flows, FPS implicitly normalizes the weight by scaling down the contribution of such flows proportional to their sending rates. A bottlenecked flow only contributes a fraction equal to its sending rate divided by that of an unbottlenecked flow. For example, if a bottlenecked flow sends at 10 Kbps, and the fair share of an unbottlenecked flow is 20 Kbps, the bottlenecked flow counts for half the weight of an unbottlenecked flow.

Local arrival rate < local rate limit. When all flows at the limiter are bottlenecked, there is no unbottlenecked flow whose rate can be used to compute the weight. Since the flow aggregate is unable to use all the rate available at the limiter, we compute a weight that, based on current information, sets the local rate limit to be equal to the local demand (line 9 of FPS-Estimate).

A limiter may oscillate between the two regimes: entering the second typically returns the system to the first, since the aggregate may become unbottlenecked due to the change in local rate limit. As a result, the local rate limit is increased during the next allocation, and the cycle repeats. We note that this oscillation is necessary to allow bottlenecked flows to become unbottlenecked should additional capacity become available elsewhere in the network; like the estimator, we apply an EWMA to smooth this oscillation. (In the next section we show that FPS is stable—given stable input demands, FPS remains at the correct allocation of weights among limiters once it arrives in that state. It remains an open question, however, whether FPS converges under all conditions, and if so, how quickly.)

Finally, TCP's slow start behavior complicates demand estimation. Consider the arrival of a flow at a limiter that has a current rate limit of zero. Without buffering, the flow's SYN will be lost and the flow cannot establish its demand. To combat this, one natural approach is to allot a percentage of the aggregate rate limit as slack at each node to allow for the start of new flows. Unfortunately, this wastes resources, as nodes that are completely unused still consume rate that could be used elsewhere.

Thus, we allow bursting of the token bucket when the local rate limit is zero to allow a TCP flow in slow start to send a few packets before losses occur. When the allocator detects nonzero input demand, it treats the demand as a bottlenecked flow for the first estimate interval. As a result, FPS allocates rate to the flow equivalent to its instantaneous rate during the beginning of slow start, thus allowing it to continue to grow.

3.3.3 Stability

Here we show that FPS correctly stabilizes to the "correct" allocations at all limiters in the presence of both unbottlenecked and bottlenecked flows. Our goal is not to exhaustively prove convergence properties about the algorithm, but simply determine whether it will remain stable—when the algorithm produces the correct allocation between (idealized) TCP flows, it will continue to correctly allocate rate.

First, we present a model of TCP estimation over n limiters. Let $a_1, a_2, \ldots, a_n$ be the number of unbottlenecked flows at limiters 1 to n respectively. Similarly, let $B_1, B_2, \ldots, B_n$ be the local bottlenecked flow rates (which may include multiple flows). At the ith limiter, there exists a local rate limit, l_i. These limits are subject to the constraint that $l_1 + l_2 + \cdots + l_n = L$, where L is the global rate limit. Let $U = L - \sum_i B_i$ represent the total amount of rate available for unbottlenecked flows. Let $A = \sum_i a_i$ represent the total number of unbottlenecked flows across all limiters. Given these values, a TCP estimator outputs a tuple of weights $(w_1, w_2, \ldots, w_n)$ that are used by FPS to assign rate limits at all limiters. Suppose we are given perfect global knowledge and are tasked to compute the correct allocations at all limiters. The allocation would be

$$I = (U \cdot \frac{a_1}{A} + B_1, U \cdot \frac{a_2}{A} + B_2, \ldots, U \cdot \frac{a_n}{A} + B_n).$$

Note that these weights are also equal to the actual rate limits assigned at each node. This corresponds to an allocation which would result for each limiter's flow aggregate had all flows (globally) been forced through a single pipe of capacity L.

FPS first estimates the rate of a single unbottlenecked flow at each limiter. Once stabilized, such a flow at limiter number i will receive a rate f (where l_i is the *current* rate limit at limiter i):

$$f = \frac{l_i - B_i}{a_i}.$$

Given these flow rates, FPS will compute a new weight w_i at each limiter:

$$w_i = \frac{l_i \cdot a_i}{l_i - B_i}.$$

Once FPS arrives at the ideal allocation, it will remain at the ideal allocation in the absence of any demand changes. That is, suppose $(l_1, \ldots, l_n) = (I_1, \ldots, I_n)$. We claim that the newly computed weights $(w_1, \ldots, w_n)$ result in the same allocation; equivalently,

$$\frac{w_1}{w_1 + \cdots + w_n} = \frac{I_1}{I_1 + \cdots + I_n}.$$

The weights computed given this starting state are, for each i,

$$w_i = \frac{(U \cdot \frac{a_i}{A} + B_i) \cdot a_i}{(U \cdot \frac{a_i}{A} + B_i) - B_i}.$$

Thus, considering the allocation at limiter 1,

$$\frac{w_1}{w_1 + \cdots + w_n} = \frac{\frac{Ua_1 + AB_1}{U}}{\frac{Ua_1 + AB_1}{U} + \cdots + \frac{Ua_n + AB_n}{U}},$$

which is equal to

$$\frac{Ua_1 + AB_1}{Ua_1 + AB_1 + \cdots + Ua_n + AB_n} = \frac{I_1}{I_1 + \cdots + I_n},$$

the ideal allocation fraction for limiter 1. The allocations at other limiters are analogous.

3.4 Methodology

Given the problem of distributed rate limiting, how might we conclude that a particular limiter design is good and for what parameters? To this end, we appeal to centralized behavior—for the representative metrics we detail below, we aim to determine how close the DRL system's behavior is to that of a centralized token bucket limiter.

Traffic policing mechanisms can affect packets and flows on several time scales; particularly, we can aim to emulate packet-level behavior or flow-level behavior. However, packet-level behavior is non-intuitive, since applications typically operate at the flow level. Even in a single limiter, any one measure of packet-level behavior fluctuates due to randomness in the physical system, though transport-layer flows may achieve the same relative fairness and throughput. This implies a weaker, but tractable goal of functionally equivalent behavior. To this end, we measure limiter performance using aggregate metrics over real transport-layer protocols.

3.4.1 Metrics

We study three metrics to determine the fidelity of limiter designs: utilization, flow fairness, and responsiveness. The basic goal of a distributed rate limiter is to hold aggregate throughput across all limiters below a specified global limit. To establish fidelity we need to consider utilization over different time scales. Achievable throughput in the centralized case depends critically on the traffic mix. Different flow arrivals, durations, round trip times, and protocols imply that aggregate throughput will vary on many time scales. For example, TCP's burstiness causes its instantaneous throughput over small time scales to vary greatly. A limiter's long-term behavior may yield equivalent aggregate throughput, but may burst on short time scales. Note that, since our limiters do not queue packets, some short-term excess is unavoidable to maintain long-term throughput. Particularly, we aim to achieve fairness equal to or better than that of a centralized token bucket limiter.

Fairness describes the distribution of rate across flows. We employ Jain's fairness index to quantify the fairness across a flow set [91]. The index considers k flows where the throughput of flow i is x_i. The fairness index f is between 0 and 1, where 1 is completely fair (all flows share bandwidth equally):

$$f = \frac{\left(\sum_{i=1}^{k} x_i\right)^2}{k\left(\sum_{i=1}^{k} x_i^2\right)}$$

We must be careful when using this metric to ascertain flow-level fidelity. Consider a set of identical TCP flows traversing a single limiter. Between runs, the fairness index will show considerable variation; establishing the flow-level behavior for one or more limiters requires us to measure the distribution of the index across multiple experiments. Additional care must be taken when measuring Jain's index across multiple limiters. Though the index approaches 1 as flows receive their fair share, skewed throughput distributions can yield seemingly high indices. For example, consider 10 flows where 9 achieve similar throughput while 1 gets nothing; this results in the seemingly high fairness index of 0.9. If we consider the distribution of flows across limiters—the 9 flows go through one limiter and the 1 flow through another—the fairness index

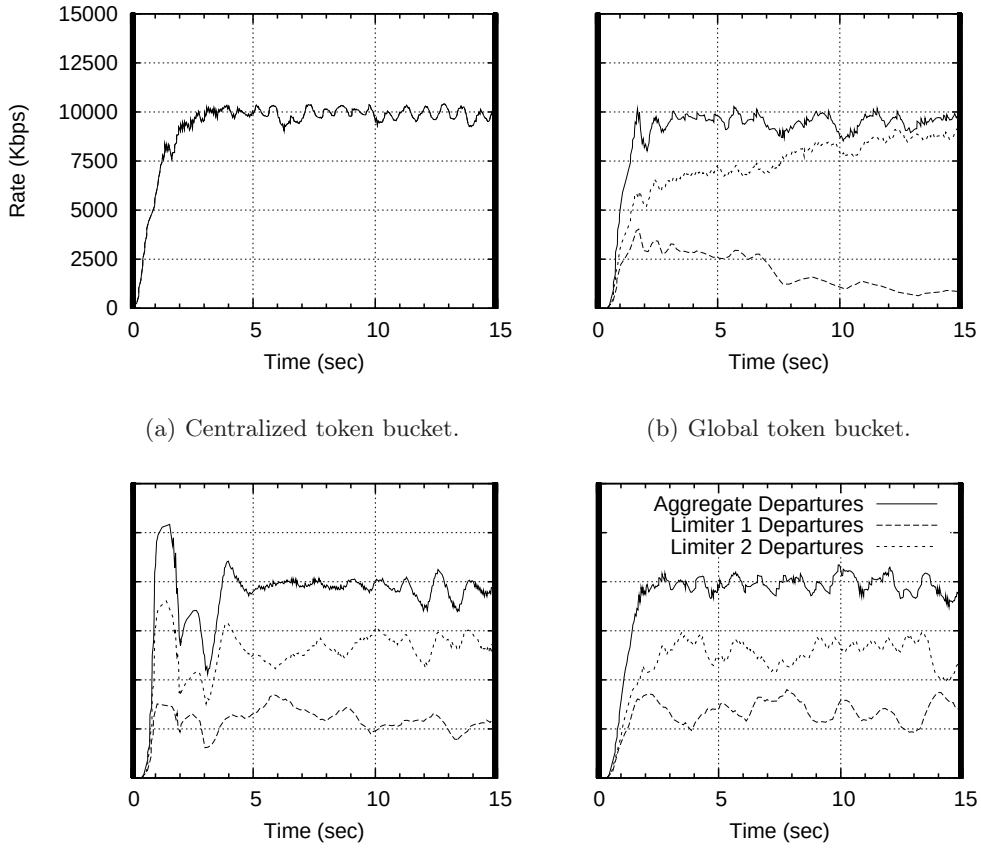
does not capture the poor behavior of the algorithm. Nevertheless, such a metric is necessary to help establish the flow-level behavior of our limiters, and therefore we use it as a standard measure of fairness with the above caveat. We point out discrepancies when they arise.

3.4.2 Implementation

To perform rate limiting on real flows without proxying, we use user-space queuing in `iptables` on Linux to capture full IP packets and pass them to the designated rate limiter without allowing them to proceed through kernel packet processing. Each rate limiter either drops the packet or forwards it on to the destination through a raw socket. We use similar, but more restricted functionality for VNET raw sockets in PlanetLab to capture and transmit full IP packets. Rate limiters communicate with each other via UDP. Each gossip message sent over the communication fabric contains a sequence number in addition to rate updates; the receiving limiter uses the sequence number to determine if an update is lost, and if so, compensates by scaling the value and weight of the newest update by the number of lost packets. Note that all of our experiments rate limit traffic in one direction; limiters forward returning TCP ACKs irrespective of any rate limits.

3.4.3 Evaluation framework

We evaluate our limiters primarily on a local-area emulation testbed using ModelNet [200], which we use only to emulate link latencies. A ModelNet emulation tests real, deployable prototypes over unmodified, commodity operating systems and network stacks, while providing a level of repeatability unachievable in an Internet experiment. Running our experiments in a controlled environment helps us gain intuition, ensures that transient network congestion, failures, and unknown intermediate bottleneck links do not confuse our results, and allows direct comparison across experiments. We run the rate limiters, traffic sources, and traffic sinks on separate endpoints in our ModelNet network topology. All source, sink, and rate limiter ma-



(a) Centralized token bucket.

(b) Global token bucket.

(c) Global random drop.

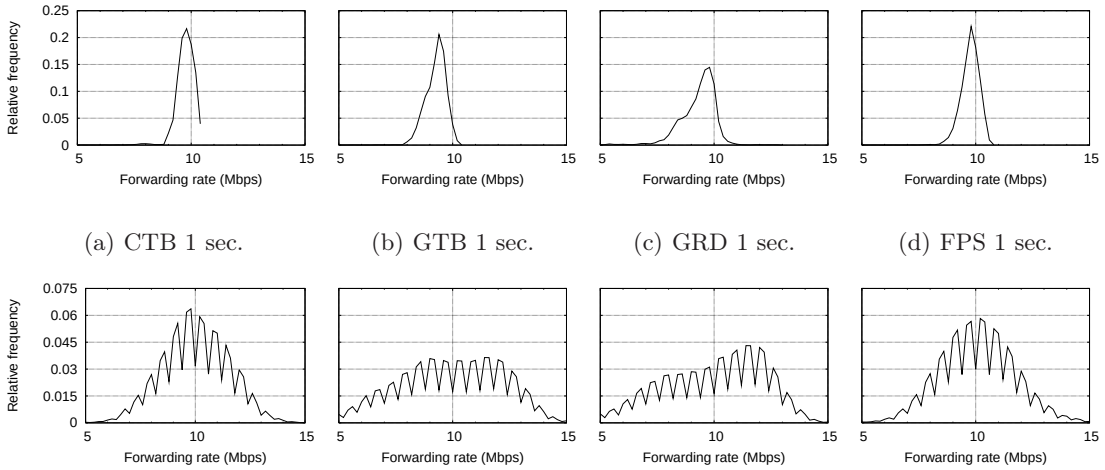
(d) Flow proportional share.

Figure 3.4 Time series of forwarding rate for a centralized limiter and our three limiting algorithms in the baseline experiment—3 TCP flows traverse limiter 1 and 7 TCP flows traverse limiter 2.

chines run Linux 2.6.9. TCP sources use New Reno with SACK enabled. We use a simple mesh topology to connect limiters and route each source and sink pair through a single limiter. The virtual topology connects all nodes using 100-Mbps links.

3.5 Experiments

Our evaluation has two goals. The first is to establish the ability of our algorithms to reproduce the behavior of a single limiter in meeting the global limit and delivering flow-level fairness. These experiments use only 2 limiters and a set of homogeneous TCP flows. Next we relax this idealized workload to establish fidelity in more realistic settings. These experiments



(a) CTB 1 sec. (b) GTB 1 sec. (c) GRD 1 sec. (d) FPS 1 sec.

(e) CTB 0.1 sec. (f) GTB 0.1 sec. (g) GRD 0.1 sec. (h) FPS 0.1 sec.

Figure 3.5 Delivered forwarding rate for the aggregate at different time scales—each row represents one run of the baseline experiment across two limiters with the "instantaneous" forwarding rate computed over the stated time period.

help achieve our second goal: to determine the effective operating regimes for each design. For each system we consider responsiveness, performance across various traffic compositions, and scaling, and vary the distribution of flows across limiters, flow start times, protocol mix, and traffic characteristics. Finally, as a proof of concept, we deploy our limiters across PlanetLab to control a mock-up of a simple cloud-based service.

3.5.1 Baseline

The baseline experiment consists of two limiters configured to enforce a 10-Mbps global limit. We load the limiters with 10 unbottlenecked TCP flows; 3 flows arrive at one limiter while 7 arrive at the other. We choose a 3-to-7 flow skew to avoid scenarios that would result in apparent fairness even if the algorithm fails (which would occur in the case of a 1-to-9 flow skew). The reference point is a centralized token-bucket limiter (CTB) servicing all 10 flows. We fix flow and inter-limiter round trip times (RTTs) to 40 ms, and token bucket depth to 75,000 bytes—slightly greater than the bandwidth-delay product, and, for now, use a loss-free communication fabric. Each experiment lasts 60 seconds (enough time for TCP to stabilize), the estimate interval is 50

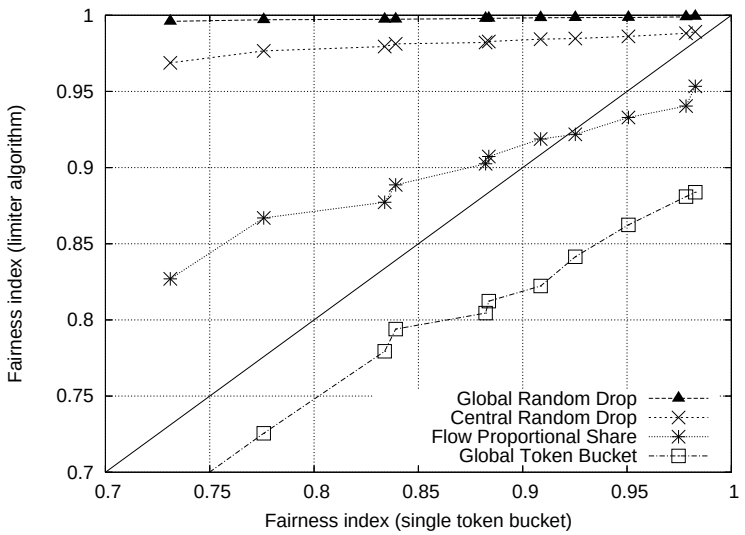


Figure 3.6 Quantile-quantile plots of a single token bucket vs. distributed limiter implementations. For each point $(x, y), x$ represents a quantile value for fairness with a single token bucket and y represents the same quantile value for fairness for the limiter algorithm.

ms, and the 1-second EWMA parameter is 0.1; we consider alternative values in the next section.

Figure 3.4 plots the packet forwarding rate at each limiter as well as the achieved throughput of the flow aggregate. In all cases, the aggregate utilization is approximately 10 Mbps. We look at smaller time scales to determine the extent to which the limit is enforced. Figure 3.5 shows histograms of delivered "instantaneous" forwarding rates computed over two different time periods, thus showing whether a limiter is bursty or consistent in its limiting. All designs deliver the global limit over 1-second intervals; both GTB and GRD, however, are bursty in the short term. By contrast, FPS closely matches the rates of CTB at both time scales. We believe this is because FPS uses a token bucket to enforce local limits. It appears that when enforcing the same aggregate limit, the forwarding rate of multiple token buckets approximates that of a single token bucket even at short time scales.

Returning to Figure 3.4, the aggregate forwarding rate should be apportioned between limiters in about a 3-to-7 split. GTB clearly fails to deliver in this regard, but both GRD and FPS appear approximately correct upon visual inspection. We use Jain's fairness index to quantify the fairness of the allocation. For each run of an experiment, we compute one fairness value across

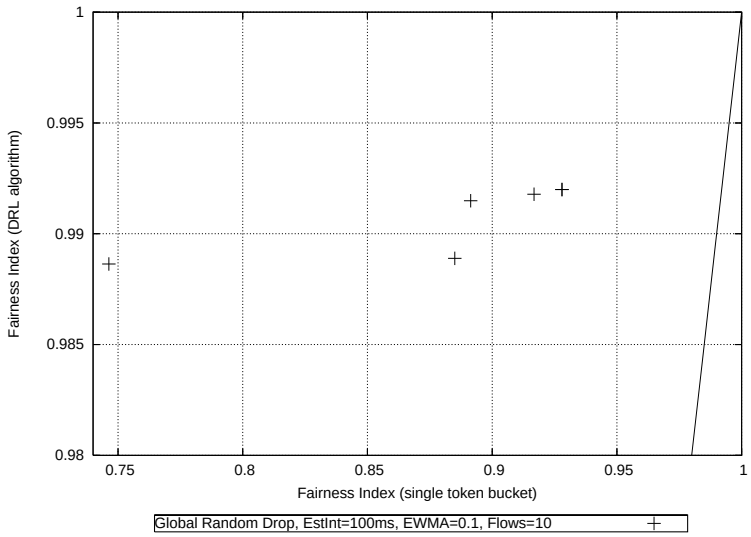


Figure 3.7 Quantile-quantile plots of a single token bucket vs. a single random drop limiter.

all flows, irrespective of the limiter at which they arrive. Repeating this experiment 10 times yields a distribution of fairness values. We use quantile-quantile plots to compare the fairness distribution of each limiter to the centralized token bucket (CTB). Recall that an important benchmark of our designs is their ability to reproduce a distribution of flow fairness at least as good as that of CTB. If they do, their points will closely follow the $x = y$ line; points below the line are less fair, indicating poor limiter behavior and points above the line indicate that the rate limiting algorithm produced better fairness than CTB.

Figure 3.6 compares distributions for all algorithms in our baseline experiment. GTB has fairness values around 0.7, which corresponds to the 7-flow aggregate unfairly dominating the 3-flow aggregate. This behavior is clearly visible in Figure Figure 3.4(b), where the 7-flow limiter receives almost all the bandwidth. GRD and FPS, on the other hand, exhibit distributions that are at or above that of CTB. GRD, in fact, has a fairness index close to 1.0—much better than CTB. We verify this counter-intuitive result by comparing the performance of CTB with that of a single GRD limiter (labeled "Central Random Drop" in the figure) in Figure 3.7. It is not surprising, then, that FPS is less fair than GRD, since it uses a token bucket at each limiter to enforce the local rate limit. In future work, we hope to experiment with a local GRD-like random

drop mechanism instead of a token bucket in FPS; this will improve the fairness of FPS in many scenarios.

Additionally, with homogeneous flows across a wide range of parameters—estimate intervals from 10 ms to 500 ms and EWMA from 0 to 0.75—we find that GTB and GRD are sensitive to estimate intervals, as they attempt to track packet-level behaviors. In general, GTB exhibits poor fairness for almost all choices of EWMA and estimate interval, and performs well only when the estimate interval is small and the EWMA is set to 0 (no filter). We conjecture that GTB needs to sample the short-term behavior of TCP in congestion avoidance, since considering solely aggregate demand over long time intervals fails to capture the increased aggressiveness of a larger flow aggregate. We verified that GTB provides better fairness if we lengthen TCP's periodic behavior by growing its RTT. Since all results show that GTB fails with anything but the smallest estimate interval, we do not consider it further.

GRD is sensitive to the estimate interval, but in terms of short-term utilization, not flow fairness, since it maintains the same drop probability until it receives new updates. Thus, it occasionally drops at a higher-than-desired rate, causing congestion-responsive flows to back off significantly. While its long-term fairness remains high even for 500-ms estimate intervals, short-term utilization becomes exceedingly poor. By contrast, for homogeneous flows, FPS appears insensitive to the estimate interval, since flow-level demand is constant. Both GRD and FPS require an EWMA to smooth input demand to avoid over-reacting to short-term burstiness. Though neither are particularly sensitive to EWMA, we empirically determined that a reasonable setting of the 1-second EWMA is 0.1. We use this value unless otherwise noted.

3.5.2 EWMA sweep

The baseline experiment uses only one point in the space of possible estimation intervals and EWMA values. At this point GTB exhibits poor fairness and GRD, though fair, is bursty. To understand this behavior we have explored changing the estimate interval and EWMA. 3.5.2

Table 3.1 Parameter space evaluation over estimate intervals T and 1-second EWMA α. Each entry represents the median fairness index over 5 runs of the experiment. The median for CTB was 0.904.

T	α	GTB	GRD	FPS
	0.0	0.922	0.912	0.838
10	0.01	0.653	0.997	0.881
	0.1	0.712	0.996	0.856
	0.0	0.697	0.996	0.875
100	0.01	0.621	0.997	0.838
	0.1	0.718	0.997	0.873
	0.0	0.670	0.969	0.843
500	0.01	0.688	0.981	0.879
	0.1	0.720	0.993	0.824

highlights the results of our parameter sweep. Given the above baseline experiment, we examine how the estimate interval and EWMA affect the behavior of GTB, GRD, and FPS. We are particularly interested in the fairness, which we consider in Figure 3.8, Figure 3.9, and Figure 3.10. Next we aim to gain an intuitive, analytical understanding of the behavior of sampling and the EWMA.

How fast must we sample to capture the behavior of TCP's congestion avoidance such that it can be reproduced faithfully? In an effort to gain some intuition about this question, we can think of TCP's AIMD as a periodic signal. For example, consider a single flow of some fixed RTT sec bottlenecked by a token bucket of rate B bytes/sec. Assuming the token bucket is of sufficient depth ($B \times RTT$ for a single flow or proportional to $\sqrt{n}$ for n flows [13]) to allow TCP to fully utilize the path, its maximum window will be of size $B \cdot RTT$ bytes. After experiencing a congestion loss, it will halve its window to $B \cdot RTT/2$, pause for $RTT/2$ seconds for outstanding ACKs to return, and resume transmission of packets. Since TCP will increase its window by 1 packet per RTT, for some maximum segment size MSS, we can determine the period P' of TCP's AIMD sawtooth:

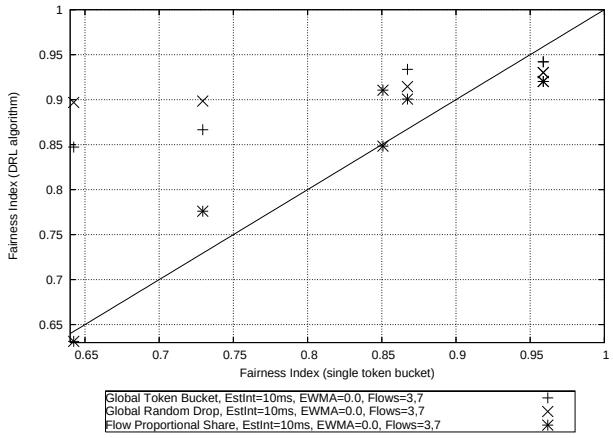
$$\frac{MSS}{RTT} = \frac{B \cdot RTT/2}{P'}.$$

Solving for P', and adding the appropriate pause time, we obtain the period P of a single TCP

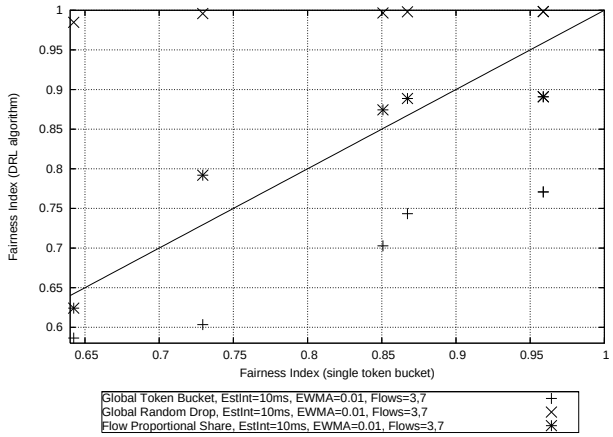
flow:

$$P = \frac{B \cdot RTT^2}{2 \cdot MSS} + \frac{RTT}{2}.$$

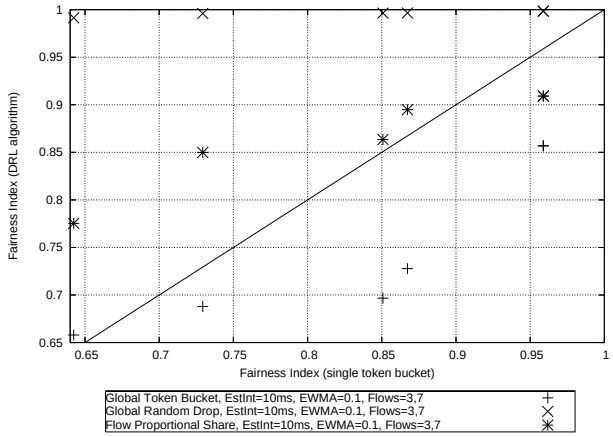
From this simple derivation, we now can upper-bound the frequency of a TCP sawtooth of n flows, to be $1/(n \cdot P)$. Thus, by Nyquist's theorem, we must sample at a frequency of at least $2/(n \cdot P)$ to be able to reproduce the signal. We note that since it is unlikely that flows will remain synchronized, it is unlikely that the signal will be stationary on long time scales [151]; particularly, while the highest-frequency component of the signal may be less than our estimate above in most cases, there are also cases that will result in even higher frequencies than we estimate (the case of a flow in an aggregate of n flows that has a congestion window of at most 3 packets that hits its ceiling every RTT due to contention with other flows). Altman *et al.* have determined a formula to compute the aggregate throughput for an arbitrary number of TCP flows, though this is insufficient to compute the effective demand exerted by such flows [8].



(a) 0.0 EWMA.

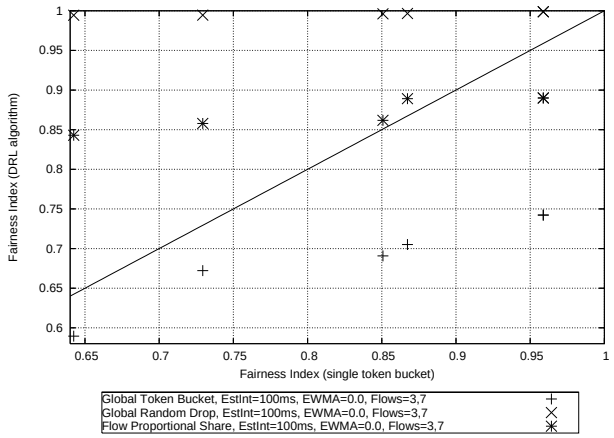


(b) 0.01 EWMA.

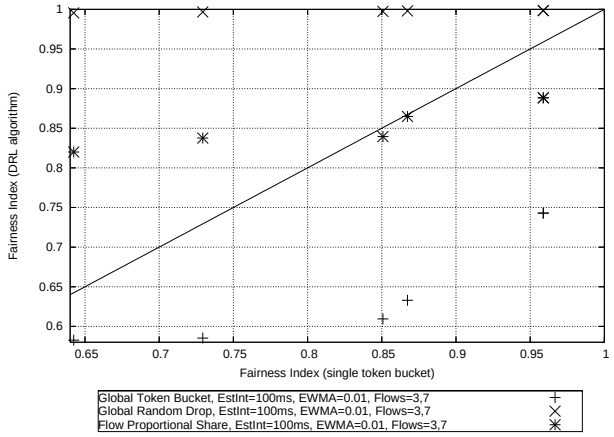


(c) 0.1 EWMA.

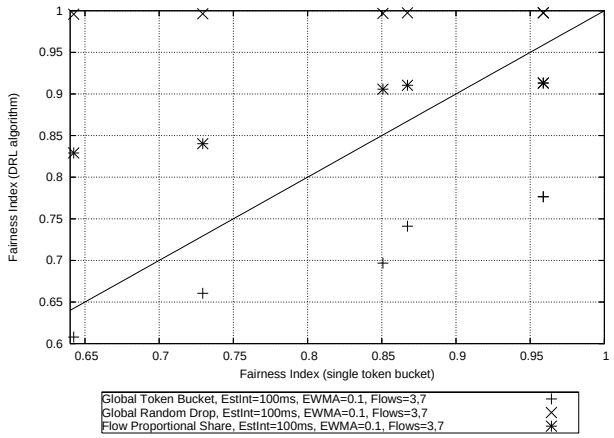
Figure 3.8 Quantile-quantile plots of a single token bucket vs. various DRL implementations with a 10ms estimate interval.



(a) 0.0 EWMA.

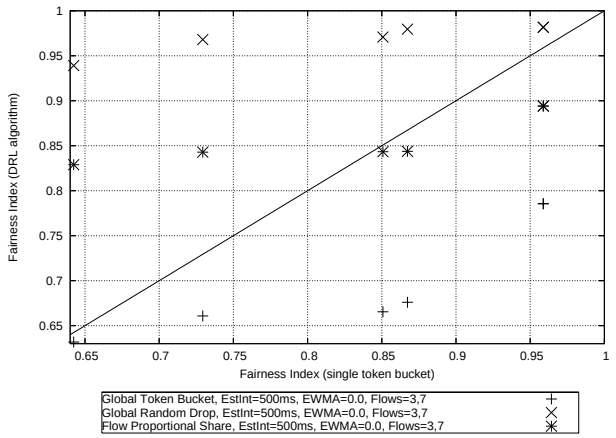


(b) 0.01 EWMA.

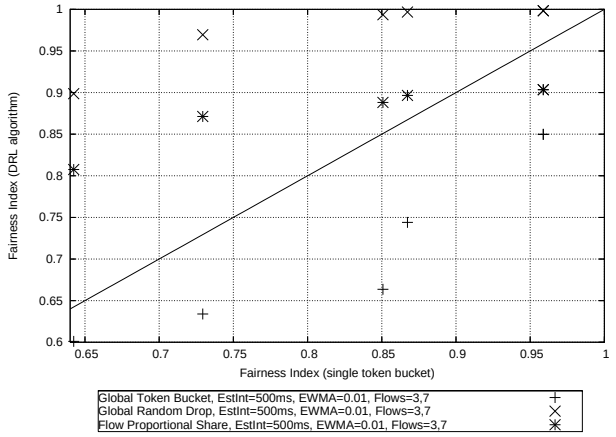


(c) 0.1 EWMA.

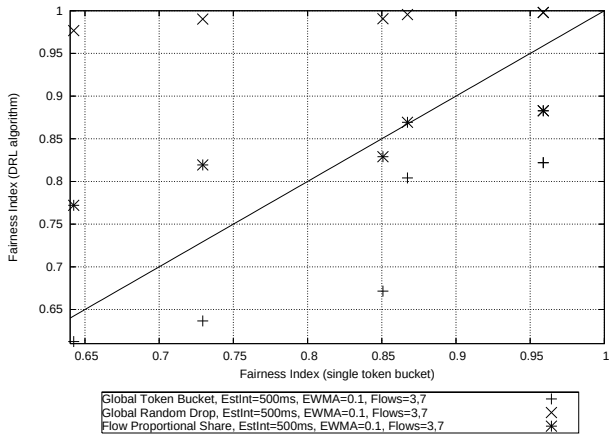
Figure 3.9 Quantile-quantile plots of a single token bucket vs. various DRL implementations with a 100ms estimate interval.



(a) 0.0 EWMA.



(b) 0.01 EWMA.

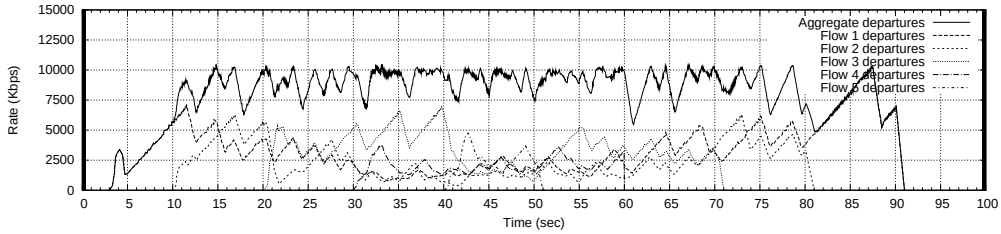


(c) 0.1 EWMA.

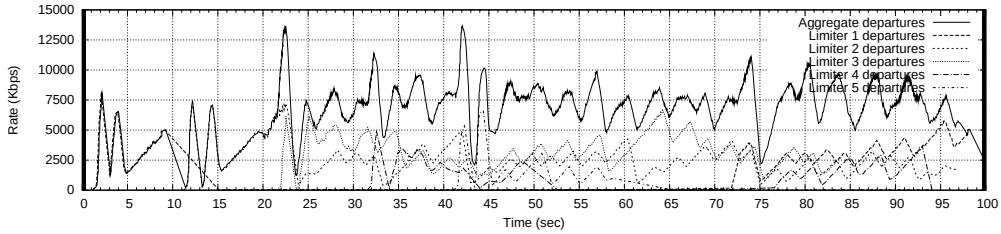
Figure 3.10 Quantile-quantile plots of a single token bucket vs. various DRL implementations with a 500ms estimate interval.

3.5.3 Flow dynamics

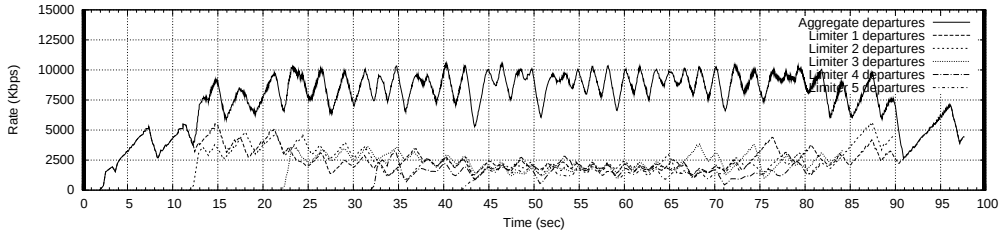
We now investigate responsiveness (time to convergence and stability) by observing the system as flows arrive and depart. We sequentially add flows to a system of 5 limiters and observe the convergence to fair share of each flow. Figure 4.7(a) shows the reference time-series behavior for a centralized token bucket. Note that even through a single token bucket, the system is not completely fair or stable as flows arrive or depart due to TCP's burstiness. With a 500-ms estimate interval, GRD (Figure 4.7(b)) fails to capture the behavior of the central token bucket. Only with an order-of-magnitude smaller estimate interval (Figure 4.7(c)) is GRD able to approximate the central token bucket, albeit with increased fairness. FPS (Figure 4.7(d,e)), on the other hand, exhibits the least amount of variation in forwarded rate even with a 500-ms estimate interval, since flow-level demand is sufficiently constant over half-second intervals. This experiment illustrates that the behavior GRD must observe occurs at a packet-level time scale: large estimate intervals cause GRD to lose track of the global demand, resulting in chaotic packet drops. FPS, on the other hand, only requires updates as flows arrive, depart, or change their behavior.



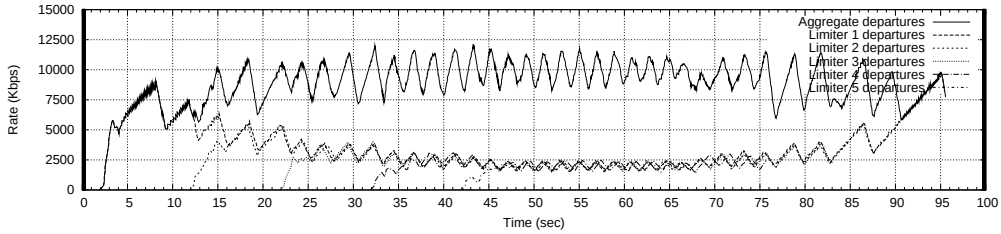
(a) Central token bucket



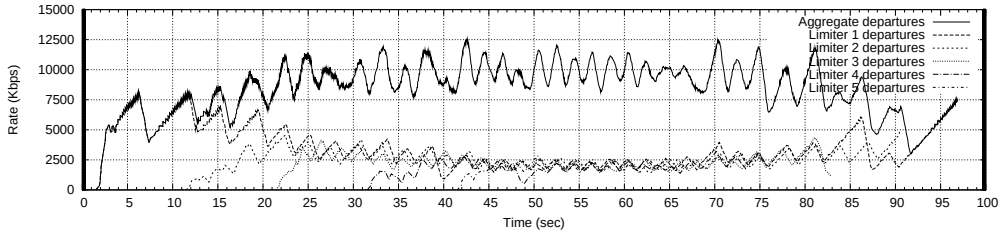
(b) Global random drop with 500-ms estimate interval



(c) Global random drop with 50-ms estimate interval



(d) Flow proportional share 50 ms.



(e) Flow proportional share 500-ms estimate interval

Figure 3.11 Time series of forwarding rate for a flow join experiment. Every 10 seconds, a flow joins at an unused limiter.

Table 3.2 Goodput and delivered rates (Kbps), and fairness for bulk flows over 10 runs of the Web flow experiment. We use mean values for goodput across experiments and use the harmonic mean of rates (Kbps) delivered to Web flows of size (in bytes) within the specified ranges.

	CTB	GRD	FPS
Goodput (bulk mean)	6900.90	7257.87	6989.76
(stddev)	125.45	75.87	219.55
Goodput (web mean)	1796.06	1974.35	2090.25
(stddev)	104.32	93.90	57.98
Web rate (h-mean) [0,5000)	28.17	25.84	25.71
[5000, 50000)	276.18	342.96	335.80
[50000, 500000)	472.09	612.08	571.40
[500000, ∞)	695.40	751.98	765.26
Fairness (bulk mean)	0.971	0.997	0.962

3.5.4 Traffic distributions

While TCP dominates cloud-based service traffic, the flows themselves are far from regular in their size, distribution, and duration. Here we evaluate the effects of varying traffic demands by considering Web requests that contend with long-running TCP flows for limiter bandwidth. To see whether our rate limiting algorithms can detect and react to Web-service demand, we assign 10 long-lived (bulk) flows to one limiter and the service requests to the other; this represents the effective worst-case for DRL since short and long flows cannot exert ordinary congestive pressures upon each other when isolated. We are interested in the ability of both traffic pools to attain the correct aggregate utilization, the long-term fairness of the stable flows, and the service rates for the Web flows.

Since we do not have access to traffic traces from deployed cloud-based services, we use a prior technique to derive a distribution of Web object sizes from a CAIDA Web trace for a high-speed OC-48 MFN (Metropolitan Fiber Network) Backbone 1 link (San Jose to Seattle) that follows a heavy-tailed distribution [202]. We fetch objects in parallel from an Apache Web server using `httperf` via a limiter. We distribute requests uniformly over objects in the trace distribution. Requests arrive according to a Poisson process with average μ of 15.

3.5.4 gives the delivered rates for the Web flows of different sizes and the delivered rates for the 10-flow aggregates in each scenario across 10 runs. This shows that the 10-flow

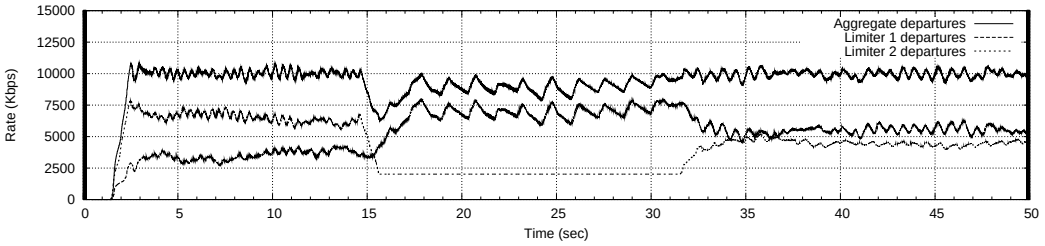


Figure 3.12 FPS rate limiting correctly adjusting to the arrival of bottlenecked flows.

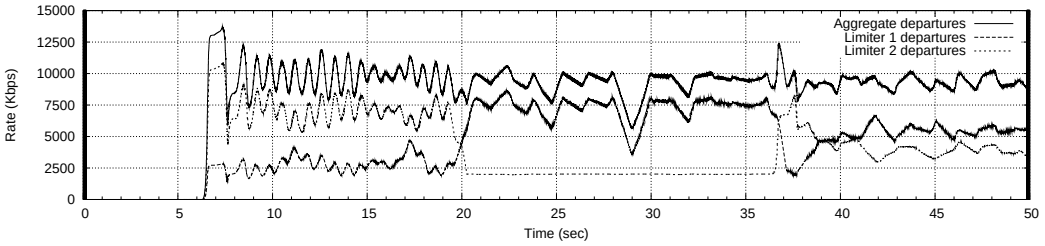


Figure 3.13 GRD rate limiting adjusting to the arrival of bottlenecked flows.

aggregate achieved a comparable allocation in each scenario. When seen in conjunction with the Web download service rates, it also indicates that the Web traffic aggregate in the other limiter received the correct allocation. Considering the Web flow service rates alone, we see that both GRD and FPS exhibit service rates close to that of a single token bucket, even for flows of significantly different sizes. The fairness index of the long-lived flows once again shows that GRD exhibits higher fairness than either CTB or FPS. FPS does not benefit from the fact that it samples flow-level behavior, which, in this context, is no more stable than the packet-level behavior observed by GRD.

3.5.5 Bottlenecked TCP flows

So far, the limiters represent the bottleneck link for each TCP flow. Here we demonstrate the ability of FPS to correctly allocate rate across aggregates of bottlenecked and unbottlenecked flows. The experiment in Figure 3.12 and Figure 3.13 begins as our baseline 3-to-7 flow skew experiment where 2 limiters enforce a 10 Mbps limit. Around 15 seconds, the 7-flow aggregate experiences an upstream 2-Mbps bottleneck, and FPS quickly re-apportions the remaining 8

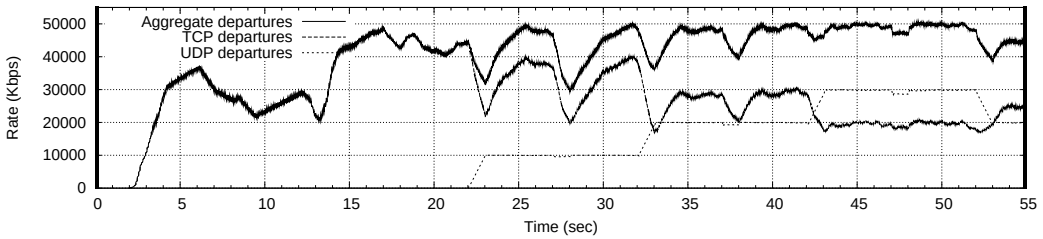
Table 3.3 Average throughput for 7 short (10-ms RTT) flows and 3 long (100 ms) RTT flows distributed across 2 limiters.

	CTB	GRD	FPS
Aggregate (Mbps)	10.57	10.63	10.43
Short RTT (Mbps)	1.41	1.35	0.92
(stddev)	0.16	0.71	0.15
Long RTT (Mbps)	0.10	0.16	0.57
(stddev)	0.01	0.03	0.05

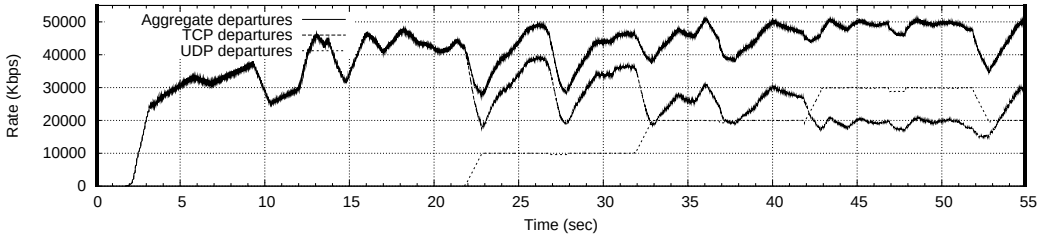
Mbps of rate across the 3 flows at limiter 1. Then, at time 31, a single unbottlenecked flow arrives at limiter 2. FPS realizes that an unbottlenecked flow exists at limiter 2, and increases the allocation for the (7+1)-flow aggregate. In a single pipe, the 4 unbottlenecked flows would now share the remaining 8 Mbps. Thus, limiter 2 should get 40% of the global limit, 2 Mbps from the 7 bottlenecked flows, and 2 Mbps from the single unbottlenecked flow. By time 39, FPS apportions the rate in this ratio.

3.5.6 Mixed TCP flow round-trip times

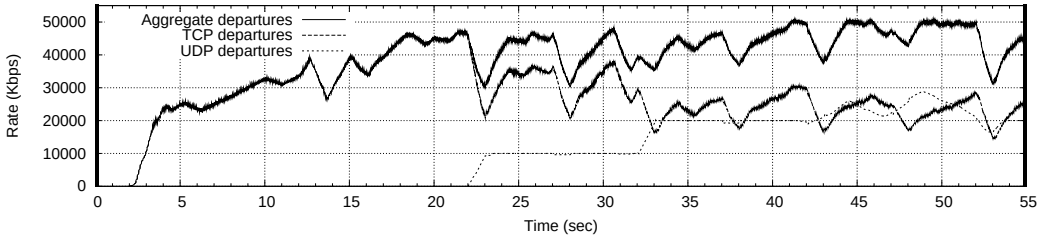
TCP is known to be unfair to long-RTT flows. In particular, short-RTT flows tend to dominate flows with longer RTTs when competing at the same bottleneck, as their tighter control loops allow them to more quickly increase their transmission rates. FPS, on the other hand, makes no attempt to model this bias. Thus, when the distribution of flow RTTs across limiters is highly skewed, one might be concerned that limiters with short-RTT flows would artificially throttle them to the rate achieved by longer-RTT flows at other limiters. We conduct a slight variant of the baseline experiment, with two limiters and a 3-to-7 flow split. In this instance, however, all 7 flows traversing limiter 2 are "short" (10-ms RTT), and the 3 flows traversing limiter 1 are "long" (100-ms RTT), representing a worst-case scenario. 3.5.6 shows the aggregate delivered throughput, as well as the average throughput for short and long-RTT flows for the different allocators. As expected, FPS provides a higher degree of fairness between RTTs, but all three limiters deliver equivalent aggregate rates.



(a) Central token bucket



(b) Global random drop



(c) Flow proportional share

Figure 3.14 Time series of forwarding rate of multi-protocol interactions. The global rate limit is 50 Mbps and RTT is 40ms. FPS and GRD use a gossip branching factor of 3 with a 50ms estimate interval.

3.5.7 Multi-protocol interactions

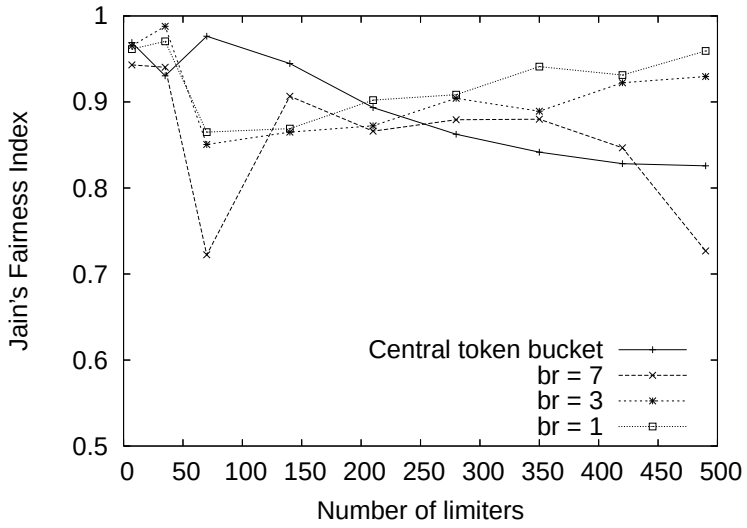
While TCP is the dominant transport protocol in use in cloud-based services today, there may be occasion to limit a mix of protocols. In the interest of completeness, we consider simultaneously limiting both congestion responsive and unresponsive flows by performing an experiment with constant-bit-rate (CBR) UDP traffic joining existing TCP traffic. We initially start three TCP flows at one limiter and seven more TCP flows at a second limiter 10 seconds later. For the next three 10-second intervals, we start a new 10-Mbps CBR UDP flow at its own limiter (again, a worst-case scenario), using 5 limiters in total at a global rate limit of 50 Mbps.

Once all UDP flows have started, we terminate them in reverse order every 10 seconds. Figure 3.14 presents a time series of this experiment. As expected, FPS provides a poor emulation of the central token bucket because it expects all flows (including UDP) to be congestion-responsive. As a result, FPS treats each UDP flow as a bottlenecked flow, and allocates rate proportionally to its knowledge of global weights. GRD, on the other hand, does not depend on the flow-fairness properties of TCP, and closely matches the behavior of a centralized token bucket.

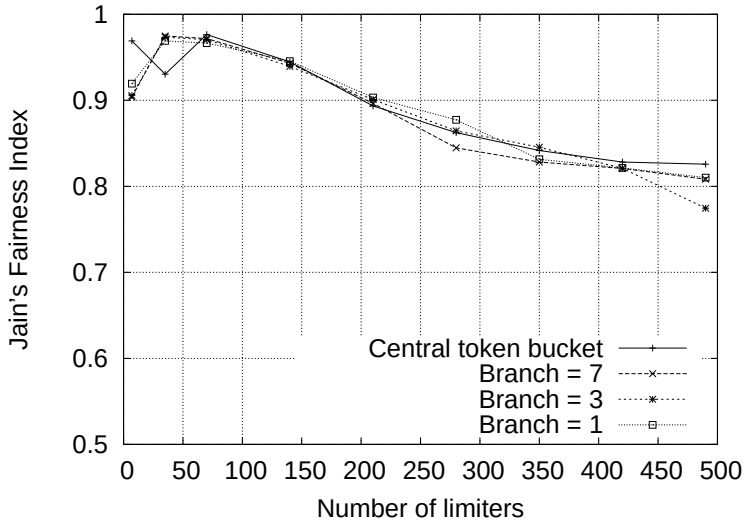
3.5.8 Scaling

We explore scalability along two primary dimensions: the number of flows, and the number of limiters. First we consider a 2-limiter setup similar to the baseline experiment, but with a global rate limit of 50 Mbps. We send 5000 flows to the two limiters in a 3-7 ratio: 1500 flows to the first and 3500 to the second. GRD and FPS produce utilization of 53 and 46 Mbps and flow fairness of 0.44 and 0.33 respectively. This is roughly equal to that of a single token bucket with 5000 flows (which yielded 51 Mbps and 0.34). This poor fairness is not surprising, as each flow has only 10 Kbps, and prior work has shown that TCP is unfair under such conditions [141]. Nevertheless, our limiters continue to perform well with many flows.

Next, we investigate rate limiting with a large number of limiters and different inter-limiter communication budgets, in an environment in which gossip updates can be lost. We consider a topology with up to 490 limiters; our testbed contains 7 physical machines with 70 limiters each. Flows travel from the source through different limiter nodes, which then forward the traffic to the sink. (We consider TCP flows here and use symmetric paths for the forward and reverse directions of a flow.) We set the global rate limit to 50 Mbps and the inter-limiter and source-sink RTTs to 40 ms. Our experiment setup has the number of flows arriving at each limiter chosen uniformly at random from 0 to 5. For experiments with the same number of limiters, the distribution and number of flows is the same. We start 1 random flow from the above distribution every 100 ms; each flow lives for 60 seconds.



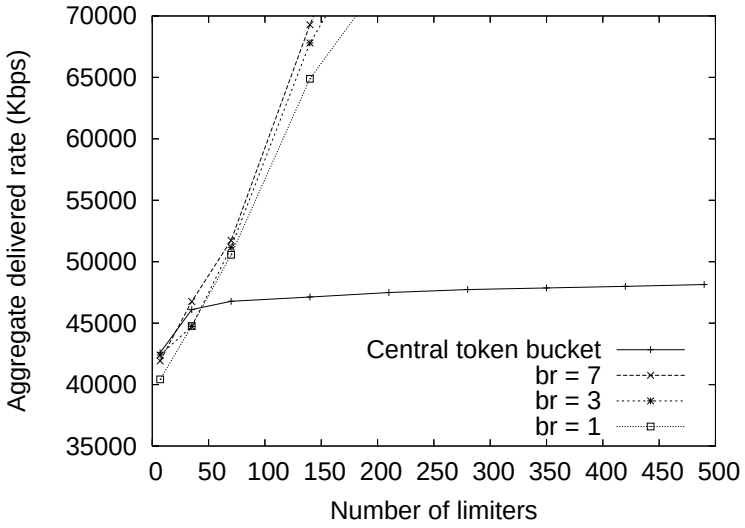
(a) GRD fairness.



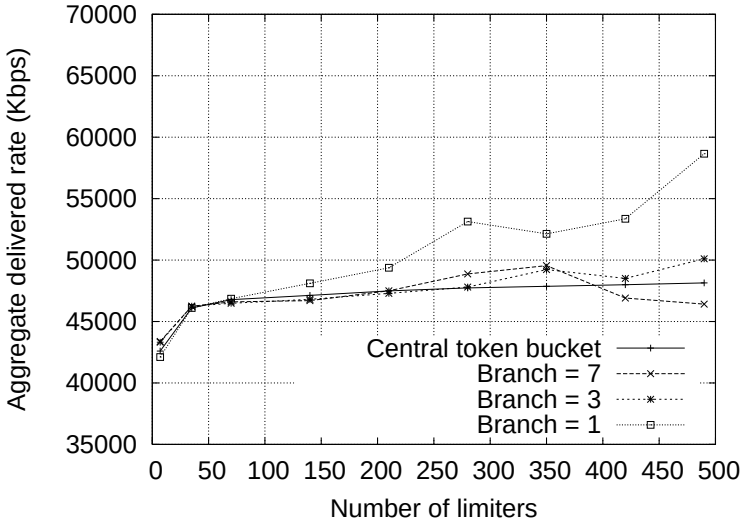
(b) FPS fairness.

Figure 3.15 Fairness vs. number of limiters in the scaling experiment.

To explore the effect of communication budget, we vary the branching factor (br) of the gossip protocol from 1 to 7; for a given value, each extra limiter incurs a fixed communication cost. Figure 3.15 and Figure 3.16 show the behavior of FPS and GRD in this scaling experiment. At its extreme there are 1249 flows traversing 490 limiters. (We stop at 490 not due to a limitation of FPS, but due to a lack of testbed resources.) When $br = 3$, each extra limiter consumes



(a) GRD delivered rate.



(b) FPS delivered rate.

Figure 3.16 Delivered rate vs. number of limiters in the scaling experiment.

$48 \times 20 \times 3 = 2.88$ KBps. Thus, at 490 limiters, the entire system consumes a total of 1.4 MBps of bandwidth for control communication.

We find that beyond a branching factor of 3, there is little benefit either in fairness or utilization. Indeed, extremely high branching factors lead to message and ultimately information loss. Beyond 50 limiters, GRD fails to limit the aggregate rate, but this is not assuaged by an

increasing communication budget (increasing br). Instead it indicates GRD's dependence on swiftly converging global arrival rate estimates. In contrast, FPS, because it depends on more slowly moving estimates of the number of flows at each limiter, maintains the limit even at 490 limiters.

This experiment shows that limiters rely upon up-to-date summaries of global information, and these summaries may become stale when delayed or dropped by the network. In particular, our concern lies with stale under-estimates that cause the system to overshoot the global rate; a completely disconnected system—due to either congestion, failure, or attack—could over-subscribe the global limit by a factor of N. We can avoid these scenarios by initializing limiters with the number of peers, N, and running a light-weight membership protocol [112] to monitor the current number of connected peers. For each disconnected peer, the limiter can reduce the global limit by $\frac{1}{N}$, and set each stale estimate to zero. This conservative policy drives the limiters toward a $\frac{1}{N}$ limiter (where each limiter enforces an equal fraction of the global aggregate) as disconnections occur. More generally, though, we defer analysis of DRL under adversarial or Byzantine conditions to future work.

3.5.9 Constraining distributed cheating

Finally, we subject FPS to inter-limiter delays, losses, and TCP arrivals and flow lengths similar to those experienced by a greedy user of a cloud-based service. As in Section 3.5.4, we are not concerned with the actual service being provided by the cloud or its computational load, nor with merely preventing cheating itself—we are interested in traffic demands and the ability of the limiters to provide useful service while enforcing the limit. Thus, we emulate a cloud-based service by using generic Web requests as a stand-in for actual service calls. We co-locate distributed rate limiters with 10 PlanetLab nodes distributed across North America configured to act as component servers. Without loss of generality, we focus on limiting only out-bound traffic from the servers; we could just as easily limit in-bound traffic as well, but that would complicate

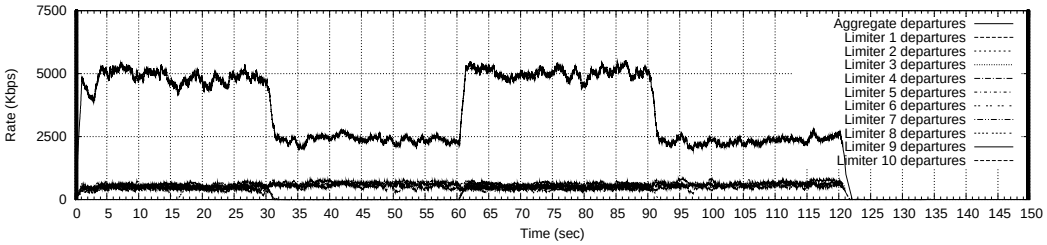
our experimental infrastructure. Each PlanetLab node runs Apache and serves a series of Web objects; an off-test-bed client machine generates requests for these objects using `wget`. The rate limiters enforce an aggregate global rate limit of 5 Mbps on the response traffic using a 100-ms estimate interval and a gossip branching factor of 4, resulting in a total control bandwidth of 38.4 KBps. (We use a 100 ms interval rather than a 50 ms interval to avoid bursty update losses due to PlanetLab per-sliver limits; we use a branching factor of 4 instead of 3 to partially compensate for the decreased gossip rate.) The inter-limiter control traffic experienced 0.47% loss during the course of the experiment.

Figure 3.17 shows the resulting time-series plot. Initially each content server has demands to serve 3 requests simultaneously for 30 seconds, and then the total system load shifts to focus on only 4 servers for 30 seconds, emulating a change in the service's request load, perhaps due to a phase transition in the service, or a flash crowd of user demand. Figure 3.17(a) shows the base case, where a static $\frac{1}{N}$ limiting policy cannot take advantage of unused capacity at the other 6 sites. In contrast, FPS, while occasionally bursting above the limit, accommodates the demand swing and delivers the full rate limit. GRD (not shown) exhibits erratic behavior due to the high inter-limiter delays (one observed maximum RTT was 800 ms) and loss (greater than 4%) on PlanetLab.

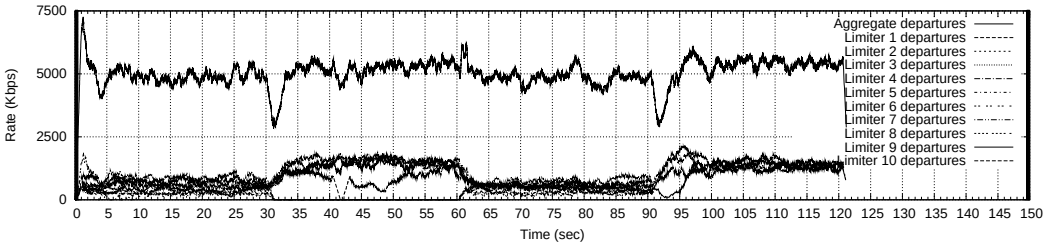
3.6 Related work

The problem of online, distributed resource allocation is not a new one, but to our knowledge we are the first to present a concrete realization of distributed traffic rate limiting.

Ensuring fairness across limiters can be viewed as a distributed instance of the link-sharing problem [76]. A number of packet scheduling techniques have been developed to enforce link-sharing policies, which provide bandwidth guarantees for different classes of traffic sharing a single link. These techniques, such as weighted fair queuing [59], apportion link capacity among traffic classes according to some fixed weights. These approaches differ from ours in



(a) $\frac{1}{N}$ limiting.



(b) Flow Proportional Share

Figure 3.17 A time-series graph rate limiting at 10 PlanetLab sites across North America. Each site is a Web server, fronted by a rate limiter. Every 30 seconds total demand shifts to four servers and then back to all 10 nodes. The top line represents aggregate throughput; other lines represent the served rates at each limiter.

two key respects. First, by approximating generalized processor sharing [150], they allocate excess bandwidth across back-logged classes in a max-min fair manner; we avoid enforcing any explicit type of fairness between limiters, though FPS tries to ensure max-min fairness between flows. Second, most class-based fair-queuing schemes aim to provide isolation between packets of different classes. In contrast, we expose traffic at each limiter to all other traffic in the system, preserving whatever implicit notion of fairness would have existed in the single-limiter case. As discussed earlier, we use a token bucket to define the reference behavior of a single limiter. There are a broad range of active queue management schemes that could serve equally well as a centralized reference [69, 75, 17, 69, 73, 149]. Determining whether similar distributed versions of these sophisticated AQM schemes exist is a subject of future work.

The general problem of using and efficiently computing aggregates across a distributed set of nodes has been studied in a number of other contexts. These include distributed monitoring [60], triggering [90], counting [185, 206], and data stream querying [19, 129], which we

review in Chapter 2. Two systems in particular also estimate aggregate demand to apportion shared resources at multiple points in a network. The first is a token-based admission architecture that considers the problem of parallel flow admissions across edge routers [30]. Their goal is to divide the total capacity fairly across allocations at edge routers by setting an edge router's local allocation quota in proportion to its share of the request load. However they must revert to a first-come first-served allocation model if ever forced to "revoke" bandwidth to maintain the right shares. Zhao *et al.* use a similar protocol to enforce service level agreements between server clusters [216]. A set of layer-7 switches employ a "coordinated" request queuing algorithm to distribute service requests in proportion to the aggregate sum of switch queue lengths.

3.7 Summary

Our work on DRL shows that it is possible to effectively limit the distributed resource consumption of greedy users and/or service classes, and to do so while providing inter-flow fairness and responsiveness, all with low coordination overhead. That we ensure these properties in DRL allows for straightforward deployment of DRL in a variety of increasingly popular distributed services. Refinements of GRD and/or FPS coupled with improved communication fabrics (perhaps that use hysteresis) may have the potential to reduce the coordination overhead of DRL and allow it to scale to thousands or even millions of limiters.

3.8 Acknowledgments

This chapter is in part a reprint of material that appears in the Proceedings of the ACM SIGCOMM Conference, 2007 by Barath Raghavan, Kashi Vishwanath, Sriram Ramabhadran, Kenneth Yocum, and Alex Snoeren.

Chapter 4

Decongestion Control

Thus far we have addressed the potential for cheating by potentially selfish network users via Distributed Rate Limiting at the network layer. While DRL can constrain the bandwidth consumed at many points in the network by a user or traffic class, it cannot cope with cheating behavior by users at the transport layer—specifically, regarding congestion control. One of the key problems in network design is ensuring that available capacity is fairly and efficiently shared between competing end points. Traditionally, fairness has been achieved either in the network itself using fair queuing at routers [59], or through the cooperation of end hosts using a common congestion control protocol such as TCP. Unfortunately, both approaches have significant drawbacks: fair queuing is expensive to implement, while end-host congestion control is typically far from optimal and, critically, relies on the goodwill of end hosts for success [179, 187].

While the Internet has relied upon end-host cooperation for some time, ill-conceived or intentionally-aggressive end-point behavior can drive a network without fair queuing into congestion collapse. This scenario is a classic tragedy of the commons; individual selfish behavior can drive the system to a globally pessimal state, yet there is no incentive for any user to unilaterally back off. Thus, in game-theoretic terms, the Nash equilibrium of the network congestion-control game is particularly suboptimal [6, 116, 117, 186, 211]. Researchers have pro-

posed a number of router-based enforcement mechanisms to avoid congestion collapse that vary in both complexity and effectiveness: some maintain per-flow state to provide near perfect fairness [27, 59, 76, 150, 189, 196], while others simply throttle the most aggressive senders [126, 149].

We observe that most of the complexity of both fair queuing and end-host-based congestion control is due to the perceived need to avoid dropping packets: in both models, well-behaved flows should experience little or no packet loss. Packet loss is taboo; to an Internet architect, it immediately signifies an inefficient design likely to exhibit instability and poor performance. In this chapter, we argue that such an implication is not fundamental. In particular, there exist design points that provide many desirable properties—including near optimal performance—while suffering high loss rates. We focus specifically on congestion control, where researchers have long held the belief that loss avoidance is central to high throughput. Starting with Jacobson's initial TCP congestion control algorithm [88], the traditional approach to end-to-end congestion control has been to optimize network performance by tempering transmission rates in response to loss [72]. We argue that by removing the unnecessary yoke of loss avoidance from congestion control protocols, they can become less complex yet simultaneously more efficient, stable, and robust.

Of course, there are a number of very good reasons to avoid loss in today's networks. Many of these stem from the fact that loss is often a symptom of overflowing router buffers in the network, which can also lead to high latency, jitter, and poor fairness. Hence, one should contemplate congestion control protocols that thrive in the face of loss solely in the absence of queues—or at least only in the presence of very short ones. Yet queues are not an end in and of themselves; they exist largely to smooth bursty traffic arrivals and facilitate fair resource sharing. If these goals can be met—or obviated—through other means, then router buffers may be removed, or at least considerably shortened.

In this chapter, we propose an alternative approach to congestion control that we call *decongestion control* [162]. As opposed to previous schemes that temper transmission rates to

arrive at an efficient, loss-free allocation of bandwidth between flows, we do not limit an end host's sending rate. All hosts send faster than the bottleneck link capacity, which ensures flows use all the available bandwidth along their paths. Our architecture separates efficiency and fairness; end hosts manage efficiency while fairness; for simplicity we focus on max-min flow fairness [89]. is enforced by the routers. (Decongestion control itself is also agnostic to the metric of fairness desired; as with XCP, network operators are free to select among different notions of fairness by implementing such policies in routers [97].)

We depart from previous architectures based upon network-enforced fairness by allowing overloaded routers to drop packets rather than queuing them, with the aim of decreasing the complexity and amount of resources required to enforce fairness. Our model suggests that efficiency can be maintained without requiring senders to decrease their transmission rates in the face of congestion.

Such an approach (sometimes referred to as a *fire-hose* [196]) may be dismissed due to the potential for congestion collapse—a condition in which the network is saturated with packets but total end-to-end goodput is low. However, congestion collapse occurs only under two conditions: if receivers are unable to deal with high loss (so-called *classical* congestion collapse), or if the network topology is such that packet drops occur deep in the network, thereby consuming network resources that could be fruitfully consumed by other flows [72]. The first concern can be addressed by applying efficient erasure coding [121, 133]. It is unknown whether the second condition arises frequently in practice; it occurs rarely in the backbone topologies we study.

Thus, we argue that packet loss may not need to be avoided, and that the potential exists to operate future networks at 100% utilization. Operating in this largely uncharted regime, where loss is frequent but inconsequential, raises a number of new design challenges, but also presents tremendous opportunity. We have identified several potential advantages over traditional schemes:

Robustness. Existing congestion control protocols are susceptible to a variety of sender

misbehaviors, many of which cannot be mitigated by router fairness enforcement. Because end points are already forced to cope with high levels of loss and reordering in steady state, decongestion is inherently more tolerant.

Efficiency. Sending packets faster than the bottleneck capacity ensures utilization of all available network resources between source and destination. With appropriate use of erasure codes, almost all delivered packets will be useful.

Simplicity. Because coding renders packet drops (and reordering) inconsequential, it may be possible to simplify the design of routers and dispense with the need for expensive, power-hungry fast line-card memory.

Stability. Decongestion transforms a sender's main task from adjusting transmission rate to ensuring an appropriate encoding. Unlike the former, however, one can design a protocol that adjusts the latter without impacting other flows.

Historically, congestion control has artificially intertwined the underlying goals of efficiency, fair multiplexing of resources, and reliable delivery. Specifically, current notions of efficiency, fairness, and reliable delivery in the Internet are inextricably tied to the behavior of TCP. Emboldened by recent initiatives to consider new Internet architectures, we consider a blue-sky approach and suggest that our goals should be efficiency, fairness, and reliability themselves. Taking a step back, we might ask: "What general approaches are there to achieve these goals?" In his seminal paper, Nagle observed that solutions to congestion control (and more generally, to the tragedy of the commons) are either authoritarian, cooperative, or "market-based" [143]. Using this taxonomy, today's Internet is largely cooperative with occasional deployments of authoritarian router-based enforcement; we offer decongestion control as a third path: a practical approach to Internet-wide congestion control that requires neither compliant end hosts nor high-powered authoritarian routers.

Given our substantial departure from conventional congestion control, we do not attempt to exhaustively address all of the details needed for a complete protocol, a process which

requires iteration via real-world deployment. Instead, we make the following concrete contributions:

- We introduce the concept of *decongestion control* and discuss the ways in which congestion control designs can be uncoupled from network packet loss.
- We present a rigorous study of the potential for congestion collapse in a number of realistic backbone topologies under a variety of conditions. In particular, we confront the key concern raised by our approach: By saturating network links, decongestion control is in danger of wasting network resources on *dead packets*—packets that are transmitted over several hops, only to be dropped before arriving at their destinations [72, 102].
- We demonstrate that while abundant, dead packets frequently do not impact network-wide goodput when fairness is enforced by routers. We introduce the concept of *zombie packets*, a far more rare, restricted class of dead packets, and show that they are the key cause of congestion collapse.
- We propose a decongestion control algorithm that eliminates zombie packets, and implement a prototype called Opal. We evaluate several basic properties by comparing its performance to TCP in simulation and identify key areas of future work, including improving the efficiency of very short flows.

To be clear, we are not claiming that existing approaches to end-to-end congestion control are ill-considered. Rather, we assert that a far wider design space exists than researchers have previously explored, and evaluate the pros and cons of one particular alternative. Our simulation results augur well for decongestion control; if the the challenges presented by short request/response flows can be overcome, decongestion may provide an avenue to deliver performance similar or superior to existing approaches while simultaneously alleviating a number of long-standing vexing issues.

4.1 Context and approach

Decongestion control is optimistic: senders always attempt to over-drive network links. Should available capacity increase at any router due to, for example, the completion of a flow, the remaining flows instantaneously take advantage of the freed link resources. To translate increased throughput into increased goodput, senders encode flows using an erasure coding scheme appropriate for the path loss rate experienced by the receiver.

4.1.1 Incentive compatibility

Decongestion control sidesteps many aspects of greed and malicious behavior. Because routers enforce fairness, it is nearly impossible for users to unilaterally increase their goodput by injecting more packets into the network. Moreover, in a network dominated by decongestion control flows senders are transmitting at a high rate anyway. The most effective way for a sender to increase its goodput is to adjust its coding rate, which, as previously mentioned, has no impact on other flows. This contrasts with TCP, whose throughput can be gamed in a number of ways, such as via simple parameter modifications by senders or via the misbehaving-receiver attacks of Savage *et al.* [179].

A well-designed decongestion control protocol may be similarly more robust to malicious behavior due to its time independence. Senders adjust coding rates based upon reported throughputs—not individual packet events—so they are not as sensitive to short-term packet behaviors as with TCP. In particular, there is little opportunity to launch well-timed bursty "shrew" attacks [110], which have been shown to be lethal to BGP [213]. Ideally, decongestion might reduce all attacks to bandwidth attacks: there should be nothing more effective that a malicious source can do than send traffic at a high rate, which both routers and decongestion end hosts are already designed to tolerate.

4.1.2 Simplicity

Much of the complexity in today's routers stems from the elaborate cross-connects and buffering schemes necessary to ensure the loss-free forwarding at line rate that TCP requires. In addition, TCP's sensitivity to packet reordering complicates parallelizing router switch fabrics. Adding traditional fairness enforcement or policing mechanisms to high-speed routers even further complicates matters [127, 149, 189, 196]. Lifting the requirements for loss-free, in-order forwarding may enable significantly simpler and cheaper router designs. Decongestion control requires only a fair dropping policy to enforce inter-flow fairness. A number of efficient schemes have been proposed in the literature [95, 148], and it may be possible to further reduce the implementation complexity in core routers [196].

In addition to their inherent complexity, a significant portion of the heat, board space, and cost of high-end routers is due to the need for large, high-speed RAM for packet buffers. While Appenzeller *et al.* have argued that smaller router buffers may suffice for large TCP flow aggregates [13], smaller router buffers make TCP more vulnerable to bursty DoS attacks [110, 213]. (Arisoylu *et al.* have also studied rate allocation in networks that have small buffers [14].) Decongestion, however, needs almost no buffering. Theoretically, this is not surprising; previous work has shown that erasure coding can reduce the need for queuing in the network. In particular, for networks with large numbers of flows, coding schemes can provide similar goodput by replacing router buffers with coding windows of similar sizes [29]. Practically speaking, in addition to reducing cost, small buffers also decrease the variance and maximum-possible end-to-end queuing delay. In decongestion control, router buffers the role of buffers is inverted. Buffers no longer exist to avoid losses during traffic bursts, as in traditional networks. Instead, they exist only to ensure that output links are driven to capacity: if the offered load is roughly equal to the router's output link capacity, small queues are needed to absorb the variation in arrival rates. We show in Section 4.5.2, however, that if the offered load is paced or sufficiently large, we can virtually eliminate buffering. Moreover, small queues help to bound delay and jitter for end-to-end traffic.

In addition to simplifying the routers themselves, decongestion control may also ease the task of configuring and managing them. Today, traffic engineers expend a great deal of effort to avoid overload during peak traffic times or during planned or unplanned maintenance. If a link fails, traffic is diverted along a backup path; as a result, the backup path may become congested, which, for TCP, can result in catastrophic performance. Decongestion control, by contrast, can operate effectively at a lower goodput despite the high loss rates caused by severely restricted capacity. Similarly, researchers have proposed deliberately diverting traffic through overlays, intelligent multihoming, and source routing to improve performance and reliability. In such systems, packets are switched across paths as traffic conditions change; additionally, in some designs, packets are sent across multiple paths simultaneously. Decongestion control's insensitivity to packet loss and reordering is a perfect fit for such aggressive route change and multipath mechanisms.

4.1.3 Stability

Traditional end-to-end congestion control protocols need to probe the network for available bandwidth [11, 70, 88, 93, 208]. While many techniques exist for conducting probes, they all require the injection of additional traffic into the network, which may be at capacity. The result is a bursty traffic load—which router queues must absorb—and short-term oscillations in available bandwidth—that further complicate end hosts' attempts to conduct measurements. We propose that senders dynamically adjust the coding rates and methods employed based on recent throughput rates. A key distinction between adjusting the coding methods and changing the transmission rate, however, is that the encoding has no impact on other flows. Hence, changes in available capacity may be less frequent than with TCP.

The challenges facing bandwidth-probing end-to-end control loops are well documented [97, 120]. In particular, Low *et al.* have shown analytically that TCP's control loop becomes oscillatory and prone to instability in high capacity and/or long-delay environments.

Furthermore, they conjecture that it is unlikely that active queue management schemes can combat this trend. This issue is particularly acute during slow start, when the sender needs to rapidly re-discover an appropriate rate. While numerous modifications to TCP have been proposed to improve slow-start [71], they still must rely upon complex mechanisms to help TCP rapidly discover additional available capacity should it become available during congestion avoidance. Moreover, even router-assisted protocols like XCP [97] are likely to be slightly inaccurate and can only capture newly available capacity on the order of round-trip times.

4.2 Fairness enforcement

Decongestion control starts from the premise that in-network fairness enforcement is required to limit the impact of greedy or misbehaving senders [59, 186]. The notion of enforcing a fair share of bandwidth for flows at the router was introduced as early as 1984 by Nagle [142]. Since then, researchers have proposed countless approaches to in-network fairness enforcement and active queue management (AQM). We hope to simplify many aspects of router design by freeing router architects from the constraints long imposed by TCP's unique quirks. Before moving forward, however, it is important to precisely define what form of fairness we expect routers to provide.

4.2.1 Defining fairness

Fairness in the Internet is inextricably tied to TCP's notion of fairness—that is, approximate max-min fairness [89] on a per-flow basis. This flow-fair notion has endured, and, if not principle, then perhaps pragmatism has driven the designers of competing protocols to tailor their notions of fairness to what is "TCP-friendly." In recent years, however, there has been a growing call to reevaluate whether the notion of flow fairness is the right one [37]. While there is increasing awareness of this issue, no consensus has emerged.

We use max-min flow fairness in our study of decongestion control. To be clear, we

do *not* seek to be TCP friendly, nor contemplate interacting with TCP in the same network. For now, we assume all flows in the network use decongestion. We define a "flow" to consist of all packets between two hosts, irrespective of ports or other multiplexing, so no communicating parties can increase their share by adding more flows [8]. In addition to aiding comparison and understanding, we opt for max-min flow fairness for several specific reasons. First, it is similar to TCP, and TCP's fairness is well understood. Second, it can be implemented in routers using only local knowledge with existing algorithms, including, but not limited to fair *queuing* algorithms such as deficit round robin (DRR) [189] and fair *dropping* algorithms such as approximate fair dropping (AFD) [148]. We opt to use fair dropping, AFD in particular, which benefits from $O(1)$ per-packet complexity, no per-class or per-flow state, and appears likely to enjoy widespread deployment (AFD is expected to ship in the next generation of routers from Cisco). Third, max-min fairness—but not just max-min—provides a useful property that we leverage in our design.

4.2.2 Brickwall dropping

In an idealized model, routers implement what we term a *brickwall* policy, meaning, given a fixed set of competing demands, an overloaded router provides each flow with a well-defined maximum outgoing share of the link regardless of the amount of demand in excess of its share that a flow offers. Concisely, throughput is *not* proportional to offered load: increasing a flow's arrival rate above its fair share will not increase its delivery rate. Most explicitly designed router fairness mechanisms that we are aware of (including AFD) have this property. Both FIFO and RED queuing are *not* brickwall policies, however. In particular, a flow can increase its share of the bottleneck by offering more load [113].

In some sense, brickwall enforcement is the opposite of economically-oriented fairness policies, among which Kelly's fairness [99] is most notable. In these schemes, routers drop packets in proportion to flow arrival rates. While it would be ideal to enforce a global, economically-

derived fairness criterion, the approaches we are aware of require the sender to behave rationally to perceived congestion. Moreover, such designs might also result in a change of operation that would impact applications in unintended ways—they would magnify the benefit to those on high-bandwidth connections, possibly undermining the spirit of fair bandwidth sharing in the Internet as it exists today.

We believe, however, that notions of fairness other than max-min may be more appropriate for today's Internet. Due to decongestion's inherent separation of fairness enforcement from end-host transmission, there is cause for optimism that our architecture may admit alternative definitions. For now, however, we suggest that max-min is more than adequate for our initial exploration, and defer the important and interesting question of whether decongestion can be made to work with other fairness models to future work.

4.2.3 Implications for senders

One task of the sender in decongestion control is to apportion its outgoing link capacity across flows. Because flows to different destinations will traverse various portions of the network—and, therefore, encounter distinct bottlenecks—some flows may be able to put the capacity to more effective use. Recall that the goal in decongestion control is to over-drive the bottleneck for each flow. The remaining question, however, is to what degree? The answer depends on the dropping policy enforced at the routers. Bottlenecked flows, by definition, will experience loss. In a brickwall network, bottlenecked flows do not benefit from additional load. Any level of bandwidth over and above the flow's bottleneck rate provides equivalent goodput in the brickwall regime. Furthermore, links downstream of the bottleneck will not observe any difference; we evaluate this in greater depth next.

4.3 The impact of loss

There are two distinct impacts of packet loss, direct and indirect. Packet loss within a flow directly reduces the throughput delivered to the intended receiver. Indirectly, packet loss may cause inefficient use of upstream network resources: flows other than the one experiencing loss may have been able to put the upstream capacity to better use.

4.3.1 Erasure coding

The congestion collapses of 1986 and 1987 gave networking researchers pause, and led to the pioneering work of Jacobson to develop end-to-end congestion control [88]. This type of congestion collapse—termed *classical* congestion collapse by Floyd and Fall [72]—occurs when the network is busy forwarding packets that are duplicates or otherwise irrelevant when they eventually arrive at their destination. (Nagle observed that networks with infinite buffers are especially susceptible [143]). TCP's congestion control mechanisms seek to ensure that a network of TCP senders avoids classical congestion collapse by judicious use of ARQ (automatic repeat request), and enhancements such as SACK further ensure that only useful packets will be retransmitted.

Erasure codes provide an alternative mechanism for transport protocols to ensure that delivered packets are useful. Their application in networking is not new. Following the vast literature comparing ARQ schemes with forward error correction (FEC) schemes [114], there have been several approaches to integrating TCP and FEC [122, 173]. As we observe in Chapter 2, FEC can be placed below, inside, or above TCP—thus, respectively, FEC can hide losses from TCP, be used to prevent retransmissions, or be applied to application-layer datagrams. Similarly, work on fountain codes have highlighted the feasibility of non-ARQ based transports [43, 44]. However, many of these approaches are rooted in today's network architecture, and are in large part designed to serve only in an auxiliary role to avoid wasted transmissions or to be used in domain-specific environments.

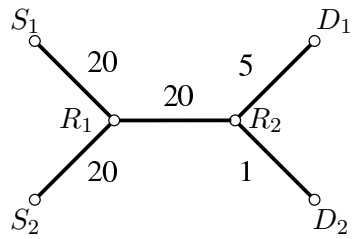


Figure 4.1 Dropped and dead packets. Each link is labeled with its capacity.

Luby's development of *rateless* erasure codes—codes for which a practically-infinite number of unique erasure-coded blocks can be efficiently generated—makes it possible to leverage erasure coding to transmit data streams that are almost impervious to loss [121, 133]. Unlike traditional codes, such as Reed-Solomon, encoding in a rateless fashion can be quite fast—many require only XOR operations—but are not maximum-distance separable (MDS). MDS codes require the reception of only the same amount of data as the original unencoded content. Non-MDS codes require the delivery of some small ϵ more coded blocks than the length of the original payload. For most codes, ϵ depends on the size of the erasure coded payload, and in practice can be as low as 0.03, as we observe in Section 4.5.2—or even lower with the benefit of partial acknowledgments from the receiver. We leverage Online codes [133] in our decongestion control prototype to inoculate end-to-end flows against ordinary packet loss, which has the potential to stave off classical congestion collapse with only small losses in goodput.

4.3.2 Dead packets

The far greater concern facing decongestion control is the potential inefficiency due to dead packets that are transmitted over several hops, only to be dropped before arriving at their destinations. To be precise, we call a packet dead if it is dropped at a router other than its source's access router.

Figure 4.1 illustrates this distinction. Suppose there are two flows, $S_1 \rightsquigarrow D_1$ and $S_2 \rightsquigarrow D_2$, and each sender is transmitting at a rate of twenty, saturating its access link. Both routers enforce max-min flow fairness, so the throughput of the $S_1 \rightsquigarrow D_1$ flow is five, while the

throughput of $S_2 \rightsquigarrow D_2$ is only one. Twenty packets are dropped at R_1, ten from each flow, but none are dead, as R_1 is the access router. At R_2, however, five more are dropped from the $S_1 \rightsquigarrow D_1$ flow. These are dead packets, as they traversed the bottleneck link $R1 \rightarrow R2$; if S_1 were sending to D_1 at a rate of five, five more packets from the $S_2 \rightsquigarrow D_2$ flow could have traversed the link. In this instance, however, the goodput of the network would not increase, so these dead packets are inconsequential. It is unknown whether there are likely to be other flows that would make better use of this "wasted" capacity in general [196], especially in networks with fairness enforcement. We set out to address this question by simulating the behavior of fire-hose protocols like decongestion in networks with router-enforced max-min fairness.

Loss sites

When Internet demand exceeds network capacity, the precise set of locations where loss is likely to occur is the subject of considerable debate and disagreement. For some time, many researchers have suspected that the effective bottlenecks in the Internet are at the edge, due to low-capacity access technologies such as POTS, ADSL, and analog cable. However, the story is much more complicated. Access networks have been growing in capacity in recent years, and in some countries a large fraction of users have fiber connections to the home [51]. As a counterpoint, edge links have not always been seen as the bottlenecks, even in domestic backbones: public peering points such as MAE-EAST and MAE-WEST were among the primary sites of congestion during the mid to late 1990s [98].

Others have observed that congestion points appear to be dispersed—Akella *et al.* find that bottlenecks are distributed evenly between intra-ISP links and inter-ISP peering links [7]. And further still, spectacular collapses such as those caused by the Baltimore tunnel [47] and Taiwan earthquake [201] cause one to suspect that there may be instances where relatively few links carry a great deal of the traffic, in which case there is likely to be a great deal of loss at their ingress. In contrast, some have made the argument that the network has or will have near infinite capacity, and that congestion is a minor concern, if at all [145]. It may seem to be the

case that these various contentions are mutually exclusive, but we suspect that they are likely all reflective of what is happening in *some* real networks.

In contrast to previous studies, our concern is not with where or if loss might occur in networks dominated by TCP. Instead, we are interested in whether fire-hose approaches like decongestion control cause significant, fundamental inefficiency when max-min fairness is enforced. We conclude from the lack of consensus on the matter of Internet choke points that a definitive answer is likely to be elusive and certainly requires a more comprehensive and in-depth study than can be reported on here. Hence, we restrict ourselves to the more modest goal of making a case for decongestion by considering a small set of intra-domain topologies with the hope of showing instances where it holds promise. We defer a rigorous study of what aspects of network topologies and traffic demands make them amenable to decongestion—and whether they can be engineered for—to future work.

Network topologies

We aim to better understand, for a given network topology, bandwidth assignment, and flow distribution, whether decongestion will induce congestion collapse due to dead packets. Our experiments are limited to a single "network" or autonomous system (AS); we do not consider super-topologies of many ISPs all connected by direct peering links or connections at public exchanges because we do not have access to such topologies annotated with link capacities and routing tables. In our experiments, we consider both actual and synthetic topologies; existing topologies demonstrate whether decongestion is practical today, while synthetic ones allow us to consider whether it will remain or become practical tomorrow.

We consider two separate real—if not necessarily representative—large-ISP PoP (point of presence)-level topologies, an August 2007 snapshot of the GEANT2 pan-European research and education network, and a once-public Level 3 PoP-level map from 2006 annotated with link capacities. The GEANT2 topology consists of a variety of European research networks attached to a core of 21 PoPs connected by a 10-Gbps backbone; we prune the topology to the 35 core

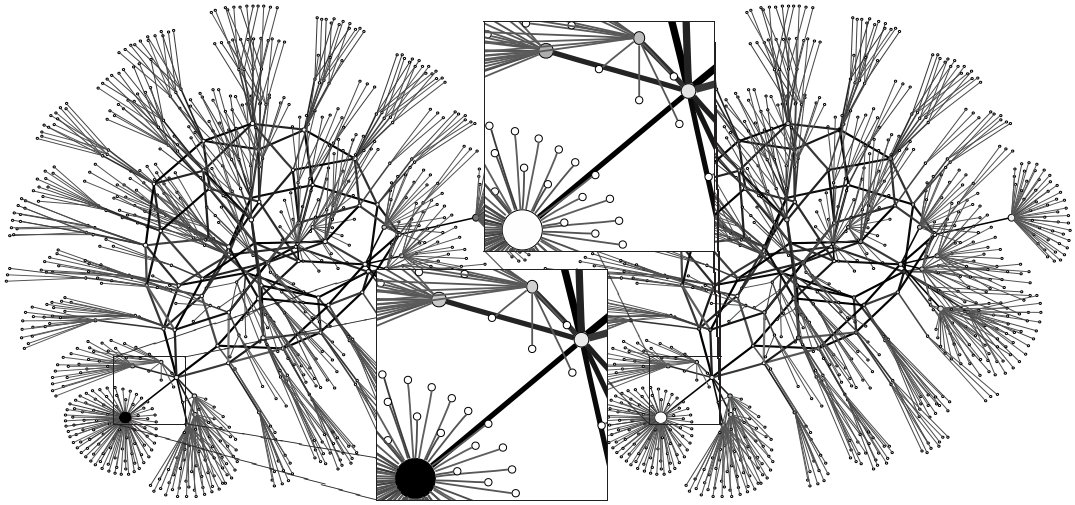


Figure 4.2 The prevalence of dead (left) and zombie (right) packets in the HOT with arithmetic capacity assignments.

links for which we were able to obtain routing and traffic information. The Level 3 topology contains 66 PoPs connected primarily by OC-198 links and secondarily by OC-48 and OC-12. In the synthetic case, we use the HOT router-level topology [111], which is thought to be highly representative of a large ISP's router-level topology. Our HOT graph consists of 989 links and 939 nodes, 768 of which are edge nodes. We augment the HOT graph with a three-tier bandwidth hierarchy, and consider two different options for assigning capacity to each tier. The first is an assignment of geometrically increasing capacities—100 Mbps on edge links, 1 Gbps on inter-PoP links, and 10 Gbps on core links—the second an arithmetic assignment of 100 Mbps at the edge, 200 Mbps at the PoP, and 300 Mbps in the core.

Traffic distributions

The prevalence and location of dead packets in any given network depends on traffic demands. For simplicity, we study steady-state behavior and do not concern ourselves with the absolute amount of data transfered between origin/destination (OD) pairs; instead, for each source we need only the distribution of destinations and proportion of demand reflected by each.

We obtained actual GEANT2 traffic matrices for August 1, 2007. In those snapshots,

the demand does not saturate—let alone overdrive—the network. To generate higher demand that continues to reflect the distribution of the original traffic matrix, we attach leaf nodes to each flow's actual origin and destination and source and sink the simulated traffic at these leaf nodes. (Hence, each leaf node sources and sinks precisely one flow.) The total leaf node access capacity at each original node is identical to the actual capacity of the original node; each leaf node's incoming/outgoing capacity is set in proportion to the fraction its flow represents of the actual node's total original incoming/outgoing demand.

We do not have access to information about the traffic on Level 3's backbone, so we generated a synthetic traffic matrix following a log-normal distribution as recommended by Nucci *et al.* [144] and augmented the topology in the same manner as for GEANT2. Unfortunately, the log-normal model depends on a number of parameters that are not specified in the HOT topology, so we construct two simple models. First, we use uniformly distributed connections—pairs of edge nodes are selected to communicate uniformly at random. Second, we model client-server traffic patterns through exponentially distributed connections with a wide range of exponents. We select the source from this exponential distribution and the destination uniformly at random. As a point of reference we find that GEANT2 traffic is well modeled by an exponential distribution with exponent 0.5.

Simulation

To facilitate evaluation at scale, we implemented a flow-based simulator that models perfect max-min link fairness. The simulation is not packet-based and is intended to model only steady-state flow behavior. The simulator takes as input a topology annotated with link capacities and a set of OD flow demands. Each flow is routed via its shortest path (or actual route based on IS-IS link weights in the case of GEANT2) from source to destination, and is subject to no notion of propagation, transmission, or queuing delay. At each link, we derive the output flow allocation using a max-min apportionment of the input flows. We compute the steady state of the network based on the offered load using a fixed-point algorithm,

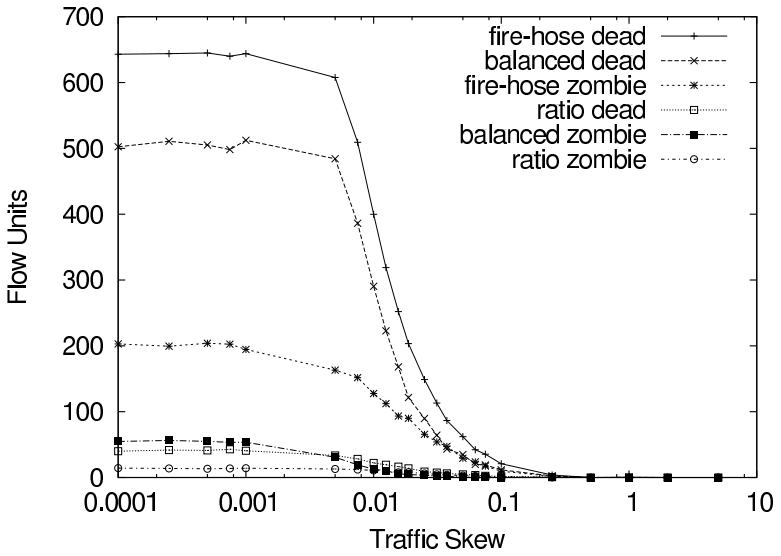


Figure 4.3 Number of dead and zombie packets as a function of traffic skew for the HOT topology.

We are now equipped to answer the question of if and where dead packets may occur. We can simulate the effect of an uncontrolled fire-hose protocol by setting all of the OD flow demands to be infinite, and keeping track of the amount of loss at each router. The left-hand side of Figure 4.2 depicts the prevalence of dead packets under fire-hose in one run of our simulator on the HOT topology, using arithmetic link capacity assignments and 10,000 flows distributed uniformly at random. Each router's diameter is scaled according to its capacity and shaded according to the amount of dead-packet induced loss occurring at that router, each link is sized according to its capacity and shaded according to its utilization. In both instances, light shades correspond to low values, higher values are darker. Note almost all links are completely utilized. In this run, almost all the dead packets occur at one particular router with a large attached subtree, called out in the lower left.

Intuitively, a uniform traffic demand will produce the most dead packets, as flows are least likely to share bottlenecks. As traffic matrices become skewed, more and more demand bottlenecks at the access links of a few popular sources, so a smaller portion of the packet loss will be caused by dead packets. Figure 4.3 confirms this hypothesis by plotting the total number of dead packets for fire-hose ("fire-hose dead," we discuss the ratio, balanced, and zombie lines in Section 4.3.4) in the HOT topology with arithmetic capacity assignments shown earlier for

increasing degrees of demand skew. Because our simulator does not model packets *per se*, we cannot use packets as the quantity of dead traffic. Instead, we consider the total quantity of dead "flow"—a unit-less quantity normalized to the access link bandwidth. I.e., if a single sender had all of its demand dropped in the network after forcing out another sender's demand, there would be one dead flow. For ease of exposition, however, we will continue to refer to this quantity as dead packets.

Each simulation data-point considers 10,000 unique source-destination flows (a flow includes all traffic from a source to destination) with sources chosen from an exponential distribution with exponent λ and destinations chosen uniformly at random. Results with a uniform distribution of sources ($\lambda = 0$, not shown) are essentially identical to those with $\lambda = 0.0001$. As expected, the number of dead packets generated by fire-hose decreases as demand concentrates.

It is difficult to directly compare the dead packet counts in real topologies, as the demand distributions and capacity hierarchies are quite different. For GEANT2, we scaled the traffic demands proportionally so that each node's demand is 50 times its capacity. Fire-hose induces 1,674 Gbps of dead packets on 162 Gbps of goodput. Level 3, at a similar demand level, produced a far higher number of dead packets: 112,000 Gbps on a goodput of 211 Gbps.

4.3.3 Potential inefficiency

While the absolute number of dead packets may seem alarming at first glance, the far more important metric is the degree of inefficiency these dead packets may cause. We use the results of our fire-hose simulation to compute the optimal max-min flow assignment in an iterative fashion through an algorithm we call *clamp*. We begin by selecting the fire-hose flow with the globally minimum throughput (ties are broken arbitrarily) and set its demand in the optimal assignment to be the throughput currently achieved in the simulation. We then subtract that demand from the capacity of each link traversed by the flow, remove the flow from the traffic matrix, and rerun the simulator. At each step, we select the flow with minimum throughput and

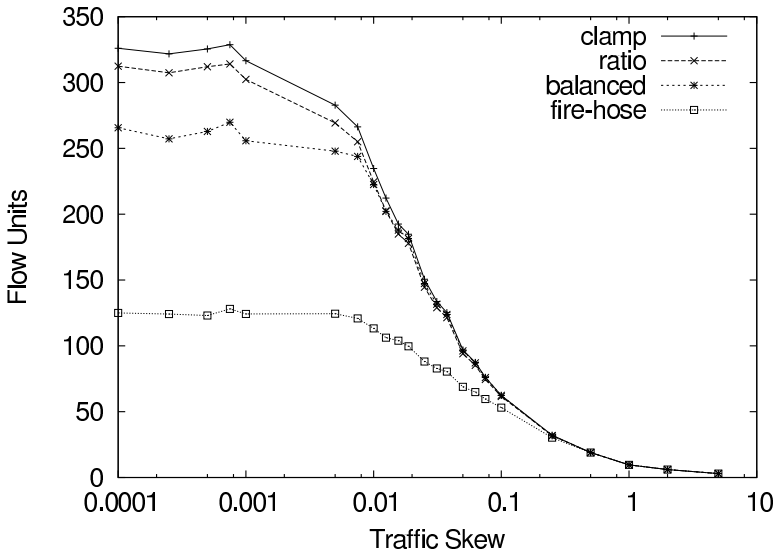


Figure 4.4 Total network goodput vs. traffic skew (λ) in the HOT topology with geometric capacity assignments.

use its achieved value as the flow's demand. After iterating for each flow, we arrive at an optimal set of traffic demands. The flow demands computed by *clamp* are admissible by construction.

In the case of GEANT2, clamp achieves a goodput of 192 Gbps, only 19% higher than fire-hose's 162 Gbps despite the huge number of dead packets that fire-hose creates. Fire-hose is similarly efficient in Level 3, as clamp is able to achieve 250 Gbps as opposed to fire-hose's 211. Figure 4.4 shows the network-wide goodput achieved by fire-hose and clamp (again, the ratio and balanced lines are introduced later) in the same HOT topology over a variety of traffic demands. The relative performance of fire-hose is quite poor for low skew. For the case of arithmetic capacity assignments, the goodput of fire-hose degrades to 40% of optimal in the worst case. Most networks are unlikely to have arithmetic bandwidth hierarchies, however. Fire-hose substantially improves in the HOT topology with geometric capacity assignment—to within 15% of optimal (not shown). Regardless of capacity, adding more flows decreases the relative performance of fire-hose; conversely, as the number of flows decreases, fire-hose approaches optimal (also not shown). Just as in real topologies, however, the decrease in performance of fire-hose does not correspond directly to the number of dead packets. The absolute difference in goodput between fire-hose and clamp in the HOT topology with uniform demand is approximately 200 flow units,

far less than the 650 units of dead packets shown in Figure 4.3.

4.3.4 Zombie packets

This gap actually corresponds directly to a slightly different quantity. The issue is not purely the presence of dead packets, rather the presence of dead packets that cause potentially live packets (those that would have otherwise contributed to some flow's goodput) to be dropped. Only this restricted class of dead packets, which we term *zombie* packets, are deleterious from the network's potential goodput. Returning to the example in Figure 4.1, if the link from $R_2 \rightarrow D_2$ had capacity greater than ten, any packets dropped from flow $S_1 \rightsquigarrow D_1$ at R_2 would be zombies.

Counting dead packets

We determine which dead packets in our simulation are actually zombie packets in the following fashion. For a given network topology and traffic pattern, we compare the simulated throughput of each flow with the ideal (the throughput computed by clamp). For each dropped packet in the simulation, we consider whether it causes the flow to decrease below its ideal allocation. Such a drop will occur only if another flow on the same link is over-sending. In other words, another flow will be restricted further along its path, and the packets dropped there are actually zombies. We superimpose the number of zombie packets caused by fire-hose in Figure 4.3. It is reassuring to note that this quantity is precisely the respective difference in goodput between fire-hose and clamp in Figure 4.4 (the same is true for GEANT2 and Level 3): In our simulation, zombie packets are the *only* cause of inefficiency. The right-hand side of Figure 4.2 shows that almost none of the dead packets at the access router impact performance, as the capacity could not have been effectively used by any other flows.

Reducing zombies

Now that we've precisely identified and quantified the source of inefficiency in a fire-hose approach, a natural question is whether it is fundamental to all decongestion control protocols.

So far, we have considered the simplest case of oblivious senders who fill their outgoing link with packets equally distributed among all of their destinations. We say that such a sender is exerting equal effort on behalf of each flow. Rational senders, on the other hand, seek to maximize their own self-interest, i.e., the total goodput across all of their flows' destinations. In particular, if they are sending zombie packets on any flow that detract from the goodput of their other flows, they are clearly motivated to adjust their behavior. We use this insight to devise for two slightly more sophisticated send algorithms, *balanced* and *ratio*.

A balanced sender seeks to equalize the loss rate experienced by flows all of its destinations. In other words, if any flow is seeing proportionally more drops than another, the sender transfers some of its send effort from a flow seeing high loss to a flow seeing low loss. Regardless of the distribution of effort, a balanced sender always fills up its access link. A ratio sender observes that blindly filling ones access link may be fruitless: any effort in excess of each flow's bottleneck capacity does not increase throughput in the steady state (although it may not hinder it, either). Recall that the main benefit of over-driving a flow is being able to instantaneously take advantage of newly available capacity. A ratio sender balances this opportunity with the potential of injecting zombie packets by attempting to achieve a certain, fixed loss rate for each of its flows. Said another way, a ratio sender transmits each flow a fixed constant, $\rho > 1$, faster than the throughput currently reported by the receiver.

In addition to clamp and fire-hose discussed previously, Figure 4.3 and Figure 4.4 report on the behavior of balanced and ratio (where $\rho = 1.2$) senders as well. Ratio approaches clamp as ρ goes to 1. As one might expect, balanced sending results in far more dead packets than ratio, but both significantly reduce the number of zombie packets, allowing their goodput to approach optimal. Balance and ratio achieve at least 80% and 97% of optimal, respectively, for the arithmetic capacity HOT topology, and 96–99% for a geometric capacity hierarchy (not shown). For both GEANT2 and Level 3, balanced is no better than fire-hose (84%), while ratio achieves 97–98% of optimal.

Our simulator is measuring steady state behavior with perfect knowledge, in which case there is no benefit to over-sending if the sender is aware of the true bottleneck capacity. A more complete measure of goodput depends on the dynamics of the system, which we cannot evaluate without making a number of engineering decisions. We therefore flesh out a particular algorithm based on the ratio sender in the next section before subsequently evaluating it further.

4.4 A prototype design

We now describe our initial approach to designing a decongestion control protocol, Opal. Our intention is to identify key tradeoffs and offer our design choices not as mandates, but simply as a concrete way forward.

4.4.1 Basic operation

Opal is a reliable transport protocol that transmits a paced packet stream composed of erasure-coded windows of data blocks which we call *caravans*. Each caravan represents some n application data blocks; in our prototype, each block is 1000 bytes. These data blocks are then encoded using an erasure code; the erasure code's blocks do not necessarily correspond to the application data blocks. In particular, when using Online codes, we subdivide each application data block into 10 blocks of size 100 bytes and perform coding over these smaller blocks. Finally, the coded blocks are placed into packets and sent. With a rateless code, the sender can generate a virtually limitless stream of encoded blocks from a fixed set of data blocks. (The strict limit on the number of unique encoded blocks in a rateless code is the size of the powerset of the number of blocks, which is 2^k given k coding blocks.) The sender selects n, and assigns a monotonically increasing caravan number, and, within each caravan, a packet sequence number starting at 0. The receiver collects packets from the same caravan together and decodes them to recover the original payload.

As with TCP, all packets contain ACK data that is sent irrespective of whether there is

application data to deliver. Opal ACK headers include a cumulative caravan acknowledgment—the number of the last completely received caravan and the total number of packets received that contributed to the flow's goodput, irrespective of the caravans to which they belonged. The receiver also encodes a *partial* ACK consisting of a caravan-wide bitmap; each bit in the map denotes whether the corresponding block has been received and decoded into its source-data form. Both the sender and receiver transmit their packet reception rates to each other in a symmetric fashion.

4.4.2 Managing goodput

In the context of this work, we only consider decongestion control protocols that provide reliable transport. As a result, we require, as every reliable transport protocol does, an acknowledgment (ACK) mechanism to confirm reception, and thus face many of the same challenges as TCP. (Just as TCP's algorithm has admitted a host of improvements over the years, we expect that an equally large number of optimizations can be made to a decongestion sender.)

Opal's acknowledgment mechanism is similar in many ways to TCP. The key difference in Opal, however, is that senders transmit data in caravans and only one ACK per caravan is required. In comparison to TCP, Opal is less sensitive to moderate increases in the loss rate, and, thus, has a less urgent need to adjust its sending behavior.

Sizing caravans

A caravan of erasure-coded packets is akin to a freight train—it can push through anything but is hard to stop: Up to one full bandwidth-delay product of outdated data can be in flight when the sender receives a caravan's ACK. This sets up a clear trade-off between short and long caravans: many coding schemes cannot deliver data from a caravan to the application until decoding the entire caravan, so applications that demand low latency require short caravans. However, short caravans are less efficient for both the control loop and for rateless erasure codes. A potential solution to this that we have yet to explore is to smoothly transition transmission

between caravans by decreasing the current caravan's send rate while increasing that of the subsequent caravan. This relies upon the sender's ability to accurately anticipate the point at which the current caravan will be acknowledged, which is ultimately tied to local history about network conditions.

Currently, we opt to increase caravan size as often as possible so as to improve the efficiency of erasure coding and to decrease inter-caravan waste. Starting with a fixed initial caravan size, we monitor the goodput of each caravan, looking for a decrease of more than a fixed percentage—currently 20%—below the expected goodput; if such a decrease occurs, we shrink the caravan's size, currently by a factor of 8. We double the caravan's length otherwise, up to a maximum caravan length of 5,000 blocks. We have yet to explore the best strategies for flows that pause intermittently and do not have enough data to send at all times to fill a full caravan. (TCP's push flag and Nagle's algorithm are designed to address a similar issue.)

Adapting coding parameters

Given a fixed caravan size, the sender selects the type and rate of coding to use. It is important to note that changes in the coding or in caravan internals are not externally-visible events—they simply have the effect of changing the *information* rate, and ultimately, the application-to-application goodput of the flow. For caravans of size 100 or less, we encode using Reed-Solomon coding with an equal number of parity blocks (that is, for a caravan of size 100, we generate 200 coded blocks, only 100 of which are needed to decode); above that size, we use Online codes [133]. When generating blocks for coding using Online codes, we subdivide packets into 100-byte words. In the course of integrating Online codes into Opal, we devised several helpful optimizations, including the use of partial ACKs. In Opal, receivers perform decoding of the coded blocks incrementally, and thus recover some original source data blocks within a caravan before others. Receivers then notify senders which original data blocks have been fully decoded, and senders simply skip random seeds—used in selecting the in-degree of each coded block and which data blocks to XOR—that yield a coded block that consists solely of

source blocks already received. We find that for our implementation, a small number of partial ACKs from the receiver can significantly improve coding efficiency.

More generally, there are several tradeoffs: while rateless codes are designed to decode fast, they are efficient at encoding only asymptotically, and thus typically require large caravans, often on the order of a few hundred blocks, which is why we subdivide packets into smaller blocks during encoding. In addition, there is no requirement that caravans within a flow use the same coding scheme—in fact, far more efficient, fixed-rate codes exist for particular small caravan sizes and loss rates, as do fully general but non-rateless codes such as Reed-Solomon. In practice, a sender can use a lookup table to select an optimal coding scheme for a given caravan size and expected loss rate. For extremely small caravan sizes, simple redundant packet transmission suffices.

4.4.3 Allocating bandwidth

Senders face three challenges in allocating their limited outgoing bandwidth. First, they aim to use their bandwidth for each flow efficiently; sending substantially above the brickwall rate of that flow is unlikely to yield dividends in terms of throughput. Second, they aim to effectively apportion their bandwidth between flows to different destinations. Third, they must reserve bandwidth for ACKs for any flows for which they are the destination. Senders perform these allocations explicitly using only local information.

Setting a flow's rate

Following the ratio algorithm described previously, senders set the transmission rate of each flow to induce a fixed loss rate. Thus, we send a fixed multiplicative factor above the current observed throughput. If a sender observes that it is obtaining a throughput of 2 Mbps for a flow, then it sends at $(2 \cdot \rho)$ Mbps, where $\rho = 1.2$ in our implementation. The excess 20% of packets will both capture bandwidth if and when it becomes available and enable the sender to detect a change in its brickwall bottleneck rate. Our experience suggests that for modest (≤ 1.5) settings

of ρ, assuming it is sufficiently large to be likely to capture freed bottleneck capacity, the precise value does not have a dramatic effect. Accordingly, we have not yet attempted to identify the optimal setting for any particular topologies.

Sharing limited bandwidth

Many senders will have orders of magnitude more access capacity than the sum of their Internet flows' bottleneck bandwidths, so it is unlikely that they will ever need to be parsimonious with transmission bandwidth. Such may not be the case for large servers or in wireless or local-area networks, however, where the networks are fast or experience congestion at the MAC layer; losses in these environments may happen at the hands of a non-brickwall process.

To combat this issue, it may be necessary to compare loss rates between flows in the manner of the previous balance algorithm. For now, we assume each host has a single, dedicated access link of fixed capacity. Should the sum of a sender's outgoing flow rates ever exceed this capacity (which may happen during flow start-up, see below), the sender restricts its flows as a router would, implementing a brickwall, max-min fair policer on its access link.

Reserving ACK bandwidth

Opal includes ACKs for the reverse-direction flow in every packet it sends, so bi-directional flows have no need to set an explicit ACK transmission rate. However, senders must weigh the benefit of transmission goodput with reception goodput, and decide how to apportion bandwidth to flows for which they are only receiving data, or those that only send data sporadically. In the case that ACKs do not compete locally for bandwidth with other flows (the receiver has outgoing access bandwidth to spare), we draw inspiration from TCP and use the incoming data rate to determine the outgoing ACK rate. Unlike TCP, however, we do not perform explicit ACK clocking; receivers pace ACK transmissions independently from packet reception.

In a decongestion-controlled network, ACKs are likely to be lost or compressed, as the ACK path may have very different congestion than the data path. Hence, we cannot use the

data packet reception rate by itself to determine the ACK rate. Instead, in Opal we use the ACK *reception* rate at the sender to set the ACK rate at the receiver. Thus, akin to setting a data transmission rate, the receiver learns of the throughput of its ACKs and sets its transmission rate such that the ACK throughput is a fixed fraction of the data throughput. This approach maintains several of the beneficial properties of TCP's control loop, notably that receivers are not obligated to transmit ACKs if they are not receiving packets from the sender. In our current implementation, we set the ACK rate to be 2% of the data rate, which yields roughly the same ratio of ACK to data bandwidth as TCP for MTU-sized packets. In the common case, TCP receivers send delayed ACKs and use TCP timestamps and SACK, so for maximum sized packets, each 52 byte ACK is a reply to 3000 bytes of data packets and the ACK channel's bandwidth is 1.73% of the data channel's. In Opal, ACKs that include partial ACKs are up to 150 bytes each (allowing for partial ACKs for caravans of up to 1000 blocks), and at a rate of 0.02 or 2%, correspond to 5 data packets.

4.4.4 Start-up behavior

At flow startup an Opal host must make two choices: how large of a caravan to send, and how fast to send it.

In practice, the appropriate first caravan size depends on the amount of data in the transmit socket buffer, which varies depending on the application involved. For now, we focus on bulk senders—ones that have an unlimited amount of data to send while active—and select a fixed initial caravan of 10 packets, striking an imperfect balance between the needs for low-delay delivery of application data to the receiver's socket buffer and the needs of the sender to estimate its goodput adequately. Ideally, applications could provide hints to the Opal stack via system calls regarding their startup needs.

There are three prominent approaches to determine how fast to send a flow initially. The first, used by TCP, is to begin transmission slowly and rapidly ramp up. Such a timid approach

is antithetical to the ethos of decongestion. The second, used by router-based protocols such as RCP [63] and XCP [97], is to rely upon routers to tell end hosts at what rate to send. The third is to use statistics collected previously about the network, which can be particularly valuable for short flows [21].

We choose not to rely upon routers to explicitly dictate initial sending rates, because appropriate operation requires obedient senders and cooperation across mutually distrustful ASes and hosts. Hints from routers, on the other hand, such as those provided by QuickStart TCP [71], might work well. Both history and hint-based approaches require significant engineering and testing on real networks to implement effectively, so for now we assume that each host is preconfigured with the capacity of its closest bottleneck access link. We set the initial send rate of a flow to half of this value.

4.5 Experimental evaluation

We evaluate the performance of Opal using the `ns2` network simulator. Opal is realized as a (short-term) fixed-rate sender that communicates partial and caravan ACKs in packet headers. Both ends of each Opal flow continually update their rate estimates and adjust caravan sizes and transmission rates accordingly. Each sender's rate is randomized very slightly in order to avoid simulation artifacts due to artificial synchronization. Max-min fairness is currently enforced at routers using the built-in stochastic fair queuing (SFQ). A research-grade implementation of AFD we obtained does not operate well at the scale of our simulations; SFQ with a very short buffer size suffices for our current purposes. We begin by validating several of our findings from our flow-level simulator, and then proceed from macro-level experiments to micro-level experiments, and test Opal on smaller topologies.

Table 4.1 Goodput comparison for fire-hose and ratio/Opal senders in simulation.

	Arithmetic		Geometric	
	`ns2`	Sim	`ns2`	Sim
Clamp (optimal)	—	126.0	—	252.7
Ratio/Opal	110.5	121.1	251.1	250.4
Fire-hose	90.4	101.7	243.5	244.7
TCP New Reno	96.0	—	221.0	—

4.5.1 Simulator validation

To facilitate comparisons between the `ns2` Opal implementation and the simulation results presented earlier, we also implement fire-hose in `ns2` as an extremely high-rate (slightly randomized) constant bit-rate (CBR) sender and repeat (scaled-down versions of) several of the steady-state simulations conducted on the HOT topology. Table 4.1 compares the goodput achieved by both approaches as implemented in `ns2` with the results provided by our simulator (where ratio corresponds to Opal). In order to bound the simulation time, we restrict these simulations to 500 uniformly distributed flows with access link speeds of 10 Mbps. To eliminate startup effects, we run the simulation for thirty seconds, but only consider the last fifteen. All goodput values are normalized to the access link capacity.

The main result from Table 4.1 is that the performance of our `ns2` implementation is quite similar to the simulated idealized ratio sender, giving us confidence that further experimentation based on the `ns2` implementation can be interpreted in the same light. Fire-hose however, performs slightly worse in in `ns2` implementation. Upon examination, we found that this performance degradation was due to imperfect fairness delivered by SFQ. Finally, we include the goodput achieved by TCP New Reno. Interestingly, both fire-hose and Opal outperform TCP, Opal by a substantial margin.

4.5.2 Smaller topologies

Next we consider the behavior of Opal in smaller-scale topologies. Our goal is not to be exhaustive, but rather to consider a few common cases that shed insight into the behavior of

our initial prototype.

Coding

Before we delve into the behavior of Opal during operation, we first consider the performance of our erasure coding. We do not implement any form of erasure coding itself in our `ns2` implementation. Instead, we compute erasure coding overhead off-line through our stand-alone C++ implementation of Reed-Solomon and Online codes. During the simulation, partial ACKs reflect blocks received, not decoded, and we assume a caravan has been fully decoded (and move on to the next one) based upon the type of coding used in that caravan. For redundant and Reed-Solomon caravans, we declare the caravan over after precisely n distinct blocks have been received. For larger caravans that use Online codes, we wait until $(n \cdot (1 + \epsilon))$ blocks have been received. By selecting an appropriate ϵ, we ensure that we report a lower bound on the achievable goodput.

We built a coding simulator to test the performance of our rateless codes under varying conditions. Using the $\rho = 1.2$ parameter to determine the loss rate expected on the forward channel and a rate of 2% of the flow's goodput for ACKs, our Online codes implementation leverages partial-ACK information to reduce the erasure-induced overhead for caravans of more than 100 packets (1,000 blocks) to less than 1%. In the absence of partial ACKs, our implementation still keeps the coding overhead to 3%, as originally suggested by Maymounkov [134].

Basic stability

We compare the stability of Opal with that of TCP New Reno. We do not intend to argue that TCP is deficient, as there are many TCP variants that likely perform better; we use New Reno as a well-understood reference point.

In Figure 4.5, we push a single bulk flow between a pair of hosts through a 10-Mbps dumbbell topology with 10-ms link delays. For this bulk flow, with TCP, we use 25-packet buffers, while for Opal we use only 5. For comparison, we also plot the small-buffer case for TCP. The

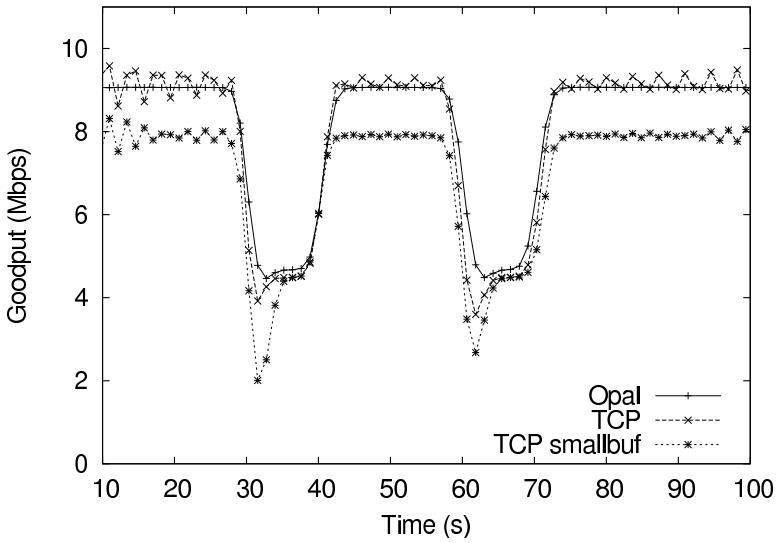


Figure 4.5 Opal and TCP reacting to congestion induced by an unresponsive UDP flow.

RTT of the flow is 60 ms. At 30 seconds and again at 60 seconds after the beginning of the simulation, we inject a 10-second burst of 10-Mbps CBR UDP traffic between another pair of hosts sharing the bottleneck link. The results are unsurprising, but lend credence to the notion that even an unengineered prototype of Opal can match TCP's steady-state throughput.

Self-congestion

A more interesting experiment considers the impact of loss on Opal's ACK channel. In order to induce disproportional loss on the ACK channel, we introduce a UDP CBR flow with the same source and destination as the flow under test, just in the reverse direction. As a result, the routers place the Opal ACK and UDP traffic in the same bin and provide no isolation between the two. Figure 4.6 shows that Opal automatically increases its ACK rate to compensate for the increased loss on the ACK channel, suffering almost no penalty at the hands of the UDP flow. By comparison, TCP's performance is well known to deteriorate, an effect only exacerbated by small buffers.

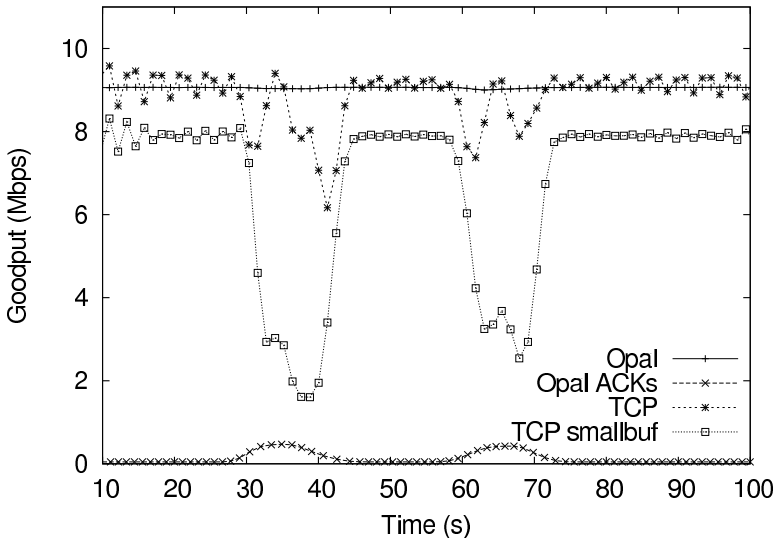


Figure 4.6 Opal and TCP reacting to congestion induced by self-induced reverse flow congestion. The Opal ACK rate is the rate of transmission of ACKs from the receiver.

Flow join

We test Opal through a fixed bottleneck with the join and departure of 10 flows, in sequence, shown in Figure 4.7. By definition, each flow has a distinct source, and we select distinct destinations as well. Fairness on the bottleneck link is dictated by the router; notably, however, Opal flows recapture the capacity freed by a departing flow faster than TCP, as the aggregate goodput indicates.

Short flows

Perhaps the most worrisome aspect of Opal is the potential for poor performance on short flows, say, less than one bandwidth-delay product, where the sender seems guaranteed to waste capacity. Intuitively, Opal has the potential to overdrive the link for some time while waiting for the ACK from the receiver. This problem is not unique to Opal, since many congestion control protocols, notably TCP, have an initial probing phase during which they may overdrive the link, but Opal is particularly aggressive in its current form.

In Figure 4.8 we consider a challenging scenario for Opal: the rapid arrival of many short flows through the same bottleneck. In this experiment, we test the arrival of 100 flows

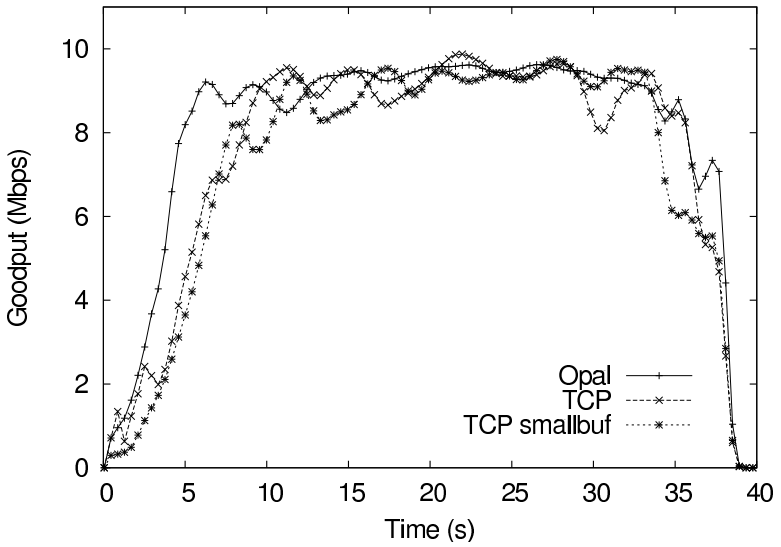


Figure 4.7 Opal reacting to congestion induced by flow joins and departures.

per second; each flow is only 20 KB long. Opal suffers as expected—each sender attempts to overdrive while sending limited amounts of data, only to cause significant overrun. We report on each Opal flow's goodput only once it completes, and as a result, the computed delivery rate ebbs.

In practice, all connections (in the TCP sense) between a single source destination are multiplexed on a single Opal flow, so there is reason for hope. In particular, a sophisticated implementation would pipeline caravans (and, as a consequence, pipeline series of small requests such as a Web page download) to decrease wasted capacity.

Jitter

While bulk transport flows seem to do well with Opal, we would also like that protocols that rely upon low delay and jitter (multimedia and VoIP, for example) do not suffer under decongestion. Just as most specialized streaming protocols use a transport other than TCP today, we expect that analogous protocols will exist in a decongestion controlled network. Thus, our concern is with the jitter induced upon a bare, CBR packet stream that is competing with a large number of Opal flows at a bottleneck.

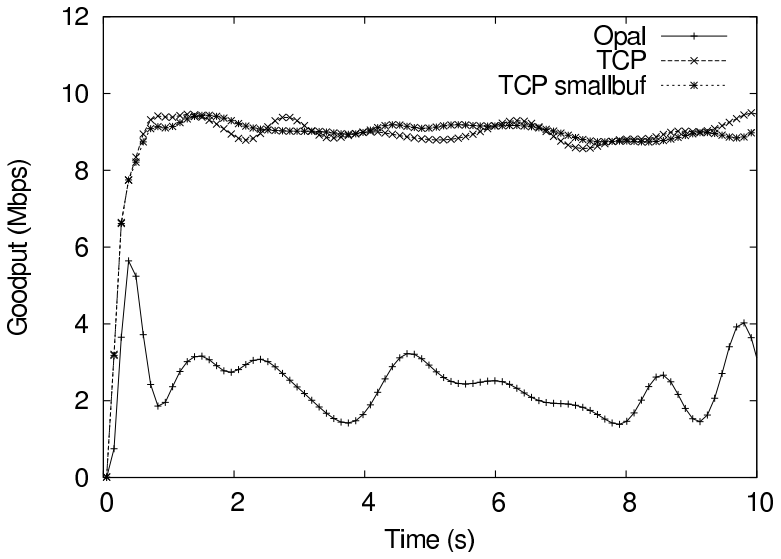


Figure 4.8 Opal and TCP reacting to the rapid arrival of many short flows.

Figure 4.9 shows the jitter of a CBR UDP flow meant to model the G.729A VoIP codec (100 pps, 64 Kbps) when competing with 30 Opal or TCP flows that arrive every 5 seconds and last for 30 seconds each. We calculate the jitter based upon the recommendation RFC 1889 for calculating jitter in RTP [181]. Since Opal uses very short-to-non-existent router buffers, there is little opportunity for varying queue levels, and thus, the jitter remains low.

Delay

One final metric of concern is delay, which is related to the depth of the queues along a packet's path. Opal will likely saturate queues at most routers, so Opal flows will generally suffer maximum delay. One of the key potential benefits of decongestion control, however, is that it operates with drastically reduced queue sizes as compared to TCP. Traditionally, the rule of thumb was to provision routers with at least a bandwidth-delay product worth of buffering to overcome TCP's burstiness. For high-speed backbone routers, this implies queue sizes that are both prohibitively expensive and difficult to implement. Recent work has shown, however, that the aggregate behavior of a large number of flows on high-capacity links is less bursty, and suggests that it is possible to scale down the queue size by the square root of the number of

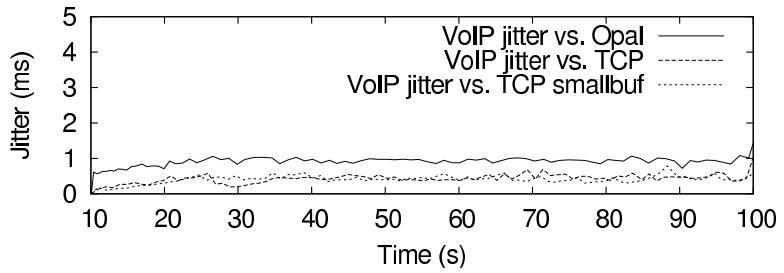


Figure 4.9 A simulated VoIP flow's jitter when competing with Opal and TCP.

flows [13]. Further studies suggest that an even smaller queue may suffice if TCP sends at a paced rate [65]; Shorten and Leith also examine queue provisioning and its effect on TCP [188].

Decongestion control is not bursty by nature: Once a caravan begins, packets are transmitted continuously until the data has been received. In such a regime, router buffers no longer exist to absorb large bursts and meter them out during the ensuing lull; instead, queues are kept consistently full and exist only to smooth out instantaneous variations in arrival rates. (If the router has no queue at all, and packets arrive at multiple interfaces at the same instant, only one can be forwarded.) Thus we ask: how long must the queue be to ensure that it never becomes empty (at which point the output link may go idle)?

In TCP, under-utilization during periods of high demand is caused by the filling of the router queue and the ensuing drops cause exponential back-off [65]; if there are too many flows in back-off, the link utilization goes down. Opal has no exponential back-off and the concern, instead, is that the queue become empty due to the simultaneous arrival of packets from several flows. The rate adaptation of Opal suggests a natural tendency away from synchronized overlaps in a situation where the bottleneck link is not overdriven. Even if we simulate completely unresponsive CBR senders with initial random offsets, we observe good utilization using small queue lengths. Using a dumbbell topology with 400 flows collectively sending at the bottleneck bandwidth of 40 Mbps, we see no loss with queues larger than 9 packets and only 10% loss with a 5-packet queue. Moreover, slower links likely to be severely overdriven can employ even shorter queues.

4.6 Related work

We are not the first to explore the relationship between erasure coding and congestion control. Researchers have compared ARQ schemes with FEC schemes [114] and integrated TCP and FEC [122, 173]. Fountain codes [121, 133] famously highlighted the feasibility of non-ARQ based transport [43, 44] for broadcast and bulk data transmission. We are unaware, however, of any previous work that directly addresses the question of what sort of congestion control protocol—encoded or otherwise—would be best suited for a network with pervasive router-based max-min fairness enforcement.

Researchers have also previously proposed pushing networks to their capacity and beyond. Isarithmic networks are always fully utilized, just with *empties* when end hosts have no useful data to transmit [57]. Tracking empties proves problematic, however: just as a token ring protocol requires a token-recovery mechanism, an isarithmic network requires some way to ensure empties are not lost forever. At the application level, Speak-Up deliberately over-drives servers and encourages clients of DDoS victims to increase their sending rates in order to drown out the attackers [204], yet all request streams remain congestion-controlled through TCP.

4.7 Summary

Our results indicate that there exist changes to the Internet architecture that can enable protection from end-host cheating at the transport layer, and that for the topologies and traffic demands we study, decongestion control holds promise. A great deal more study is required to determine just how efficient a practical decongestion protocol could be; our current design clearly admits many optimizations. A well-engineered solution could have fair bandwidth allocation with simple routers, predictable traffic patterns, low latency and jitter, and sender isolation. We recognize that considerable additional development is required to convert on each of these potential benefits, but the ensuing rewards may be the effort. Moreover, even if deployment of decongestion turns out to be impractical, its design raises questions about long-standing Internet

architectural principles that may currently be promulgated neither of necessity nor sufficiency, but habit.

4.8 Acknowledgments

This chapter is in part a reprint of material that appears in the Proceedings of the ACM SIGCOMM Workshop on Hot Topics in Networks, 2006 by Barath Raghavan and Alex Snoeren.

Chapter 5

Future Directions

The two approaches to cheat-proof networking examined in this dissertation only scratch the surface of the Internet architecture challenges and issues that must be addressed in the years to come. In parting, we consider subsequent work in the space of Distributed Rate Limiting and Decongestion Control and explore several future directions for work in this space.

5.1 Impact

Our work on Distributed Rate Limiting demonstrated that it is possible to limit the aggregate bandwidth consumed by potentially greedy network users at hundreds of locations, and to do so with low inter-limiter communication overhead. Our work on Decongestion Control addressed the design of congestion control in an environment dominated by greedy users, and in doing so challenged the conventional wisdom regarding the network and end-host performance impacts of aggressive transport protocols.

Since our initial study of Distributed Rate Limiting [164], others have conducted further research in the area. Though we showed that one algorithm—Flow Proportional Share—computes stable allocations, we did not show its convergence, nor did we attempt to construct a theoretical framework within which DRL algorithms could be studied. Thus, of particular note

is the work of Stanojevic and Shorten on the theoretical underpinnings of DRL. They found that it is both possible to provide strong theoretical guarantees for DRL and also generalize the underlying problem statement, though they do not experimentally validate their results [194]. In their work they also developed revised versions of Global Random Drop and Flow Proportional Share and prove various properties about their algorithms.

Since we introduced Decongestion Control [162], researchers have continued to explore the space in several directions. Lopez *et al.* have explored the game theoretic consequences of the use of a FEC-based protocol versus ordinary TCP-like congestion control, and have found in a simple model that hosts unilaterally switch to using the FEC-based protocol [118]. Their work indicates that were a decongestion-like protocol implementation be made publicly available, its use might spread rapidly. In addition, Bonald *et al.* approached the question of congestion collapse in decongestion-like networks from an analytic perspective and concluded that congestion collapse may be unlikely [32].

5.2 Direct extensions

In our study of cheat-proof networking protocols, we have not addressed the problem of identifying traffic and traffic clusters within the network. These topics provide ample opportunity for future study.

5.2.1 Distributed Traffic Classification

Our work on Distributed Rate Limiting leaves an important issue unresolved: how to identify traffic to limit. This is the job of a distributed traffic classifier, for which there are several key issues to address: devising concise representations of packet information, defining upper and lower bounds for detection guarantees, and applying traffic classification results in a distributed, active policer.

The consequences of a narrow view of the network are immediate: those who wish to

consume more resources, thwart network policies, or simply operate under the radar can do so across multiple hosts or addresses. (Another consequence is that auditing traffic becomes difficult; we consider auditing next.) In this context, a natural problem is distributed rate limiting, as we have studied. Nevertheless, the distributed rate limiters policing the user's traffic should adapt to changing demands and provide fairness guarantees comparable to a centralized network pipe. Before we can perform distributed rate limiting, however, we must select users or traffic clusters to limit—in our work on DRL, we assumed that some external mechanism was used to pre-process traffic before presenting it to a limiter.

To resolve this question we turn to distributed traffic classification, which requires that individual network observation points share summaries of the traffic they see. In this way, traffic aggregates [66] cannot escape detection simply by spreading across multiple links. We believe that we must consider treating classic network traffic policing problems from this new, pan-network perspective. In this dissertation we have addressed distributed rate limiting using algorithms that look at packet and/or flow behavior at different levels of granularity and communicate between different network choke points to limit traffic aggregates.

Unlike DRL, distributed traffic classification requires semantic information about the traffic streams seen at various network locations; this may increase coordination overhead significantly. There are two basic approaches to distributed traffic classification: coordinating fundamentally centralized classifiers and emulating a centralized classifier using a distributed algorithm. The former approach could use techniques such as Autofocus [66] to summarize local traffic and then communicate the resulting summaries to others, though such an approach may miss important traffic clusters. Alternatively, the latter approach could use similar techniques to those in distributed rate limiting to directly emulate the behavior of a centralized classifier by sharing summaries of the actual packets being processed.

5.2.2 Traffic identification

While most network users are cooperative, or at worst, greedy, others are explicitly malicious. Such users mount attacks on a daily basis, leaving ordinary users, network operators, and law enforcement with the impossible task of figuring out what happened given little forensic information. Indeed, in this context, the statelessness of modern networks is both a feature and a bug. Instead of reasoning about communication abstractly, routers, switches, hosts, and other network elements, to a first approximation, have a narrow, per-packet view of data transmission. One consequence of this statelessness is the difficulty of securing and auditing networks; a classic example is the complexity of finding the origin of spoofed denial of service attacks [180, 191]. The need for such complex protocols is just one instance of the problems faced in an unauditable Internet.

Instead, to aid in both enforcing network policy in real time and enabling post-mortem analysis, packets must be self-identifying, proving their senders' right to transmit; it should be difficult to mount Sybil attacks [61]. Nevertheless, we would like that packet owners remain anonymous unless there is a legitimate reason to de-anonymize packets (such as a legal inquiry); furthermore, it should be easy for almost anyone to verify packets' validity.

To date, much of the work on authenticating traffic has focused squarely on preventing denial of service attacks attacks. Prior work has aimed to prevent DoS attacks in the network through pushback [87, 126], monitoring [125, 149], and overlay routing [10, 104]; these approaches aim to prevent attacks without requiring per-packet authentication. Alternatively, researchers have proposed using capabilities for authenticating packets to prevent DoS attacks [68, 209, 210] and to authenticate source routes [161]. In these approaches, routers are trusted and the goal is to authorize packets for entry into the network; no effort is made to provide forensic information for later use. Researchers have also considered scenarios in which routers themselves may be malicious [18, 138].

While we have worked on an system that enables network providers to authenticate

the source routes of packets [161], this is only a first step towards properly stamping packets with authentication information. To enforce network policy, there is a need to stamp packets with more than the right to source routes: the right to exist in the network at all [209, 210]. However, such stamping has several consequences—the identity of the packet's owner becomes easily traceable by routers and other empowered parties. Worse, only pre-chosen parties can verify the packet's validity; since end hosts are typically unaware of the networks their packets transit, it is difficult for them to coordinate a priori with transit networks. To address these concerns, we can employ a variety of cryptographic protocols, including ring signatures [172], group signatures [24, 25, 50], transitive signatures [137], identity-based signatures [49, 77, 183], and proxy signatures [16, 123]. Each of these classes of signature schemes will yield, when applied appropriately, a packet authentication scheme with specific guarantees and tradeoffs, though their straightforward application cannot resolve all the engineering challenges.

Beyond simply enabling packet verification, network operators would benefit from mechanisms to glean more information from packet stamps with the proper authorization. While the above signature schemes enable this property, many issues remain: selecting the right cryptographic mechanisms, maintaining confidentiality, anonymity, and authenticity of packets, stamping and checking packets at line rate, dividing authority among appropriate parties, logging large packet traces efficiently for post-mortem analysis, building an incremental deployment path, and identifying legal mechanisms for enforcement.

5.3 Media access control

Even if we are able to protect network users from wide-area misbehavior and selfishness, they may still face contention at access links, particularly at the link layer. Link-layer media access control is an old problem [2]; in its most basic form, multiple nodes wish to share a communication channel. Traditionally, it is a assumed that all parties are cooperative and obey the protocol. In such circumstances, many MAC protocols can achieve high channel utilization.

However, what happens if nodes are malicious or greedy—what if they cheat? In most protocols, such nodes can arbitrarily increase their own throughput to the detriment of others, or can simply cause the network to perform poorly by introducing interference. This problem is worse in wireless networks, where access to the network itself is typically less protected, either for social (public-access wireless) or technical reasons (insecurity, such as with WEP [197]).

Today, 802.11 wireless networks are ubiquitous, and are increasingly deployed by cities for open use. In these deployments, users have an incentive to cheat by disobeying the MAC protocol to increase their throughput. In many ways, this problem is analogous to that of selfish behavior in congestion control, but manifests itself at the link layer.

Research on wireless MAC insecurity has dealt with two distinct environments: infrastructure (access-point) networks and ad-hoc networks. While the techniques used to prevent ad-hoc network misbehavior can be generalized [4, 40, 42, 82, 128, 130], here we focus on infrastructure networks. Specifically, several researchers have analyzed [45, 198] and proposed techniques to detect [46, 157, 168, 175] greedy wireless nodes in 802.11. Unfortunately, such work can only detect and/or prevent misbehavior in restricted scenarios; for example, DOMINO primarily prevents attacks on the 802.11 DCF [168], leaving it trivially vulnerable to tricks such as the de-authentication attack [23].

Existing wireless MAC protocols have insufficient mechanisms to prevent cheating by malicious or greedy nodes. While some malicious attacks may be nearly impossible to defend against (for example, out-of-band, radio interference attacks), there are ways to improve the MAC layer to provide substantially-improved protection.

Prior work has studied techniques to find nodes disobeying backoff timing rules and, similarly, timing transmissions inappropriately. A next step may require modifying the MAC: through appropriately randomized slotting of transmissions, the MAC can help prevent timed attacks. One approach may be to not notify nodes of the transmission schedule until they are scheduled to transmit; this makes anticipating the schedule and stealing the channel difficult.

The problem space of protecting wireless MACs is fraught with issues—many out-of-band attacks are feasible, making it hard both to protect such networks and to model malicious or greedy behavior accurately. Particularly important challenges remain, including modeling the capabilities and motives of a malicious or greedy node, considering tradeoffs in power, efficiency, and security, and analyzing the maximum theoretical throughput of the system assuming selfish behavior.

Bibliography

[1] S. P. Abraham and A. Kumar. A new approach for asynchronous distributed rate control of elastic sessions in integrated packet networks. *IEEE/ACM Trans. Netw.*, 9(1):15–30, 2001.

[2] N. Abramson. The ALOHA system - another alternative for computer communication. In *Proceedings of the AFIPS Fall Joint Computer Conference*, 1970.

[3] A. Aggarwal, S. Savage, and T. Anderson. Understanding the performance of tcp pacing. In *Proceedings of IEEE INFOCOM*, 2000.

[4] G.-S. Ahn, A. T. Campbell, A. Veres, and L.-H. Sun. Swan: service differentiation in stateless wireless ad hoc networks. In *Proceedings of IEEE INFOCOM*, 2002.

[5] Akamai Technologies. Personal communication, June 2007.

[6] A. Akella, R. Karp, C. Papadimitrou, S. Seshan, and S. Shenker. Selfish behavior and stability of the internet: A game-theoretic analysis of TCP. In *Proceedings of ACM SIGCOMM*, 2002.

[7] A. Akella, S. Seshan, and A. Shaikh. An empirical evaluation of wide-area internet bottlenecks. In *SIGMETRICS '03: Proceedings of the 2003 ACM SIGMETRICS international conference on Measurement and modeling of computer systems*, pages 316–317, New York, NY, USA, 2003. ACM.

[8] E. Altman, D. Barman, B. Tuffin, and M. Vojnovic. Parallel TCP sockets: Simple model, throughput and validation. In *Proceedings of IEEE INFOCOM*. IEEE, 2006.

[9] Amazon. Elastic compute cloud. `http://aws.amazon.com/ec2`.

[10] D. G. Andersen. Mayday: Distributed filtering for internet services. In *Proceedings of the 4th USENIX Symposium on Internet Technologies and Systems (USITS)*, Seattle, Washington, Mar. 2003.

[11] T. Anderson, A. Collins, A. Krishnamurthy, and J. Zahorjan. PCP: Efficient endpoint congestion control. 2006.

[12] T. Anderson, S. Shenker, I. Stoica, and D. Wetherall. Design guidelines for robust Internet protocols. In *Proceedings of the 1st ACM Workshop on Hot Topics in Networks (HotNets-I)*, Princeton, NJ, Oct. 2002.

[13] G. Appenzeller, I. Keslassy, and N. McKeown. Sizing router buffers. In *Proceedings of ACM SIGCOMM*, Aug. 2004.

[14] M. Arisoylu, T. Javidi, and R. Cruz. Rate assignment in micro-buffered high speed networks. In *Proceedings of the Allerton Conference on Communication, Control, and Computing*, Sept. 2005.

[15] J. Aspnes, M. Herlihy, and N. Shavit. Counting networks. *Journal of the ACM*, 41(5):1020–1048, 1994.

[16] G. Ateniese and S. Hohenberger. Proxy re-signatures: new definitions, algorithms, and applications. In *Proceedings of ACM CCS*, 2005.

[17] S. Athuraliya, S. H. Low, and D. E. Lapsley. Random early marking. In *QofIS*, pages 43–54, 2000.

[18] I. Avramopoulos and J. Rexford. Stealth probing: Efficient data-plane security for IP routing. In *Proceedings of USENIX*, 2006.

[19] B. Babcock and C. Olston. Distributed top-k monitoring. In *Proceedings of ACM SIGMOD*, 2003.

[20] H. Balakrishnan, R. H. Katz, and V. N. Padmanbhan. The effects of asymmetry on tcp performance. *Mobile Networks and Applications*, 4(3):219–241, 1999.

[21] H. Balakrishnan, H. Rahul, and S. Seshan. An integrated congestion management architecture for Internet hosts. In *Proceedings of the ACM SIGCOMM Conference*, pages 175–187, Cambridge, Massachusetts, Sept. 1999.

[22] D. Bansal, H. Balakrishnan, S. Floyd, and S. Shenker. Dynamic behavior of slowly-responsive congestion control algorithms. In *SIGCOMM '01: Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 263–274, New York, NY, USA, 2001. ACM.

[23] J. Bellardo and S. Savage. 802.11 denial-of-service attacks: Real vulnerabilities and practical solutions. In *Proceedings of USENIX Security*, 2003.

[24] M. Bellare, D. Micciancio, and B. Warinschi. Foundations of group signatures: formal definition, simplified requirements and a construction based on trapdoor permutations. In *Proceedings of EUROCRYPT*, 2003.

[25] M. Bellare, H. Shi, and C. Zhang. Foundations of group signatures: The case of dynamic groups. In *Proceedings of CT-RSA*, 2005.

[26] J. C. Bennet and H. Zhang. Hierarchical packet fair queueing algorithms. In *Proceedings of ACM SIGCOMM*, 1996.

[27] J. C. R. Bennett and H. Zhang. Wf2q: Worst-case fair weighted fair queueing. In *Proceedings of IEEE INFOCOM*, 1996.

[28] D. Bertsekas and R. Gallager. *Data Networks*. Prentice Hall, 1987.

[29] S. Bhadra and S. Shakkottai. Looking at large networks: Coding vs. queueing. In *Proceedings of IEEE INFOCOM*, 2006.

[30] S. Bhatnagar and B. Nath. Distributed admission control to support guaranteed services in core-stateless networks. In *Proceedings of IEEE INFOCOM*, 2003.

[31] E. Blanton and M. Allman. On making tcp more robust to packet reordering. *SIGCOMM Comput. Commun. Rev.*, 32(1):20–30, 2002.

[32] T. Bonald, M. Feuillet, and A. Proutiere. Is the "Law of the Jungle" Sustainable for the Internet? In *Proceedings of IEEE INFOCOM*, 2009.

[33] R. Braden, ed., L. Zhang, S. Berson, S. Herzog, and S. Jamin. Resource reSerVation Protocol (RSVP) – version 1 functional specification. RFC 2205, IETF, Sept. 1997.

[34] L. S. Brakmo, S. W. O'Malley, and L. L. Peterson. Tcp vegas: New techniques for congestion detection and avoidance. In *Proceedings of ACM SIGCOMM*, 1994.

[35] L. Breslau, E. W. Knightly, S. Shenker, I. Stoica, and H. Zhang. Endpoint admission control: architectural issues and performance. In *SIGCOMM '00: Proceedings of the conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, pages 57–69, New York, NY, USA, 2000. ACM.

[36] B. Briscoe. Flow rate fairness: Dismantling a religion. Internet Draft, IETF, 2006.

[37] B. Briscoe. Flow rate fairness: Dismantling a religion. *ACM SIGCOMM CCR*, 37(2), 2007.

[38] B. Briscoe, A. Jacquet, C. Di Cairano-Gilfedder, A. Salvatori, A. Soppera, and M. Koyabe. Policing congestion response in an internetwork using re-feedback. In *Proceedings of ACM SIGCOMM*, 2005.

[39] B. Briscoe, A. Jacquet, A. Salvatori, and M. Koyabe. Re-ECN: Adding Accountability for Causing Congestion to TCP/IP. Internet Draft, IETF, 2007.

[40] S. Buchegger and J.-Y. L. Boudec. Performance analysis of the CONFIDANT protocol. In *Proceedings of the third ACM International Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc '02)*, pages 226–236. ACM Press, 2002.

[41] bugtraq archives. http://cert.uni-stuttgart.de/archive/bugtraq/1999/11/msg00036.html, 1999.

[42] L. Buttyan and J. Hubaux. Stimulating cooperation in self-organizing mobile ad hoc networks, 2001.

[43] J. Byers, J. Considine, M. Mitzenmacher, and S. Rost. Informed content delivery across adaptive overlay networks. In *Proceedings of ACM SIGCOMM*, 2002.

[44] J. W. Byers, M. Luby, M. Mitzenmacher, and A. Rege. A digital fountain approach to reliable distribution of bulk data. In *Proceedings of the ACM SIGCOMM '98 conference on Applications, technologies, architectures, and protocols for computer communication*, pages 56–67. ACM Press, 1998.

[45] M. Cagalj, S. Ganeriwal, I. Aad, and J. P. Hubaux. On selfish behavior in CSMA/CA networks. In *Proceedings of IEEE INFOCOM*, 2005.

[46] A. A. Cardenas, S. Radosavac, and J. S. Baras. Detection and prevention of MAC layer misbehavior in ad hoc networks. In *Proceedings of ACM SASN*, 2004.

[47] M. Carter et al. Effects of catrastrophic events on transportation system management and operations: Howard street tunnel fire. `http://www.its.dot.gov/JPODOCS/REPTS_TE/13754.html`, 2002.

[48] V. Cerf and R. Kahn. A protocol for packet network intercommunication. *IEEE Transactions on Communications*, 22(5):637–648, May 1974.

[49] J. Cha and J. Cheon. An identity-based signature from gap diffie-hellman groups. In *Proceedings of PKC*, 2002.

[50] D. Chaum and E. van Heyst. Group signatures. In *Proceedings of EUROCRYPT*, 1991.

[51] K. Cho, K. Fukuda, H. Esaki, and A. Kato. The impact and implications of the growth in residential user-to-user traffic. In *Proceedings of ACM SIGCOMM*, 2006.

[52] D. Clark. The design philosophy of the darpa internet protocols. In *Symposium proceedings on Communications architectures and protocols*, pages 106–114. ACM Press, 1988.

[53] D. D. Clark, S. Shenker, and L. Zhang. Supporting real-time applications in an integrated services packet network: architecture and mechanism. In *Conference proceedings on Communications architectures and protocols*, pages 14–26. ACM Press, 1992.

[54] D. D. Clark, J. Wroclawski, K. R. Sollins, and R. Braden. Tussle in cyberspace: Defining tomorrow's Internet. In *Proc. ACM SIGCOMM*, 2002.

[55] R. Cocchi, S. Shenker, D. Estrin, and L. Zhang. Pricing in computer networks: motivation, formulation, and example. *IEEE/ACM Transactions on Networking (TON)*, 1(6):614–627, 1993.

[56] R. L. Cruz. Quality of service guarantees in virtual circuit switched networks. *IEEE Journal on Selected Areas in Communications*, 13:1048–1056, 1995.

[57] D. W. Davies. The control of congestion in packet switching networks. In *Proceedings of ACM Problems in the optimizations of data communications systems*, 1971.

[58] A. Demers, D. Greene, C. Hauser, W. Irish, J. Larson, S. Shenker, H. Sturgis, D. Swinehart, and D. Terry. Epidemic algorithms for replicated database maintenance. In *Proceedings of ACM PODC*, 1987.

[59] A. Demers, S. Keshav, and S. Shenker. Analysis and simulation of a fair queueing algorithm. In *Proceedings of ACM SIGCOMM*, 1989.

[60] M. Dilman and D. Raz. Efficient reactive monitoring. In *Proceedings of IEEE INFOCOM*, 2001.

[61] J. R. Douceur. The sybil attack. In *Proceedings of IPTPS*, 2002.

[62] N. G. Duffield, P. Goyal, A. Greenberg, P. Mishra, K. K. Ramakrishnan, and J. E. van der Merive. A flexible model for resource management in virtual private networks. In *SIGCOMM '99: Proceedings of the conference on Applications, technologies, architectures, and protocols for computer communication*, pages 95–108, New York, NY, USA, 1999. ACM.

[63] N. Dukkipati and N. McKeown. Why flow-completion time is the right metric for congestion control. *ACM SIGCOMM CCR*, 36(1), 2006.

[64] D. Ely, N. Spring, D. Wetherall, S. Savage, and T. Anderson. Robust congestion signaling. In *Proceedings of IEEE ICNP*, 2001.

[65] M. Enachescu, Y. Ganjali, A. Goel, N. Mckeown, and T. Roughgarden. Routers with very small buffers. In *Proceedings of IEEE INFOCOM*, 2006.

[66] C. Estan, S. Savage, and G. Varghese. Automatically inferring patterns of resource consumption in network traffic. In *Proceedings of ACM SIGCOMM*, 2002.

[67] C. Estan and G. Varghese. New directions in traffic measurement and accounting. In *Proceedings of ACM SIGCOMM*, 2002.

[68] D. Estrin, J. C. Mogul, and G. Tsudik. Visa protocols for controlling interorganizational datagram flow. *IEEE J. SAC*, 7(4):486–498, May 1989.

[69] W. Feng, K. Shin, D. Kandlur, and D. Saha. The blue active queue management algorithms. *IEEE/ACM Transactions on Networking*, 10(4), 2002.

[70] S. Floyd. HighSpeed TCP for large congestion windows. RFC 3649, IETF, 2003.

[71] S. Floyd, M. Allman, A. Jain, and P. Sarolahti. Quick-start for TCP and IP. Internet Draft, IETF, 2006.

[72] S. Floyd and K. Fall. Promoting the use of end-to-end congestion control in the internet. *IEEE/ACM Transactions on Networking (TON)*, 7(4):458–472, 1999.

[73] S. Floyd, R. Gummadi, and S. Shenker. Adaptive red: An algorithm for increasing the robustness of red, 2001. unpublished.

[74] S. Floyd, M. Handley, J. Padhye, and J. Widmer. Equation-based congestion control for unicast applications. In *Proceedings of ACM SIGCOMM*, pages 43–56, 2000.

[75] S. Floyd and V. Jacobson. Random early detection gateways for congestion avoidance. *IEEE/ACM Transactions on Networking (TON)*, 1(4):397–413, 1993.

[76] S. Floyd and V. Jacobson. Link-sharing and resource management models for packet networks. *IEEE/ACM Transactions on Networking*, 3(4):365–386, 1995.

[77] C. Gentry and A. Silverberg. Hierarchical id-based cryptography. In *Proceedings of ASIACRYPT*, 2002.

[78] A. Goel, A. Meyerson, and S. Plotkin. Distributed admission control, scheduling, and routing with stale information. In *SODA '01: Proceedings of the twelfth annual ACM-SIAM symposium on Discrete algorithms*, pages 611–619, 2001.

[79] I. Gupta, A.-M. Kermarrec, and A. J. Ganesh. Efficient epidemic-style protocols for reliable and scalable multi cast. In *Proceedings of IEEE SRDS*, 2002.

[80] S. Ha, I. Rhee, and L. Xu. CUBIC: a new TCP-friendly high-speed TCP variant. *SIGOPS Operating Systems Review*, 42(5):64–74, 2008.

[81] D. Hinchcliffe. 2007: The year enterprises open thier SOAs to the Internet? *Enterprise Web 2.0*, Jan. 2007.

[82] Y.-C. Hu, A. Perrig, and D. B. Johnson. Ariadne: a secure on-demand routing protocol for ad hoc networks. In *Proc. ACM MOBICOM*, 2002.

[83] X. Hua, D. Nguyen, B. Raghavan, J. Arsuaga, and M. Vazquez. Random State Transitions of Knots: A First Step Towards Modeling Unknotting by Type II Topoisomerases. *Topology and its Applications*, 154(7), 2007.

[84] M. Huang. Planetlab bandwidth limits. `http://www.planet-lab.org/doc/BandwidthLimits.php`.

[85] Institute of Electronic and Electrical Engineers (IEEE). Wireless medium access control (MAC) and physical layer (PHY) specifications. Standard 802.11, 1999.

[86] Institute of Electronic and Electrical Engineers (IEEE). Carrier sense multiple access with collision detection (CSMA/CD) access method and physical specifications. Standard 802.3, 2002.

[87] J. Ioannidis and S. M. Bellovin. Implementing pushback: Router-based defense against DDoS attacks. In *Proceedings of ISOC NDSS*, 2002.

[88] V. Jacobson. Congestion avoidance and control. In *Proceedings of ACM SIGCOMM*, 1988.

[89] J. M. Jaffe. Bottleneck flow control. *IEEE Transactions on Communcations*, 7(29), July 1980.

[90] A. Jain, J. M. Hellerstein, S. Ratnasamy, and D. Wetherall. A wakeup call for internet monitoring systems: The case for distributed triggers. In *Proceedings of Hotnets-III*, 2004.

[91] R. Jain, D. M. Chiu, and W. Hawe. A quantitative measure of fairness and discrimination for resource allocation in shared computer systems. Technical report, DEC Research Report TR-301, Sept. 1984.

[92] S. Jamin, P. Danzig, S. Shenker, and L. Zhang. A measurement-based admission control algorithm for integrated services packet networks. In *Proceedings of ACM SIGCOMM*, 1995.

[93] C. Jin, D. X. Wei, and S. H. Low. FAST TCP: motivation, architecture, algorithms, performance. In *Proceedings of IEEE INFOCOM*, 2004.

[94] B. Kantor and P. Lapsley. Network news transfer protocol. RFC 977, IETF, Feb. 1986.

[95] R. M. Karp. Fair bandwidth allocation without per-flow state. *LNCS, Essays in Memory of Shimon Even*, 3895, 2006.

[96] R. M. Karp, E. Koutsoupias, C. H. Papadimitriou, and S. Shenker. Optimization problems in congestion control. In *FOCS*, pages 66–74, 2000.

[97] D. Katabi, M. Handley, and C. Rohrs. Congestion control for high bandwidth-delay product networks. In *Proceedings of the 2002 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 89–102. ACM Press, 2002.

[98] M. Kaufman. MAE-West congestion, NANOG mailing list. `http://www.irbs.net/internet/nanog/9603/0200.html`, 1996.

[99] F. Kelly. Charging and rate control for elastic traffic. *European Transactions on Telecommunications*, 8:33–37, 1997.

[100] F. P. Kelly, P. B. Key, and S. Zachary. Distributed admission control. *IEEE Journal on Selected Areas in Communications*, 18(12), Dec. 2000.

[101] T. Kelly. Scalable TCP: improving performance in highspeed wide area networks. *SIGCOMM Computer Communication Review*, 33(2):83–91, 2003.

[102] T. Kelly, S. Floyd, and S. Shenker. Patterns of conjestion collapse, 2003. unpublished.

[103] D. Kempe, A. Dobra, and J. Gehrke. Gossip-based computation of aggregate information. In *Proceedings of IEEE FOCS*, Oct. 2003.

[104] A. Keromytis, V. Misra, and D. Rubenstein. Sos: Secure overlay services. In *Proceedings of ACM SIGCOMM*, 2002.

[105] E. Kiltz, A. Mityagin, S. Panjwani, and B. Raghavan. Append-Only Signatures. In *Proceedings of ICALP*, 2005.

[106] D. E. Knuth. *The Art of Computer Programming*. Addison-Wesley Longman Publishing Co., Inc., 1998.

[107] E. Kohler, M. Handley, S. Floyd, and J. Padhye. Datagram congestion control protocol (DCCP). Internet Draft, IETF, June 2002. `draft-kohler-dcp-04.txt` (work in progress).

[108] A. Kumar, R. Rastogi, A. Siberschatz, and B. Yener. Algorithms for provisioning virtual private networks in the hose model. *IEEE/ACM Transactions on Networking*, 10(4):565–578, 2002.

[109] A. Kumar, J. Xu, L. Li, J. Wang, and O. Spatschek. Space-code bloom filter for efficient per-flow traffic measurement. In *Proceedings of IEEE INFOCOM*, Mar. 2004.

[110] A. Kuzmanovic and E. W. Knightly. Low-rate TCP-targeted denial of service attacks: the shrew vs. the mice and elephants. In *Proceedings of ACM SIGCOMM*, 2003.

[111] L. Li, D. Alderson, W. Willinger, and J. Doyle. A first-principles approach to understanding the internet's router-le vel topology. In *Proceedings of ACM SIGCOMM*, 2004.

[112] J. Liang, S. Y. Ko, I. Gupta, and K. Nahrstedt. MON: On-demand overlays for distributed system management. In *Proceedings of USENIX WORLDS*, 2005.

[113] D. Lin and R. Morris. Dynamics of random early detection. In *Proceedings of ACM SIGCOMM*, 1997.

[114] S. Lin, D. J. C. Jr., and M. J. Miller. Automatic-repeat-request error-control schemes. *IEEE Communications Magazine*, 22(12), 1984.

[115] Z. Liu and D. Towsley. Burst reduction properties of rate-control throttles: downstream queue behavior. *IEEE/ACM Trans. Netw.*, 3(1):82–90, 1995.

[116] L. López, G. del Rey Almansa, S. Paquelet, and A. Fernández. A mathematical model for the tcp tragedy of the commons. *Theor. Comput. Sci.*, 343(1-2):4–26, 2005.

[117] L. López and A. Fernández. A game theoretic analysis of protocols based on fountain codes. In *Proceedings of IEEE ISCC*, 2005.

[118] L. López, A. Fernández, and V. Cholvi. A game theoretic comparison of tcp and digital fountain based protocols. *Comput. Netw.*, 51(12):3413–3426, 2007.

[119] M. Lottor. Internet growth (1981-1991). RFC 1296, IETF, Jan. 1992.

[120] S. H. Low, F. Paganini, J. Wang, S. Adlakha, and J. C. Doyle. Dynamics of TCP/AQM and a scalable control. In *Proceedings of IEEE INFOCOM*, 2002.

[121] M. Luby. Lt codes. In *Proceedings of IEEE FOCS*, 2002.

[122] H. Lundqvist and G. Karlsson. Tcp with end-to-end forward error correction. In *Proceedings of International Zurich Seminar on Communications*, 2004.

[123] G. B. M. Blaze and M. Strauss. Divertible protocols and atomic proxy cryptography. In *Proceedings of EUROCRYPT*, 1998.

[124] J. Ma, K. Levchenko, C. Kriebich, S. Savage, and G. M. Voelker. Automated protocol inference: Unexpected means of identifying protocols. In *Proceedings of USENIX/ACM IMC*, 2006.

[125] S. Machiraju, M. Seshadri, and I. Stoica. A scalable and robust solution for bandwidth allocation. Technical Report UCB/CSD-02-1176, EECS Department, University of California, Berkeley, 2002.

[126] R. Mahajan, S. M. Bellovin, S. Floyd, J. Ioannidis, V. Paxson, and S. Shenker. Controlling high bandwidth aggregates in the network. *ACM SIGCOMM CCR*, 32(3):62–73, 2002.

[127] R. Mahajan, S. Floyd, and D. Wetherall. Controlling high-bandwidth flows at the congested router. In *Proceedings of IEEE ICNP*, 2001.

[128] R. Mahajan, M. Rodrig, D. Wetherall, and J. Zahorjan. Sustaining cooperation in multi-hop wireless networks. 2005.

[129] A. Manjhi, V. Shkapenyuk, K. Dhamdhere, and C. Olston. Finding (recently) frequent items in distributed data streams. In *Proceedings of IEEE ICDE*, Apr. 2005.

[130] S. Marti, T. J. Giuli, K. Lai, and M. Baker. Mitigating routing misbehavior in mobile ad hoc networks. In *Proceedings of the sixth ACM International Conference on Mobile Computing and Networking (MobiCom '00)*, pages 255–265. ACM Press, 2000.

[131] M. Mathis and J. Mahdavi. Forward acknowledgement: refining tcp congestion control. In *SIGCOMM '96: Conference proceedings on Applications, technologies, architectures, and protocols for computer communications*, pages 281–291, 1996.

[132] M. Mathis, J. Mahdavi, S. Floyd, and A. Romanow. TCP selective acknowledgment options. RFC 2018, IETF, Oct. 1996.

[133] P. Maymounkov. Online codes. 2002.

[134] P. Maymounkov and D. Mazieres. Rateless codes and big downloads. In *Proceedings of IPTPS*, 2003.

[135] M. K. McKusick, K. Bostic, M. J. Karels, and J. S. Quarterman. *The Design and Implementation of the 4.4BSD Operating System*. Addison Wesley, Reading, Massachusetts, Apr. 1996.

[136] M. Mehyar, D. Spanos, J. Pongsajapan, S. H. Low, and R. M. Murray. Asynchronous distributed averaging on communication networks. *IEEE/ACM Trans. Netw.*, 15(3):512–520, 2007.

[137] S. Micali and R. L. Rivest. Transitive signature schemes. In *Proceedings of CT-RSA*, pages 236–243, 2002.

[138] A. T. Mizrak, Y.-C. Cheng, K. Marzullo, and S. Savage. Fatih: Detecting and isolating malicious routers. In *Proceedings of IEEE DSN*, 2005.

[139] J. C. Mogul and G. Minshall. Rethinking the tcp nagle algorithm. *SIGCOMM Comput. Commun. Rev.*, 31(1):6–20, 2001.

[140] A. Mondal and A. Kuzmanovic. When tcp friendliness becomes harmful. In *Proceedings of IEEE INFOCOM*, 2007.

[141] R. Morris. TCP behavior with many flows. In *Proceedings of IEEE ICNP*, 1997.

[142] J. Nagle. Congestion control in IP/TCP internetworks. RFC 896, IETF, Jan. 1984.

[143] J. B. Nagle. On packet switches with infinite storage. *IEEE Transactions on Communications*, COM-35(4), Apr. 1987.

[144] A. Nucci, A. Sridharan, and N. Taft. The problem of synthetically generating IP traffic matricies: Initi al recommendations. *ACM SIGCOMM CCR*, 35(3), July 2005.

[145] A. Odlyzko. Data networks are lightly utilized, and will stay that way. *Review of Network Economics*, 2(3), 2003.

[146] A. M. Odlyzko. Internet pricing and the history of communications. *Computer Networks*, 36:493–517, 2001.

[147] Packeteer. `http://www.packeteer.com`.

[148] R. Pan, L. Breslau, B. Prabhakar, and S. Shenker. Approximate fairness through differential dropping. *ACM SIGCOMM CCR*, 33(2), 2003.

[149] R. Pan, B. Prabhakar, and K. Psounis. CHOKe - A stateless queue management scheme for approximating fair bandwidth allocation. In *Proceedings of IEEE INFOCOM*, 2000.

[150] A. Parekh and R. Gallager. A generalized processor sharing approach to flow control in integrated services networks: The single node case. *IEEE/ACM Transactions on Networking*, 1, 1993.

[151] C. Partridge, D. Cousins, A. W. Jackson, R. Krishnan, T. Saxena, and W. T. Strayer. Using signal processing to analyze wireless data traffic. In *Proceedings of the ACM Workshop on Wireless Security*, Sept. 2002.

[152] J. Postel. Internet Protocol. RFC 791, IETF, Sept. 1981.

[153] J. Postel. Transmission control protocol. RFC 793, IETF, Sept. 1981.

[154] J. Postel. Simple mail transfer protocol. RFC 821, IETF, Aug. 1982.

[155] J. Postel and J. Reynolds. File transfer protocol (FTP). Technical report, IETF, Oct. 1985. RFC 959.

[156] J. Postel and J. K. Reynolds. TELNET protocol specification. RFC 854, IETF, May 1983.

[157] S. Radosavac, J. S. Baras, and I. Koutsopoulos. A framework for MAC protocol misbehavior detection in wireless netwo rks. In *Proceedings of ACM WiSe*, 2005.

[158] B. Raghavan, T. Kohno, A. C. Snoeren, and D. Wetherall. Enlisting ISPs to Improve Online Privacy: IP Address Mixing by Default. In *Proceedings of the Privacy Enhancing Technologies Symposium*, 2009.

[159] B. Raghavan, S. Panjwani, and A. Mityagin. Analysis of the SPV Secure Routing Protocol: Weaknesses and Lessons. *ACM SIGCOMM Computer Communication Review*, Apr. 2007.

[160] B. Raghavan and A. C. Snoeren. Priority Forwarding in Ad Hoc Networks with Self-Interested Parties. In *Proceedings of the Workshop on Economics of Peer-to-Peer Systems*, 2003.

[161] B. Raghavan and A. C. Snoeren. A System for Authenticated Policy-Compliant Routing. In *Proceedings of ACM SIGCOMM*, 2004.

[162] B. Raghavan and A. C. Snoeren. Decongestion Control. In *Proceedings of ACM SIGCOMM Workshop on Hot Topics in Networks*, 2006.

[163] B. Raghavan, P. Verkaik, and A. Snoeren. Secure and Policy-Compliant Source Routing. *IEEE/ACM Transactions on Networking*, Aug. 2009.

[164] B. Raghavan, K. Vishwanath, S. Ramabhadran, K. Yocum, and A. C. Snoeren. Cloud Control with Distributed Rate Limiting. In *Proceedings of ACM SIGCOMM*, 2007.

[165] S. Ramabhadran and G. Varghese. Efficient implementation of a statistics counter architecture. In *Proceedings of ACM SIGMETRICS*, 2003.

[166] K. Ramakrishnan and S. Floyd. A Proposal to add Explicit Congestion Notification (ECN) to IP. RFC 2481, IETF, Jan. 1999.

[167] K. K. Ramakrishnan and R. Jain. A binary feedback scheme for congestion avoidance in computer networks. *ACM Transactions on Computer Systems (TOCS)*, 8(2):158–181, 1990.

[168] M. Raya, J.-P. Hubaux, and I. Aad. Domino: A system to detect greedy behavior in ieee 802.11 hotspots. In *Proceedings of USENIX/ACM MobiSys*, 2004.

[169] D. D. Redell, Y. K. Dalal, T. R. Horsley, H. C. Lauer, W. C. Lynch, P. R. McJones, H. G. Murray, and S. C. Purcell. Pilot: an operating system for a personal computer. *Commun. ACM*, 23(2):81–92, 1980.

[170] J. Reif and P. Spirakis. Real time resource allocation in distributed systems. In *Proceedings of ACM PODC*, Aug. 1982.

[171] J. Rexford, F. Bonomi, A. G. Greenberg, and A. Wong. A scalable architecture for fair leaky-bucket shaping. In *INFOCOM*, pages 1054–1062, 1997.

[172] R. L. Rivest, A. Shamir, and Y. Tauman. How to leak a secret. In *Proceedings of ASIACRYPT*, 2001.

[173] L. Rizzo. Effective erasure codes for reliable computer communication protocols. *SIGCOMM CCR*, 27(2):24–36, 1997.

[174] J. A. Rochlis and M. W. Eichin. With microscope and tweezers: the worm from MIT's perspective. *Communications of the ACM*, 32(6):689–698, 1989.

[175] Y. Rong, S.-K. Lee, and H.-A. Choi. Detecting stations cheating on backoff rules in 802.11 networks using sequential analysis. In *Proceedings of IEEE INFOCOM*, 2006.

[176] S. Sahu, P. Nain, C. Diot, V. Firoiu, and D. Towsley. On achievable service differentiation with token bucket marking for tcp. *SIGMETRICS Perform. Eval. Rev.*, 28(1):23–33, 2000.

[177] J. H. Saltzer, D. P. Reed, and D. D. Clark. End-to-end arguments in system design. *ACM Transactions on Computer Systems*, 2(4):277–288, Nov. 1984.

[178] P. Sarolahti, M. Allman, and S. Floyd. Determining an appropriate sending rate over an underutilized network path. *Comput. Netw.*, 51(7):1815–1832, 2007.

[179] S. Savage, N. Cardwell, D. Wetherall, and T. Anderson. Tcp congestion control with a misbehaving receiver. *ACM SIGCOMM CCR*, 29(5):71–78, 1999.

[180] S. Savage, D. Wetherall, A. Karlin, and T. Anderson. Practical network support for IP traceback. In *Proceedings of ACM SIGCOMM*, 2000.

[181] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson. RTP: A transport protocol for real-time applications. RFC 1889, IETF, Jan. 1996.

[182] D. Shah, S. Iyer, B. Prabhakar, and N. McKeown. Maintaining statistics counters in router line cards. *IEEE Micro*, 22(1):76–81, 2002.

[183] A. Shamir. Identity-based cryptosystems and signature schemes. In *Proceedings of CRYPTO*, 1984.

[184] V. Sharma, S. Kalyanaraman, K. Kar, K. K. Ramakrishnan, and V. Subramanian. Mplot: A transport protocol exploiting multipath diversity using erasure codes. In *INFOCOM*, pages 121–125. IEEE, 2008.

[185] N. Shavit and A. Zemach. Diffracting trees. *ACM TOCS*, 14(4):385–428, 1996.

[186] S. Shenker. Making greed work in networks: a game-theoretic analysis of switch service disciplines. In *Proceedings of ACM SIGCOMM*, 1994.

[187] R. Sherwood, B. Bhattacharjee, and R. Braud. Misbehaving tcp receivers can cause internet-wide congestion collapse. In *Proceedings of ACM CCS*, 2005.

[188] R. N. Shorten and D. J. Leith. On queue provisioning, network efficiency and the transmission control protocol. *IEEE/ACM Trans. Netw.*, 15(4):866–877, 2007.

[189] M. Shreedhar and G. Varghese. Efficient fair queuing using deficit round robin. In *Proceedings of ACM SIGCOMM*, 1995.

[190] J. Shu and P. Varaiya. Pricing network services. In *To appear in Proceedings of IEEE INFOCOM 2003*. IEEE, 2003.

[191] A. C. Snoeren, C. Partridge, L. A. Sanchez, C. E. Jones, F. Tchakountio, S. T. Kent, and W. T. Strayer. Hash-based IP traceback. In *Proceedings of ACM SIGCOMM*, 2001.

[192] A. C. Snoeren and B. Raghavan. Decoupling Policy from Mechanism in Internet Routing. In *Proceedings of ACM SIGCOMM Workshop on Hot Topics in Networks*, 2003.

[193] B. Spirrison. Chicago spreads welcome mat for citywide Wi-Fi. *Chicago Sun-Times*, October 2, 2006.

[194] R. Stanojevic and R. Shorten. Fully decentralized emulation of best-effort and processor sharing queues. In *Proceedings of ACM SIGMETRICS*, 2008.

[195] W. R. Stevens. TCP tcp slow start, congestion avoidance, fast retransmit, and fast recovery algorithms. RFC 2001, IETF, Jan. 1997.

[196] I. Stoica, S. Shenker, and H. Zhang. Core-stateless fair queueing: a scalable architecture to approximate fair bandwidth allocations in high speed networks. In *Proceedings of ACM SIGCOMM*, 1998.

[197] A. Stubblefield, J. Ioannidis, and A. Rubin. Using the Fluhrer, Mantin, and Shamir attack to break WEP. In *Proceedings of ISOC NDSS*, 2002.

[198] G. Tan and J. Guttag. The 802.11 MAC protocol leads to inefficient equilibria. In *Proceedings of IEEE INFOCOM*, 2005.

[199] K. Tan and J. Song. A compound TCP approach for high-speed and long distance networks. In *Proceedings of IEEE INFOCOM*, 2006.

[200] A. Vahdat, K. Yocum, K. Walsh, P. Mahadevan, D. Kostić, J. Chase, and D. Becker. Scalability and accuracy in a large-scale network emulator. In *Proceedings of USENIX OSDI*, Dec. 2002.

[201] Verizon Business Group. The Taiwan earthquake: A case study in network disaster recovery. `http://www.verizonbusiness.com/us/resources/resilience/taiwanearthquake.pdf`, 2007.

[202] K. Vishwanath and A. Vahdat. Realistic and responsive network traffic generation. In *Proceedings of ACM SIGCOMM*, 2006.

[203] W. Vogels, R. van Renesse, and K. Birman. The power of epidemics: robust communication for large-scale distributed systems. *ACM SIGCOMM CCR*, 33(1):131–135, 2003.

[204] M. Walfish, H. Balakrishnan, D. Karger, and S. Shenker. DoS: Fighting fire with fire. In *Proceedings of ACM SIGCOMM Workshop on Hot Topics in Networks*, 2005.

[205] L. Wang, K. Park, R. Pang, V. S. Pai, and L. Peterson. Reliability and security in the CoDeeN content distribution network. In *Proceedings of USENIX*, 2004.

[206] R. Wattenhofer and P. Widmayer. An inherent bottleneck in distributed counting. In *Proceedings of ACM PODC*, Aug. 1997.

[207] D. X. Wei, C. Jin, S. H. Low, and S. Hegde. Fast tcp: motivation, architecture, algorithms, performance. *IEEE/ACM Trans. Netw.*, 14(6):1246–1259, 2006.

[208] L. Xu, K. Harfoush, and I. Rhee. Binary increase congestion control for fast long-distance networks. In *Proceedings of IEEE INFOCOM*, 2004.

[209] A. Yaar, A. Perrig, and D. Song. Siff: A stateless internet flow filter to mitigate ddos flooding attacks. In *Proceedings of IEEE Security and Privacy*, 2004.

[210] X. Yang, D. Wetherall, and T. Anderson. A dos-limiting network architecture. In *Proceedings of ACM SIGCOMM*, 2005.

[211] H. Zhang, D. Towsley, and W. Gong. TCP connection game: A study on the selfish behavior of TCP users. In *Proceedings of IEEE ICNP*, 2005.

[212] M. Zhang, B. Karp, S. Floyd, and L. Peterson. RR-TCP: A Reordering-Robust TCP with DSACK. *Network Protocols, IEEE International Conference on*, 0:95, 2003.

[213] Y. Zhang, Z. M. Mao, and J. Wang. Low-rate TCP-targeted DoS attack disrupts Internet routing. In *Proceedings of ISOC NDSS*, 2007.

[214] Z.-L. Zhang, Z. Duan, L. Gao, and Y. T. Hou. Decoupling qos control from core routers: a novel bandwidth broker architecture for scalable support of guaranteed services. *SIGCOMM Comput. Commun. Rev.*, 30(4):71–83, 2000.

[215] Z.-L. Zhang, Z. Duan, and Y. T. Hou. On scalable design of bandwidth brokers. *IEICE Trans. Commun.*, E84-B(8), 2001.

[216] T. Zhao and V. Karamcheti. Enforcing resource sharing agreements among distributed server clusters. In *Proceedings of IEEE IPDPS*, Apr. 2002.

[217] H. Zimmerman. OSI reference model — the ISO model of architecture for open system intereconnection. *IEEE Transactions on Communications*, COM-28(4):425–432, Apr. 1980.