Rethinking Traffic Management: Design of Optimizable Networks

Jiayue He

A Dissertation
Presented to the Faculty
of Princeton University
in Candidacy for the Degree
of Doctor of Philosophy

Recommended for Acceptance
by the Department of
Electrical Engineering

June 2008

Abstract

Traffic management refers to controlling how much traffic traverses each path in a network. On the Internet today, end hosts run congestion control to adapt sending rates, routers route traffic on shortest paths based on link weights, and operators tune link weights to direct traffic away from heavily-loaded links. This dissertation performs a top-down redesign of traffic management to support diverse application requirements, leveraging emerging technology trends in network virtualization and multipath routing.

We begin by analyzing, then redesigning today's traffic-management system. In the 'bottom-up' approach, we study the interaction of congestion control and traffic engineering using established optimization models. We find congestion control and traffic-engineering interact in a stable, though not always efficient manner. Efficiency can be improved by tuning the operator's traffic-engineering function, but at the cost of robustness.

In the 'top-down' approach, we propose a new objective function that captures the goals of both end users and network operators. Next, using various optimization decomposition techniques, we generate four distributed algorithms that divide traffic over multiple paths based on feedback from the network links. These distributed algorithms are provably stable and optimal, but can converge slowly and are sensitive to tuning parameters. Finally, combining the best features of these distributed algorithms, we construct TRUMP: TRaffic-management Using Multipath Protocol. TRUMP converges quickly and contains a single easy to tune parameter. Packet-level simulations show TRUMP behaves well with realistic topologies, feedback delays, capacities, and traffic loads.

Since applications today may have different performance objectives, we next redesign traffic management to handle multiple traffic classes. A natural objective for an ISP is to maximize aggregate performance objectives across multiple traffic classes.

Decomposing the ISP's problem leads to a stable and optimal solution where each traffic class optimizes according to its own performance objective, with an algorithm to dynamically allocate bandwidth shares. The distributed protocols can be implemented using DaVinci: Dynamically Adaptive VIrtual Networks for a Customized Internet. In DaVinci, each virtual network runs traffic-management protocols optimized for a traffic class, and link bandwidth is dynamically allocated between virtual networks through separate queues.

Overall, we show that using optimization theory as a foundation, simulations as a building block, and engineering intuition as a guide can be a principled approach to architecture and protocol design.

Acknowledgements

There are so many different ways to be connected to people. There are the people you feel this unspoken connection to, even though there's not even a word for it. There's the people who you've known forever who know you in this way that other people can't because they've seen you change. — Angela Chase

The wondrously diverse connections I have made with mentors, collaborators and friends have immensely enriched my graduate student life. In particular, I thank:

- Professors Mung Chiang and Jennifer Rexford, my advisors, for introducing me to the world of network research. Thanks to Mung for encouraging me to fearlessly pursue research ideas. Thanks to Jennifer for mentoring me selflessly in all aspects of being a researcher: writing, presenting, networking, teaching, and managing. Special thanks to Jen's partner for helping me with the non-academic job search.

- Prof. Robert Calderbank, Prof. Larry Peterson, and Prof. Kevin Tang for serving on my thesis committee. Kevin's attention to detail has made this thesis more complete than it would have been otherwise.

- Ma'ayan Bresler, Martin Suchara, Rui Zhang-Shen, Ying Li, Mike Lee, and Umar Javed for helping with the simulation results in Chapters 4 and 5. Special thanks to Rui for her friendship and support through my final stretch of deadlines. Thanks to Tanya Monga for her detailed edits to the earlier chapters and Vytautas Valancius for his edits to the epilogue.

- Augustin Chaintreau and Christophe Diot for giving me the opportunity to intern at Thomson Research Labs in Paris. Thanks to all the interns there for making it an unforgettable summer, especially Haakon for introducing me to the art of wine with dinner.

- Graduate Student Government, Graduate Engineering Ambassadors and the rock climbing crew for keeping my life in graduate school sane! Special thanks to Jeff for supplying me with desserts in my hours of desperation; my belay partner Shannon for her encouragement in and out of the climbing gym.

- Princeton University for letting me live in 11 Dickinson Street for my entire graduate career. House mates Sambuddho, Leo, Eddie and Kevin created an eccentric and fun environment in my first years. Then Arvid, Taylor, Arie and Jun kept up the warm environment.

- My family for their support and encouragement. Thanks dad for always asking me how many papers I have published since our last conversation, and thanks mom for never asking. Special thanks to my husband for convincing me to stay in graduate school during the famed post generals slump.

- Princeton University, Gordon Wu Fellowship, National Science and Engineering Research Council (Canada), National Science Foundation, DARPA and Cisco Systems for the fellowships and grants which allowed me to write this dissera-tion.

Contents

Chapter 1

Prologue

The Internet today provides a best-effort packet-delivery service for many popular applications including e-mail, web, file sharing, IPTV, online gaming and voice-over-IP. As the Internet grows in size and complexity, managing how packets traverse the Internet has become an increasingly important and challenging task. *Traffic management* controls how much traffic traverse each path in a network, which directly impacts the congestion experienced by each data packet, as well as how efficiently the network resources are utilized. Due to its practical importance and intellectual breadth, traffic management has been an active area of networking research for many years.

1.1 Case for Rethinking Traffic Management

Traffic management includes three players: users, routers, and operators. In today's Internet, users run *congestion control* to adapt their sending rates at the edge of the network depending on network conditions. Inside a single Autonomous System (AS), routers run shortest-path routing based on link weights. Operators tune link weights to minimize congested links [30]. Routing, congestion control and traffic engineering sequentially became part of traffic management as the Internet itself evolved from a small research network (ARPAnet) in 1969 to the huge commercial networks of today.

Since the first transmission of packets between two end hosts in 1969, routing protocols have been used to route packets through the network. In the 1970s, the ARPAnet attempted to implement a routing protocol that automatically selected the shortest-delay path (to improve efficiency), but observed oscillatory behavior [63]. The routing can be stabilized, but at the cost of efficiency, thus the design decision at the time was to keep routers from automatically adapting to traffic shifts. This historical decision led researchers and practitioners alike to believe it was fundamentally challenging to attain both stability and efficiency. This dissertation, and other recent research [46, 28], suggests otherwise.

In October 1986, the first *congestion collapse* was observed when NSFnet's (research network funded by the National Science Foundation) capacity dropped three orders of magnitude from 32kbps to 40bps [2]. The congestion collapse occurred because the Internet has no admission control, so end hosts can send as much traffic as they like. When packets are lost, the earlier congestion control mechanisms would immediately retransmit packets, without reducing the sending rate, causing even more packet loss. To avoid congestion collapse, end hosts implemented new congestion control mechanisms where sending rates are decreased whenever a packet has been lost. Overall, congestion control allocates network resources fairly amongst greedy users.

Following commercialization in the 1980s and introduction of privately run Internet Service Providers (ISPs) [3], the Internet is now composed of multiple commercial entities, each responsible for managing how traffic traverse its network. Further, Internet's expansion into mass popular use in the 1990s placed an increased strain on network resources. As bandwidth is expensive, ISP operators started to monitor their networks for signs of overloaded links and adapt the routing of traffic to alleviate congestion in a process called *traffic engineering*. Traffic engineering allows operators to use existing bandwidth more efficiently, thus reducing costs. In addition, for a given set of network resources, traffic engineering allows operators to provide better packet delivery services for their customers.

Internet traffic management has significantly improved since the ARPAnet. Still, due to the organic evolution of traffic management, there are several shortcoming. First, operators tune link weights assuming that traffic is static, and end hosts adapt their sending rates assuming routing is fixed. Second, tuning link weights is an indirect way to control traffic flow through a network; further, the link-weight setting problem is computationally challenging, forcing operators to resort to heuristics that can lead to highly suboptimal solutions. Third, since this offline optimization occurs at the timescale of hours, it does not adapt to changes in the offered traffic. Finally, traffic management today is designed to maximize throughput for users, and does not consider that some applications have different performance objectives, such as minimizing delay.

These shortcomings prompts us to rethink the traffic-management system as a whole. A natural objective for an ISP is to *maximize aggregate performance across multiple traffic classes*, where each traffic class has a different performance objective. Our design goals for the overall traffic-management system are:

1. **Fair:** bandwidth should be fairly allocated between multiple traffic classes.

2. **Efficient:** bandwidth should be efficiently utilized to maximize aggregate performance objectives.

3. **Distributed:** in order to adapt on a small timescale, all protocols should be possible with distributed computation. When possible, message passing between network elements should be minimized.

4. **Robust:** all protocols should be robust to traffic shifts and topology changes.

5. **Implementable:** all protocols should be implementable using existing technology.

To accomplish these design goals, we leverage both optimization theory and technology trends. Section 1.2 discusses how optimization theory can help design distributed traffic-management protocols that are fair and efficient. In addition, numerical experiments and simulations can test the robustness of the resulting protocols under realistic network conditions. Section 1.3 reviews current technology trends in traffic management to determine the deployability of a protocol. Finally, Section 1.4 highlights how each chapter of this dissertation contributes to Internet traffic management.

1.2 Role of Optimization in Traffic Management

Of the many mathematical tools available, optimization theory is a natural choice for analyzing and redesigning Internet traffic management. Due to its role in analysis and design of various components of traffic management, optimization can place new traffic-management protocols on a strong foundation.

Traffic engineering and congestion control both solve, explicitly or implicitly, optimization problems defined for the entire network. Traffic engineering consists of collecting measurements of the traffic matrix—the observed load between each pair of entry and exit points—and performing a centralized minimization of a cost function that considers the resulting utilizations on all links (*e.g.*, [30, 74]). In contrast, TCP (Transport Control Protocol) congestion control can be viewed as *implicitly* solving an optimization problem in a distributed fashion (*e.g.*, [48, 59, 58, 82]), where the many variants of TCP differ in the shape of user utility as a function of the source rate. Further, optimization theory is used to *analyze* proposed traffic-engineering protocols, *e.g.*, [24], as well as to *design* distributed congestion control protocols, *e.g.*, [90].

Distributed solutions can be derived using *optimization decomposition*: a standard optimization technique for decomposing a single optimization problem into multiple sub-problems. Each subproblem can be solved by an individual network element such as a router, an end host or a link. In order for the distributed solution to achieve an overall objective, the network elements coordinate with each other explicitly through message passing, or implicitly through measuring locally observable quantities such as link load, delay and packet loss. To ensure convergence, distributed solutions derived from optimization decomposition often contain iterative updates with tunable parameters. The tunable parameters serve to moderate the rate of adaptation. Optimization decomposition has been widely used to derive distributed solutions to a variety of networking problems, as surveyed in [21].

This dissertation leverages optimization theory in three distinct ways to rethink the traffic-management system as a whole. First, in chapter 3, we use established optimization models to *analyze* today's interaction between congestion control and traffic engineering. Second, optimization decomposition is used to *design* distributed traffic-management protocols for throughput sensitive traffic in Chapter 4 and resource allocation between multiple traffic classes in Chapter 5. Third, optimization theory is used to *dictate* the placement of function in Chapter 5.

While optimization theory puts our work on a rigorous foundation, it has limitations, as with any mathematical tool. First, mathematics do not specify the translation from an algorithm into a packet-level protocol. Second, when simplifying assumptions made in modeling do not hold, the derived algorithms do not naturally handle them. Third, while distributed algorithms derived using optimization provably converge to a stable and optimal point, optimization theory only provides loose bounds on the rate of convergence, and provides little guidance on setting tunable parameters. Fortunately, properties which can be proven mathematically are just a subset of properties which are true. To further understand the capabilities of the system, we supplement optimization theory with numerical experiments and packet-level simulations.

Numerical experiments are useful for sweeping a large parameter space, and can serve as early indicators of an algorithm's potential. For example, we can compare multiple distributed algorithms derived using different decomposition methods [70]. While useful, numerical experiments cannot fully capture realistic network conditions. So after translating an algorithm into a packet-level protocol, packet-level simulations can be used to understand the behavior of a distributed protocol under realistic feedback delays and traffic loads.

1.3 Technology Supporting Traffic Management

Optimizing for performance is not the sole design goal. For an architecture or a protocol to succeed, an equally important, though sometimes opposing goal is simplicity. Simplicity can be defined as the ease of implementing a protocol or an architecture with existing technology, while keeping the overhead low. In this dissertation, current technology trends are used to guide modeling assumptions, as well as understand the implementation possibilities for an algorithm derived from optimization.

A key assumption in this dissertation is that routers can *flexibly divide traffic over multiple paths*. Most current routing protocols select a single path between two end hosts in spite of existing path diversity. Today, support for Internet-wide multipath routing faces two significant deployment barriers. First, multipath routing could impose significant computational and storage overheads in a network the size of the Internet. Second, the independent networks that comprise the Internet will not relinquish control over the flow of traffic without appropriate incentives. Fortunately, having one or two extra paths is enough for significant gains in security, performance, and reliability.

In fact, today's routers are already capable of establishing multiple paths between each other using MultiProtocol Label Switching (MPLS) technology, and there are existing options for traffic engineering using MPLS. This dissertation proposes distributed multipath traffic-management protocols, which can be implemented using MPLS. Unlike existing traffic engineering using MPLS, implementing the distributed multipath protocols in this dissertation also requires changes to the computations performed by the routers. *Router programmability*, ability to run customized protocols, is a flexible and extensible way to implement a variety of distributed protocols. Though not yet a reality, vendors have recently indicated their interest in supporting programmable routers [4, 5].

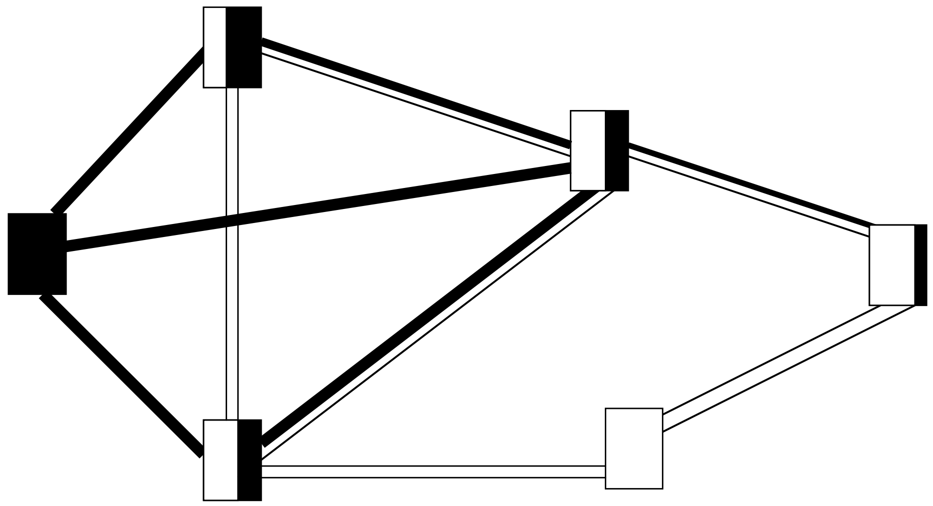


Figure 1.1: Two virtual networks are shown. The shaded regions identify the portion of node and link resources allocated to one virtual network. The remaining resources are allocated to the second virtual network.

To support multiple traffic classes in parallel, optimization theory indicates the need for *separate resources* for each traffic class. One potential implementation is to run each traffic class on a virtual network. Virtual networks are constructed over a substrate network by first subdividing each physical node (*i.e.*, router) and each physical link into multiple virtual nodes and virtual links, as in Figure 1.1. A virtual node controls a subset of the underlying node resources (such as CPU and bandwidth). A virtual link can span several substrate links, taking up a portion of the bandwidth of each underlying substrate link. Each virtual link has its own queue and possibly customized forwarding logic. The substrate runs schedulers that arbitrate access to the shared node and link resources, to give each virtual network the illusion that it runs on a dedicated physical infrastructure. Today, network virtualization is moving from fantasy to reality. Major router vendors already support router virtualization (to run multiple virtual routers in parallel on a single router) [62, 1], and MPLS technology can be used to establish virtual links.

1.4 Contributions of Thesis

This dissertation provides a holistic view of traffic management, using a mixture of theoretical and practical tools. To lay the background for the technical chapters, Chapter 2[1] surveys multipath routing techniques in existing literature. In particular, Chapter 2 focuses on techniques which are both scalable and incentive compatible.

The analysis and redesign of today's traffic management proceeds in two phases. First, taking a "bottom-up" approach that analyzes and characterizes the interaction between TCP congestion control and conventional traffic-engineering practices in Chapter 3.[2] Then taking a "top-down" approach where we redesign and evaluate a new, dynamic, distributed algorithm in Chapter 4.[3] The two systems differ in four ways, summarized in Table 1.1.

In the "bottom-up" approach, we find the TE model is stable, but does not maximizes aggregate user utility. By tuning the cost function used for traffic engineering, we prove the joint system can maximize aggregate utility. Such a change is unde-

[1]Chapter 2 has been published as [39].

[2]Chapter 3 appeared as [37], and was later published as the first half of [35].

[3]Some of the ideas in this Chapter 4 has been published as [36] and in the second half of [35]. The main publication is [40], and an journal version has been submitted to Transactions of Networking.

	TE Model	**TRUMP**
Focus	analysis	design
Approach	bottom-up	top-down
Timescale of route adaptation	offline	online
Computation of routes	centralized	distributed

Table 1.1: Differences between "TE Model" in chapter 3 and "TRUMP" in 4.

sirable, however, since the system will be fragile to traffic bursts. This is one of the motivations for redesigning traffic management in the subsequent chapter.

In the "top-down" approach, we propose a *new objective function* that captures the goals of both end users and network operators. Next, using various optimization decomposition techniques, we *generate* four distributed algorithms that divide traffic over multiple paths based on feedback from the network links. These distributed algorithms are provably stable and optimal, but can converge slowly and be sensitive to tuning parameters. Finally, combining the best features of these distributed algorithms, we *construct* TRUMP: TRaffic-management Using Multipath Protocol. TRUMP converges quickly and contains a single easy to tune parameter. Packet-level simulations show TRUMP behaves well with realistic topologies, feedback delays, capacities, and traffic loads.

Today's traffic management is a 'one-size-fits-all' packet-delivery service, not tailored to suit the needs of any specific application. Since different applications today may have different performance objectives, we extend our redesign of traffic management to handle multiple traffic classes in Chapter 5. Taking an ISP's perspective, the objective is to maximize aggregate performance objectives across multiple traffic classes. Decomposing the ISP's problem leads to a stable and optimal solution where each traffic class optimizes for its own performance objective, with an algorithm to dynamically allocate bandwidth shares.

To tie together the ideas presented in the dissertation, the second half of Chapter 5 presents a novel *architecture* DaVinci: Dynamically Adaptive Virtual Networks for a Customized Internet. In DaVinci, each virtual network runs its own set of traffic-management protocols optimized for a particular traffic class, and a link coordinator assigns bandwidth shares dynamically between the virtual networks. In addition, a traffic shaper on each queue prevents virtual networks from claiming excess bandwidth on a small timescale. DaVinci has the following properties:

- **Stability:** The bandwidth shares computed by each link coordinator converge to a stable value, without requiring information about the bandwidth shares of other links.
- **Efficiency:** The multiple virtual networks and the link coordinator collectively maximize the aggregate performance of all virtual networks.
- **Independence:** Although bandwidth shares are changing over time, each virtual network can design and run its traffic-management protocols as if it had dedicated resources.

Finally, Chapter 6 wraps up the dissertation by exploring configuration complexity.

Chapter 2

Multipath Routing

2.1 Introduction

Researchers and practitioners alike agree multipath routing provides performance benefits to traffic management. Still, most currently deployed routing protocols select only a single path for the traffic between each source-destination pair. This chapter explores techniques that allow a *flexible* division of traffic over multiple paths. That is, a source (an end host or edge network) has access to multiple paths through the Internet, and direct control over which traffic traverses each path. Though sources have limited knowledge of and control over multiple paths today, flexible multipath routing is feasible with existing technology.

2.1.1 Motivation for Flexible Multipath Routing

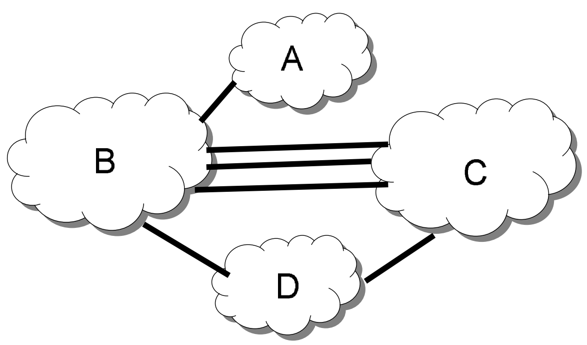
Flexible Internet-wide multipath routing would offer many benefits, including the following:

- *Customizing to application performance requirements:* Different applications have different needs. If multiple paths exist, VoIP and online-gaming traffic can use a low-delay path, while file-sharing traffic uses a high-throughput path.

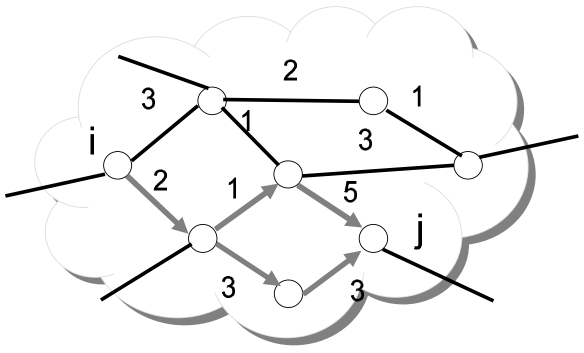
In addition, an application can access more bandwidth by using multiple paths simultaneously.

- *Improving end-to-end reliability:* If multiple paths exist, traffic can switch quickly to an alternate path when a link or router fails. Similarly, if an adversary drops packets along a path, the traffic could be moved to an alternate path to circumvent the adversary [91]. This is particularly useful if disjoint paths are available.

- *Avoiding congested paths:* When multiple paths are available, traffic can move to an alternate path to circumvent congestion. Despite problems with routing oscillation in the early ARPANET, recent work has shown how to dynamically split traffic over multiple paths in a stable fashion [46]. In fact, by just having two paths and flexible splitting between them, protocols can be easily tuned to efficiently utilize network resources.



(a) Topology between 4 networks: B and C are ISPs, A and D are enterprise networks



(b) Topology inside network C

Figure 2.1: Sample inter-network topology, with a close-up on one network.

Past work indicates that the Internet's network-layer topology has significant underlying path diversity.[1] Each network is a collection of routers and links under the

[1]This chapter focuses on the path diversity at the network layer. There is a large body of research on path diversity at the physical layer which is important for reliability and security, but the shared risks at the physical layer are not visible to the IP routing system.

control of one entity, such as an *Internet Service Provider* (ISP) that offers connectivity to other networks or a *stub network* that just provides connectivity to its own users and services. This chapter explores how to give stub networks greater end-to-end path diversity. Extra end-to-end paths may arise because a stub network is connected to multiple ISPs, individual ISPs have intradomain path diversity, or ISPs connect to each other in multiple locations. In fact, a measurement study of a large ISP found that almost 90% of Point-of-Presence (PoP) pairs have at least four link-disjoint paths between them [85]. Another study showed that, although Internet traffic traverses a single path, 30% to 80% of the time, an alternate path with lower loss or smaller delay exists [78].

2.1.2 Challenges: Scalability and Incentives

Unfortunately much of the existing path diversity in today's Internet is never exploited. The scalability challenges of multipath routing is one of the reasons. Multipath routing would introduce extra overhead in both the *control plane* and *data plane* of the routers. In the control plane, routers exchange information and compute the forwarding tables that the data plane then uses to direct incoming packets to outgoing links. Multipath routing would increase the overhead in both the control and data planes:

- *Control-plane overhead:* First, exchanging the extra topology or path information required for multipath routing would consume extra bandwidth and processing resources. Second, storage overhead at each router would grow with the number of paths. Third, computing multiple paths would require more computational power.

- *Data-plane overhead:* Forwarding traffic on different paths requires the data packets to carry an extra header or label. In addition, forwarding tables need extra entries for each destination, thus consuming more memory; in addition, this data-plane memory is expensive, due to the need to forward packets at high speed.

The ultimate flexibility would be for sources to see the entire Internet-wide topology and utilize *any* path to each destination. This would create a large scaling problem, however, since the Internet has more than 25,000 networks and many more paths. Even if the scalability challenges were surmountable, accessing *all* paths would require many (or even all) networks to cooperate, which may be unrealistic. Instead, it is more likely for ISPs to allow other networks to select from a small set of paths, under a specific business agreement. Since business models in the Internet today are *bilateral*, multipath solutions based on cooperation between pairs of networks are much more likely to succeed than solutions that require widespread cooperation between many (sometimes competing) networks. Fortunately, multipath routing solutions that limit the number of additional paths and the coordination between different networks are aligned with both goals—scalability and business incentives. As such, this chapter focuses on solutions where stub networks select amongst a small set of paths provided by a limited number of bilateral agreements, rather than techniques that require a stub network to compute and signal a complete, end-to-end path.

This survey focuses on multipath routing schemes with low overhead and minimal cooperation between networks. The sections progress from deployed techniques to proposed solutions that are easily deployable, to techniques that rely on new business models. We start by reviewing how Internet routing works today in Section 2.2, with an eye towards the limitations of the existing routing system. End-to-end multipath routing relies on two key capabilities: discovering extra end-to-end paths and directing packets over them. Section 2.3 covers a range of solutions for flexible forwarding such as tunneling and tagging, for directing packets to different paths, while Sections 2.4 and 2.5 describe control-plane extensions that enable networks to learn additional paths. In particular, Section 2.4 discusses techniques for a single network to achieve multipath routing, without requiring cooperation from other networks. The impact of a single network on end-to-end path performance is limited, however, and more end-to-end paths would be available if networks cooperated. Section 2.5 discusses techniques which only require cooperation between a pair of networks. Finally, we conclude in Section 2.6.

2.2 Internet Routing Today

In this section, we introduce the key routing protocols used in today's Internet. Routers use the Border Gateway Protocol (BGP) [73] to exchange reachability information with neighboring networks. BGP is a *path-vector* protocol, where routing decisions are made based on local policies. Inside a network, routers communicate using an Interior Gateway Protocol (IGP) [19, 68]. Most ISPs run *link-state* protocols that perform shortest-path routing based on configurable link weights. The link weights in IGP and the policies in BGP are configured by human operators to satisfy business objectives.

2.2.1 Interdomain: Path-Vector Protocol and Multihoming

Figure 2.1a represents a network-level topology, where each cloud is a network and each link represents a physical connection, as well as the existence of a business relationship between two networks. In a path-vector protocol, the *entire routing path* is exchanged between neighbors. Edge routers in each network learn multiple paths to reach a particular destination and store all of them in a routing table. From the list of paths, a router then applies a set of policies to select a *single active route*. A router optionally advertises the active route to each neighboring network, depending on the business relationship. Using a path-vector protocol allows BGP to support flexible local policies that give each network control over its incoming and outgoing traffic. For example, a stub network, like network D in Figure 2.1a, would not advertise routes learned from B to C (and vice versa) because D does not wish to carry transit traffic between the two neighbors.

Today's BGP has two limitations as a single-path protocol. First, since only the active path is advertised, customer networks are prevented from seeing alternate paths, including ones they might prefer. Second, by using only the active path, a network does not have fine-grained control, and can only balance traffic over multiple paths at the IP address block (i.e., prefix) level. Extending BGP to a multipath protocol, however, requires alignment of economic incentives between networks. The economic incentives are likely to grow stronger in the future as the demand for performance and robustness increase, and customers are willing to pay for value-added Internet services. Today, two networks usually have a customer-provider relationship or a peering relationship. In a peering relationship, two networks could mutually provide additional paths to each other without any economic exchange, similar to how they carry traffic on peering links for free today. In a customer-provider relationship, the provider could offer additional paths to its customers as a value-added service.

One such example is multihoming [12], where a stub network pays to connect to more than one ISP. The use of multihoming has seen a dramatic increase in recent years for two main reasons. First, as more enterprises rely heavily on the Internet for their business transactions, having a second provider is important to survive a failure of the other provider. Second, multihoming can be used to drive down the cost of Internet access. For example, the multihomed network can use a cheap ISP for most traffic and an expensive but better ISP for performance-sensitive traffic [75]. In Figure 2.1a, network D is multihomed to networks B and C. Despite having two upstream routes, network D can only balance load between the two at the prefix level, and only forwards traffic for each destination on a single path. So, while multi-homing provides additional paths to stub networks, fine-grained control remains elusive.

2.2.2 Intra-domain: Link-State Protocol

Unlike the interdomain case, each network has full control of its internal network. In addition, a network typically has just tens or hundreds of routers, much fewer than the 25,000 networks in the Internet. Inside a single network, each router is configured with a static integer weight on each of its outgoing links, as shown in Figure 2.1b. The routers flood the link weights throughout the network and compute shortest paths as the sum of the weights using Dijkstra's algorithm. Each router uses this information to construct a table that drives the forwarding of each IP packet to the next hop in its path to the destination. Link-state protocols offer several advantages. First, routing is based only on a single link metric, *i.e.* link weights. Second, to reduce message-passing overhead, routers only disseminate information when the topology changes. Finally, by flooding the link-weight information, each router has a complete view of the topology and associated link weights.

On the other hand, even though each router can see the whole topology, the existing path diversity is under-exploited [85]. Even when alternative paths have been computed, packets towards a destination are often forwarded on a single path. *Equal-cost multipath* is a commonly deployed technique where the routers keep track of all *shortest* paths, and then evenly split amongst them. In Figure 2.1b, we see that router i has two shortest paths to reach router j. In today's IGPs, the traffic would be divided evenly between the two paths. Even this limited version of multipath routing is useful for fast reaction to failures. In fact, some operators tune the link weights to create equal-cost multipaths [41].

Multiple shortest paths enable the operator to balance load and react quickly to failures, but does not enable the operator customize paths for different applications. An existing option for operators to customize paths inside their own network is the

Constrained Shortest Path First (CSPF) protocol [42], an extension of the shortest-path protocol. The path computed using CSPF is a shortest path fulfilling a set of constraints. A constraint could be minimum bandwidth required per link, end-to-end delay or maximum number of links traversed. CSPF can be useful for a range of applications, *e.g.*, picking a low-delay path for a VoIP call, but cannot pick paths based on dynamic constraints such as packet loss.

2.3 Towards Flexible Forwarding

The most prevalent forwarding mechanism in the Internet today is *destination-based hop-by-hop forwarding.* Each router forwards a packet to an outgoing link based on the destination address from the IP packet header and the corresponding longest-prefix match entry in the forwarding table. For example, in Figure 2.1b, a router will forward a packet destined for j, independent of where the packet came from. Destination-based hop-by-hop forwarding leads to small forwarding tables, but cannot realize flexible forwarding policies. For example, in Figure 2.1a, if network A wanted to reach D via (B, C), but B wanted to reach D directly, then A is forced to use path (A, B, D). Even when the forwarding table contains multiple next hops for the same destination, common practice would divide the traffic evenly amongst the multiple paths.

In this section, we describe alternative schemes which forward traffic over multiple paths. This is useful for customizing paths for different applications. In order to decide which path should carry a packet, an edge router or end host need to first classify a packet, and then map the packet to a corresponding path:

- *Packet classification:* Packets can be classified based on the requirements of the application, [61]. For example, the application may want low delay, high throughput, or a secure path. The application could be defined by a prefix, a destination, or a TCP flow (source and destination addresses and port numbers). Packets within the same flow are normally classified in the same way. One option is to mark the Type of Service (ToS) bits in the IP header, and later forward the packet using the same bits.

- *Mapping packets to paths:* The edge routers can measure (or infer) path properties, to determine which path is best-suited to each class of traffic. By examining the packet header, a packet can be mapped to an appropriate path. Designing a measurement infrastructure to monitor path performance is challenging. One reason is that measurements of path performance can be inherently inaccurate; for example, round-trip time estimation is a classic challenge. In addition, the inaccuracies can be even greater in a competitive environment where other networks may treat probing packets differently than data packets to make paths look more attractive than they are.

Both steps incur extra data-plane overhead. Though the overhead of marking packets and processing the marked packets is minimal, the measurement overhead associated with monitoring path performance can be significant, particularly if the measurements are fine-grained (e.g., the destination prefix level).

If multiple paths are associated with a particular class of traffic, the router can send a fraction of the packets on each path, to balance load and circumvent congestion. In Section 2.3.1, we survey existing techniques for forwarding packets on alternate paths. In Section 2.3.2, we discuss the pros and cons of splitting traffic at different granularities. We focus on existing techniques (round-robin, hashing, and flow-cache), but also describe flowlet-cache, a promising technique that is yet to be deployed.

2.3.1 Forwarding on Alternate Paths

Tunneling is a widely available alternative to destination-based hop-by-hop forwarding that offers much more flexibility. At a high level, tunneling establishes a *logical* link between two routers (or hosts). Forwarding packets over a tunnel usually involves "pushing" a header (or label) at the tunnel ingress and "popping" the header (or label) at the tunnel egress, in a process called *encapsulation*. For example, in Figure 2.2, a packet going from B to F could be encapsulated to ensure it travels through E. At B, an extra header would be "pushed" on the packet to indicate E is the destination. Once the packet reaches router E, the extra header would be "popped" from the packet, then E would forward the packet to F hop-by-hop. Encapsulation can be implemented through IP-in-IP tunnels or MultiProtocol Label Switching (MPLS) [77]. MPLS is a label-based forwarding mechanism that encapsulates packets using labels. In either case, encapsulation requires packets to carry an extra label or an extra IP header. In the case of MPLS, each router also stores the label-based forwarding table, although a label-based look-up is simpler than matching the longest prefix of the destination address.

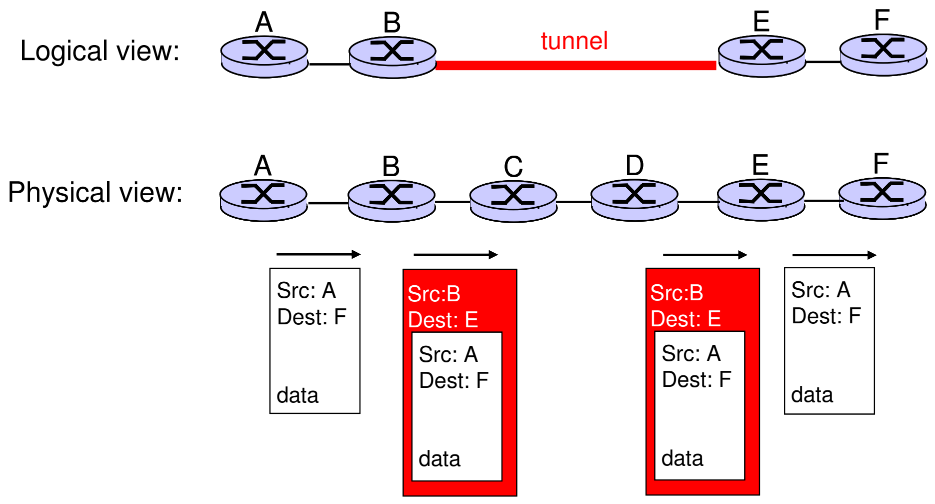


Figure 2.2: Illustration of how a tunnel works.

The path between the tunnel ingress to tunnel egress can depend on the underlying routing protocol, or the entire path can be specified explicitly. Encapsulation alone is often sufficient for most application needs such as directing a packet to a particular egress point or through a particular network. When the path between tunnel endpoints only depends on the underlying protocol, the path adapts automatically when the topology changes. For example, in Figure 2.2, B could forward the packet towards E one hop at a time. This implies if the link from C to D fails, the encapsulated packets would *transparently* switch to another path. Still, by only specifying the endpoints of the tunnel, it is difficult to satisfy certain applications needs, *e.g.*, an end-to-end bandwidth requirement. So for those specialized applications, explicit routing is a useful alternative.

Explicit routing specifies every router (or network) along the path. The routers (or networks) along the path can be specified directly in the packet header or indirectly through a label in the packet header. One possibility is to implement explicit routing by specifying the whole router-level path with IP options. In Figure 2.2, if the path sequence (A, B, C, D, E, F) is an explicit path for certain packets traveling from A to F, then A would know to forward to router B based on the IP options in the packet header. An alternative is to implement explicit routing with MPLS as a combination of Constrained Shortest Path First (CSPF) and Resource Reservation Protocol (RSVP). CSPF selects the path using a variety of metrics, while RSVP is the signaling protocol used to set-up the path within a single network. RSVP establishes a hop-by-hop chain of labels to represent the path and it reserves bandwidth along the path by signaling in advance. At source end of the path, a label would be pushed onto the packet based on information from the packet header such as source address, destination address, and port numbers. Each intermediate router would do a label look-up to find the outgoing label and outgoing link. Compared to tunneling, explicit routing does impose more data-plane overhead (to swap the labels at each hop), though the overhead is manageable when the number of explicitly-routed paths is limited.

2.3.2 Flexible Splitting Amongst Multiple Paths

The network management system may wish to balance traffic between multiple paths to achieve certain traffic engineering objectives. For example, sending 40% of traffic on one path and 60% on another could lead to less congestion in the network. To achieve a splitting percentage determined by the network management system, traffic can be switched onto different paths using four major techniques: round-robin, hashing, flow cache, and flowlet cache [79]. Each technique strikes a different trade-off between overhead, splitting percentage accuracy, and the likelihood of packet reordering.

A weighted **round-robin** will switch traffic at the granularity of packets. Since packets are small in size, round-robin scheduling can achieve very accurate splitting percentages on a small timescale. Round-robin scheduling also adds very little extra overhead on today's forwarding functions. The downside is that since different paths between the same source-destination pair often have different delays, some packets which belong to the same TCP flow could arrive out-of-order. This is problematic as TCP considers out-of-order packet delivery as a sign of network congestion, and consequently, the TCP sender would slow down the transfer. If the paths have very similar delay, then weighted round-robin is a good choice due to its low overhead and accurate splitting percentages.

Hashing involves first dividing the hash space into weighted partitions corresponding to the outbound paths. Then packets are hashed based on their header information and forwarded on the corresponding path. A flow is defined by the following attributes in the packet header: source IP address, destination IP address, transport protocol, source port, and destination port. Hashing ensures in-order delivery of most packets since a flow is likely to be mapped to a specific path for its entire duration. On the other hand, since flows vary drastically in their sizes and rates, it is difficult to realize accurate splitting percentages. Finally, if splitting percentages change or a path fails, a flow is likely to be hashed onto a different path, possibly causing a few out-of-order packets during the transition. A variant of hashing (consistent hashing) can minimize the fraction of flows that must change paths when the splitting ratio changes.

The best way to avoid out-of-order packets is to implement a **flow cache**. A flow cache is a forwarding table that keeps track which path each active flow traverses. A flow cache ensures packets belonging to the same flow always follow the same path. Another advantage of flow caching over hashing is that when new flows arrive, they can be placed on any path, which leads to better control of dynamic splitting percentages, although the splitting percentages achieved are less accurate than in round-robin scheduling. The major drawback is that a high-speed link could easily carry tens of thousands concurrent flows [79], leading the flow cache to consume a significant amount of additional memory in the router.

It is possible to reduce data-plane overhead and improve splitting ratios by dividing traffic at the granularity of packet-bursts, using a **flowlet cache** [79]. If the time between two successive packets is larger than the maximum delay difference between the multiple paths, the second packet can be safely forwarded on any available path without the risk of packet reordering. A flowlet cache is typically much smaller than a flow cache, since there are significantly fewer active packet bursts than active flows [79]. In addition, flowlet switching always achieves within a few percent of the desired splitting percentage, without reordering any packets. Overall, flowlet cache would be the best choice for most applications, although it is not yet implemented in routers today.

2.4 Multipath Routing by a Single Network

In this section, we present incrementally deployable techniques which can be adopted by a single network. Each ISP can exploit its internal path diversity, and a multi-homed stub network can split traffic over multiple end-to-end paths.

2.4.1 Intradomain: Non-shortest Paths within an ISP

Each network can select its own IGP, allowing it to change the protocol without requiring cooperation from others. In link-state protocols, since link weights and topology information are already flooded to all routers, multipath routing does not incur extra dissemination overhead. One natural way to extend a link-state protocol is to compute the K-shortest paths rather than just the shortest path. This is cumbersome for several reasons. To start with, computing the K-shortest paths is more computationally intensive (i.e., $O(N \log N + KN)$ for a network with N routers [25]) than computing a single shortest path (i.e., $O(N \log N)$). The forwarding-table size would also grow with the increase in number of paths per destination. Perhaps the biggest overhead increase is in the data plane, where K tunnels need to be established between each source-destination pair. If each router does destination-based hop-by-hop forwarding, then there is no guarantee packets would travel on the K-shortest paths from source to destination. This is significantly more cumbersome than the current hop-by-hop forwarding.

Another approach is to run multiple instances of the link-state routing protocol [67]. Instead of having a *single* weight associated with each link, each link has a *vector* of weights. Each instance of the link-state protocol can just compute the shortest path and create a forwarding table for the corresponding topology. The vector of weights does not lead to the K shortest paths, but rather a shortest path for each of K sets of link weights. Each set of link weights can be tuned independently to customize the paths to different applications; for example, one set of weights could be tuned for high throughput and another for low delay. The link weights could even be specialized to handle different failure scenarios. In the control plane, if K routing instances run simultaneously, the control-plane overhead would be exactly K times as much as shortest-path routing. In the data plane, there are two ways to forward packets on the multiple topologies. The simpler (and more restrictive) way is for each packet to belong to a single topology [53]. Further benefits are possible when packets can switch between topologies based on network conditions [67].

An alternate approach to multipath routing is to forward traffic on *all* paths that make forward progress toward the destination [94, 92], based on a single set of link weights. Each router can make local forwarding decisions based on the cost of the shortest path through each of its neighbors [92]. Forwarding packets only to routers that have a shorter path to the destination guarantees that the path is loop-free [94]. To encourage the use of shorter paths, diminishing proportions of the traffic would be directed on the longer paths. For example, in Figure 2.1b, i has two outgoing links along shorter paths to j. Since these paths have costs 8 and 9, less traffic would be placed on the path with cost 9 [92]. Under this scheme, the path-computation costs are still $O(N \log N)$, since each router will just run Dijkstra's shortest-path algorithm. Compared to shortest-path routing, forwarding along the "downward" paths requires more entries in the forwarding tables. In addition, there will be slightly more data-plane overhead in order to implement the splitting percentages, as explained in Section 2.3.2. Still, each router can make local forwarding decisions without the use of tunnels.

2.4.2 Interdomain: Fine-grained Splitting by a Multihomed Stub

So far, we have described how to exploit path diversity inside a single network. Next, we will examine how to exploit interdomain path diversity. Many routers learn multiple interdomain paths and could conceivably split traffic over them by installing multiple next-hops in the forwarding table. This is not done in practice due to the extra control-plane overhead. For ISP networks, edge routers would need to announce multiple paths to neighboring networks, and the neighboring networks would now need to store multiple paths. In addition, tunneling would be needed to direct packets on any non-default path, as explained in Section 2.3.

In a stub network, however, edge routers do not need to propagate any of the learnt paths. In addition, packet classification is simpler for a stub network since the data rates tend to be lower and all packets originate from a single domain. Therefore, stub networks are natural places to deploy flexible splitting. Since applications are run at the edge, the stub network also has direct knowledge of the application requirements. Flexible splitting enables a network to place different classes of traffic onto different paths and balance load across multiple paths. Balancing load between multiple classes allows for efficient use of network resources and can avoid potential routing oscillations. For example, if all traffic is forwarded on the least-delay path, route oscillations can occur [50]. Luckily, flexible path selection adds very little extra overhead on the data plane for a stub network, since choosing an outgoing link determines the entire path a packet will follow and no tunneling is required.

2.5 Cross-network Cooperation for Multiple Paths

When multiple networks cooperate, even more paths are available than when a network acts alone. In addition to scalability challenges, new business models must be put in place to enable inter-network cooperation, *e.g.*, charging for providing additional paths. In this section, we focus on proposed schemes which access additional paths with only limited cooperation between networks. Sources can encapsulate packets to direct the traffic through a *deflection point*—an end host or edge router that lies on an alternate path. This only requires a bilateral agreement between two parties. Sources can also deflect packets indirectly via *tagging*, where a few opaque bits in the packet header are used to indicate dissatisfaction with the current path. Tagging requires more networks to cooperate since routers need to be modified to forward packets based on the tags.

2.5.1 Encapsulation: Forwarding through a Deflection Point

Encapsulation can be used to explicitly force traffic onto an alternate path with better performance properties. A packet would be encapsulated to first arrive at the deflection point, then follow that deflection point's default path to the destination [91, 93]. Deflections can occur at the application layer or the network layer.

The easiest way to access another path is by deflecting through another end host, which does not require cooperation from or coordination between ISPs. First, an *overlay* or logical topology can be established between end hosts using tunnels [7]. Then each end host can measure the end-to-end performance properties of paths to a destination via other end hosts. If a path with better performance is found, packets can be deflected through another end host as seen in Figure 2.3. In addition to ease of deployment, application-layer deflections are attractive because they avoid advertisement of additional paths. On the other hand, as the overlay grows in size, probing all paths through other end hosts imposes a significant amount of measurement overhead and does not scale beyond tens of end hosts [7]. In addition, sending traffic through other end hosts consumes edge link bandwidth and potentially incurs extra costs for the edge network.

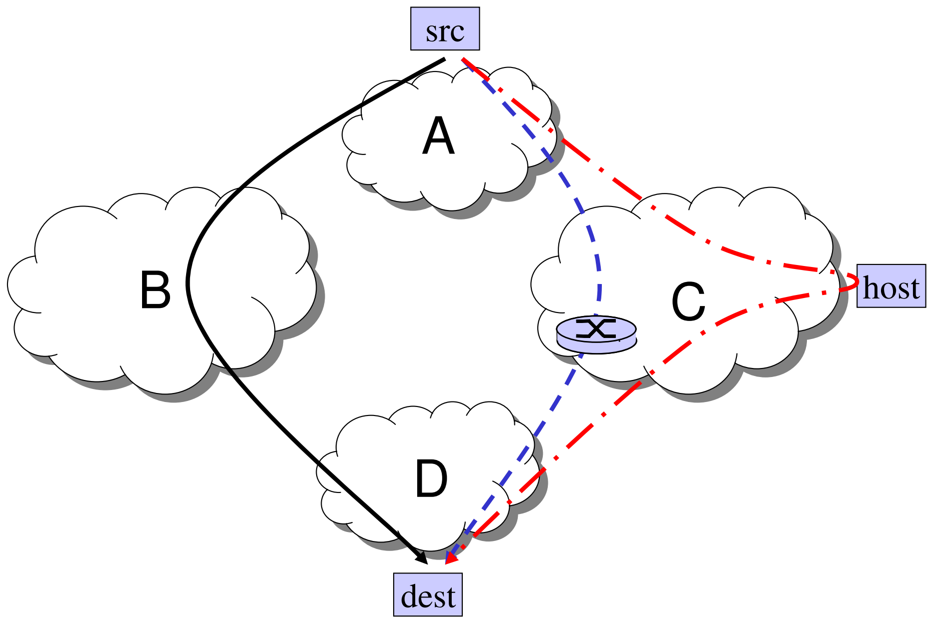


Figure 2.3: The default path, shown in solid line is through network B. Deflection through network C is possible either with an overlay (dot-dash line) or through an ISP (dashed line).

A more scalable and efficient approach is for ISPs to provide alternative paths [91, 93]. As seen in Figure 2.3, the deflection point can be an edge router inside a network, rather than an end host. While this approach requires more cooperation from (and between) ISPs, it is still incrementally deployable. To ensure scalability, a network would only request an alternative path (perhaps with certain properties) from another network if it is unhappy with its default path. For example in Figure 2.3, the source could request an alternative path from its provider network A for reaching the destination D, network A can then choose to forward traffic on the alternative path (A, C, D), possibly for a price. Encapsulation would be used to deflect the packets through network C. The amount of control-plane overhead is directly proportional to the portion of networks unhappy with their default paths.

2.5.2 Tagging: Requesting an Alternate Path

An alternative to encapsulation is for end-hosts to simply *tag* their packets to request an alternate path [67, 94], without knowing the details of the path. A router forwards an incoming packet on the default path or an alternate path, based on the associated tag. Alternative paths inside an ISP can be constructed by one of the methods described in Section 2.4.1.

Tagging without path visibility is effective when an end-to-end path is undesirable due to one particular segment of the path. For example, the path could contain a low capacity link, a high delay link, or a point of congestion. In these cases, routing around the problem link or router does not require direct knowledge of the route. By trying out a few tag values, the source network is likely to find a better path.

Tagging is quite scalable in the control plane since intermediate networks do not need to disseminate extra information or store network-level paths. An intermediate network can merely exploit the path diversity inside its own domain. There is little extra data-plane overhead, since the tag can use some rarely used bits in the existing IP header [94]. The extra data-plane overhead only comes from an ISP processing the received tag, and directing the packet onto an alternate path based on its tag value. Although tagging imposes less overhead than forwarding through a deflection point, it may require business relationships between the stub network and multiple ISPs. The incentives for honoring the tags are the most obvious in the context of hosts served by a single ISP.

2.6 Conclusions

The ability to forward traffic on multiple paths would be useful for customizing paths for different applications, improving reliability, and balancing load. Yet Internet-wide multipath routing remains elusive, due to scalability and economic challenges.

This chapter surveys a variety of deployed and incrementally deployable techniques which achieve flexible multipath routing. Routers already have data-plane support for forwarding on alternate paths through tunneling: encapsulation and explicit routing, though such techniques should be used in moderation for scalability reasons. We believe flowlet-cache is the most accurate and scalable technique for fine-grained traffic division. To access more end-to-end paths, stub networks can continue the trend of multihoming and extend it to perform fine-grained load balancing. Inside an ISP, multi-topology routing and forwarding on "downward" paths are both light-weight and easily deployable methods to leverage internal path diversity. Finally, we argue that deflecting packets at the network layer is a promising way to access more end-to-end paths with limited cooperation between networks, though new business models are needed to enable inter-network cooperation.

Though outside the scope of this dissertation, we believe that more research could be done to better quantify the trade-off between overhead and performance for the more heavy-weight solutions, including end-to-end signaling techniques [10, 23] not surveyed in this chapter. As technology advances, routers may become more capable of handling the overhead, making a wider range of solutions viable in practice. In addition, the economic incentives for providing value-added services will likely grow in the future and hopefully motivate the creation of new inter-network business models that enable Internet-wide multipath routing.

Chapter 3

Can Congestion Control and Traffic Engineering Be at Odds?

3.1 Introduction

In the Internet today, end hosts running the Transmission Control Protocol (TCP) adapt their sending rates in response to network congestion. Separately, network operators monitor their networks for signs of overloaded links and adapt the routing of traffic to alleviate congestion, in a process known as traffic engineering. TCP congestion control assumes that the network paths do not change, and traffic engineering assumes that the offered traffic does not change. Due to the layered network architecture, congestion control and traffic engineering operate independently, though their individual decisions are inevitably coupled. This chapter investigates whether the joint system is stable and optimal.

Traffic engineering and congestion control both solve, explicitly or implicitly, optimization problems defined for the entire network. Traffic engineering consists of collecting measurements of the traffic matrix—the observed load between each pair of edge routers—and performing a centralized minimization of a cost function that considers the resulting utilizations on all links (*e.g.*, [29, 74]). In contrast, TCP congestion control can be viewed as *implicitly* solving an optimization problem in a distributed fashion (*e.g.*, [48, 59, 58, 82]), where the many variants of TCP differ in the shape of user utility as a function of the source rate.

Previous analysis of congestion control and traffic engineering used congestion price as link weights (*e.g.*, [89, 38]) rather than modeling the current traffic engineering practices. In this chapter, we use the established optimization models (*e.g.*, [29, 74, 48, 59, 58, 82]) to study the interaction between traffic engineering and congestion control, and examine the following key questions through both analysis and simulation:

- **Stability**: Do the joint dynamics of congestion control and traffic engineering converge to an equilibrium?
- **Optimality**: If the joint system does converge, does the equilibrium maximize the aggregate user utility, over both the routing parameters and source rates?
- **Better design**: Can we modify the current system to guarantee stability and optimality?

In our joint congestion control and traffic engineering (CC-TE) model, TCP congestion control converges under a fixed routing configuration, before any routing changes are made. From our analysis and simulation experiments, we obtain the following insights:

- **Confirming the intuition of network operators**: Our simulation results show the CC-TE model is stable for a variety of topologies.
- **Tension between performance and robustness**: A modified CC-TE model is provably stability and optimality (Theorem 1), but at the cost of robustness.

The rest of the chapter is organized as follows. Section 3.2 introduces our joint congestion control and traffic engineering model. We simulate the CC-TE model in Section 3.3 and analyze a modified version in Section 3.4. Finally, Section 3.5 concludes the chapter.

3.2 Network Model

We focus on traffic engineering and congestion control in a single Autonomous System, where the operator has full view of the offered traffic load and a multipath routing model where traffic between source-destination pairs can be split arbitrarily across multiple paths. This is not the OSPF [68] or IS-IS [19] protocols used today, but can be implemented using MPLS [77] as explained in Chapter 2. The CC-TE Model considers average TCP traffic profiles.

Our notation follows the work in [89, 38]: in general, boldface are used to denote vectors and small letters are used to denote its components, *e.g.*, $\mathbf{x}$ with x_i as its i^{th} component; capital letters to denote matrices, *e.g.*, $\mathbf{R}$, or constants, *e.g.*, L and N. Superscript is used to denote vectors, matrices, or constants pertaining to source i, *e.g.*, $\mathbf{w}^i$ and $\mathbf{H}^i$. Also t is used to denote the iteration number, *e.g.*, $\mathbf{x}(t)$, in iterative algorithms. Table 3.1 presents a summary of the notation used.

Symbol	**Meaning**
x_i	Rate of source i.
R_{li}	Fraction of traffic on link l for source i.
w_j^i	The fraction of source i on its j^{th} path.
c_l	Capacity of link l.
$U_i(x_i)$	Utility function for source i.
α	Parameterizing the TCP utility function.
$U_\alpha(x)$	Utility function modeling α-fairness.
β	Step size of the TCP algorithm.
u_l	Utilization of link l.
$f(u_l)$	Cost function.

Table 3.1: Summary of notation used in Chapter 3

3.2.1 Network Topology and Routing

A network is modeled as a set of L bidirectional links with finite capacities $\mathbf{c} = (c_l, l = 1, \ldots, L)$, shared by a set of N source-destination pairs, indexed by i; we often refer to a source-destination pair simply as "source i." [1]

We consider a traffic engineering model for best-effort packet-switched networks that closely reflects the operational practices in Internet Service Provider (ISP) backbones [29, 74]. We represent the current routing through a matrix R_{li} that captures the fraction of i's flow that traverses each link l; as such, we do not explicitly model the assignment of link weights, which has been explored in depth in previous work [29, 74]. The operators measure the offered load between each ingress-egress pair x_i. Based on the known network topology and the traffic matrix, the operators try to find the best routing matrix $\mathbf{R}$ to minimize network congestion[2].

For a given routing configuration, the utilization of link l is $u_l = \sum_i R_{li} x_i / c_l$. To penalize routing configurations that congest the links, candidate routing solutions are evaluated based on a cost function $f(u_l)$ that is strictly convex and increasing. In a recent comparison study [11], the cost function we consider is found to be the best network-wide traffic-engineering objective for a range of traffic conditions and performance metrics. The following optimization problem over $\mathbf{R}$, for fixed $\mathbf{x}$ and $\mathbf{c}$, captures the traffic-engineering practices:

$$\text{minimize} \quad \sum_l f\left(\sum_i R_{li} x_i / c_l\right). \tag{3.1}$$

This optimization problem avoids solutions that operate near the capacity of the links and shifts flows to less utilized links where they can increase more freely. In

[1]Index i here refers to a TCP session between two physical nodes in a topology where there could be multiple sessions between two physical nodes.

[2]By focusing on the operational practices in IP networks, our model differs substantially from earlier work on quality-of-service routing in connection-oriented networks (*e.g.* [20, 52] and references therein), where arriving connections are routed dynamically and each link performs admission control.

practice, the network operators often use a piecewise-linear f for faster computation time [29, 74].

3.2.2 TCP Congestion Control

While the various TCP congestion-control algorithms were originally designed based on engineering heuristics, recent work, such as those surveyed in [58, 82], has shown through reverse engineering that they implicitly solve a convex optimization problem in a distributed fashion. Consider a network where each source i has a utility function $U_i(x_i)$ as a function of its total transmission rate x_i. The basic network utility maximization problem over source rate vector $\mathbf{x}$, for a given *fixed* routing matrix $\mathbf{R}$, is:

$$\begin{array}{ll} \text{maximize} & \sum_i U_i(x_i) \\ \text{subject to} & \mathbf{Rx} \preceq \mathbf{c}. \end{array} \tag{3.2}$$

The goal is to maximize aggregate user utility by varying $\mathbf{x}$ (but not $\mathbf{R}$), subject to the linear flow constraint that link loads cannot exceed capacity. TCP congestion-control algorithms implicitly solve (3.2), with different TCP variants maximizing different (increasing and concave) utility functions.

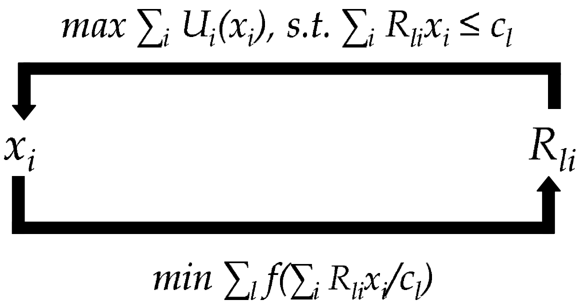
It is well-known that the utility functions can be picked based on several different grounds. First, a utility function can capture a user's degree of satisfaction with a particular throughput. Second, a utility function can be viewed as a measure of the elasticity of the traffic. Third, the aggregate utility captures the efficiency of the system in allocating bandwidth to the traffic. Fourth, some utility functions can lead to fair resource allocation. A particular family of widely-used utility functions is parameterized by $\alpha \geq 0$ [65]:

$$U_\alpha(x) = \begin{cases} \log x, & \alpha = 1 \\ (1-\alpha)^{-1}x^{1-\alpha}, & \alpha \neq 1. \end{cases} \tag{3.3}$$

Maximizing these α-fair utilities over linear flow constraints leads to rate-allocation vectors that satisfy the definitions of α-fairness in the economics literature.

The notion of α-fairness from [65] led to many TCP variants with different α-fairness interpretations. A utility function with $\alpha = 2$ was linked to TCP Reno. Through reverse engineering, TCP Vegas can be interpreted as $\alpha = 1$ [59], as can FAST [90]. XCP [47] is shown to be maximizing for U_α as $\alpha \to \infty$ in the single-link case. One exception to this family of α-fair utility functions is TCP Tahoe, which has been reverse engineered to be maximizing the utility function $U(x) = \arctan x$ [58].

3.2.3 Traffic Engineering Model of Joint System



$max \sum_i U_i(x_i)$, s.t. $\sum_i R_{li} x_i \le c_l$
x_i
R_{li}
$min \sum_l f(\sum_i R_{li} x_i / c_l)$

Figure 3.1: A detailed view of the TE Model of joint congestion control and routing system.

Our **TE Model of the joint congestion control and routing system** has two steps in a feedback loop, as shown in Figure 3.1. At time $t+1$, the congestion-control step computes new source rates based on the routing configuration from time t:

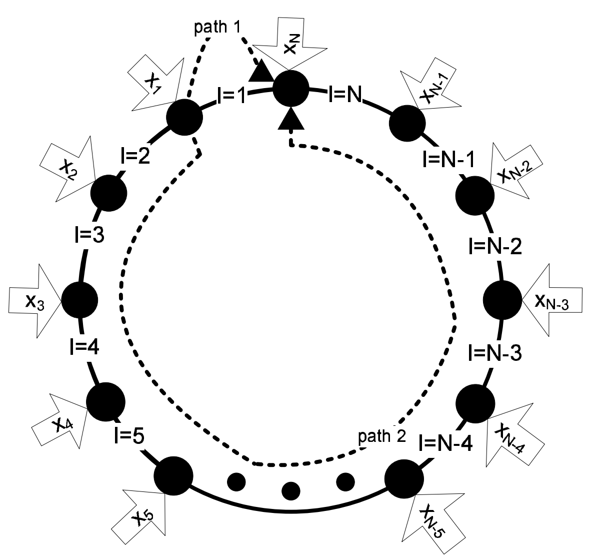
$$\mathbf{x}(t+1) = \text{argmax}_{\mathbf{x}} \sum_i U_i(x_i), \text{ subject to } \mathbf{R}(t)\mathbf{x} \preceq \mathbf{c}. \tag{3.4}$$

Then the routing step computes new paths based on the source rates:

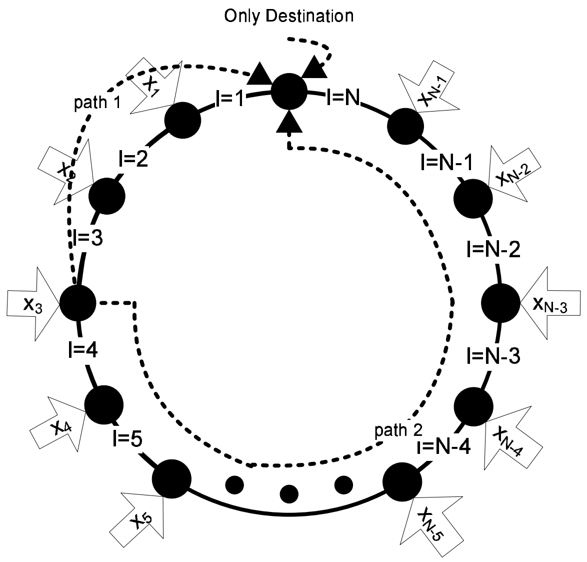
$$\mathbf{R}(t+1) = \text{argmin}_{\mathbf{R}} \sum_l f\left(\sum_i R_{li} x_i(t+1)/c_l\right). \tag{3.5}$$

The iterations of (3.4,3.5) repeat over time, with congestion control adapting the source rates to the new routes, and traffic engineering adapting the routes to the measured traffic.

3.3 Simulation of The TE Model

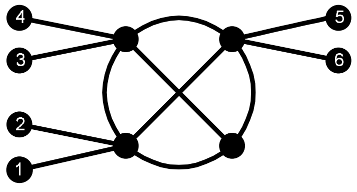


(a) N nodes, N sources

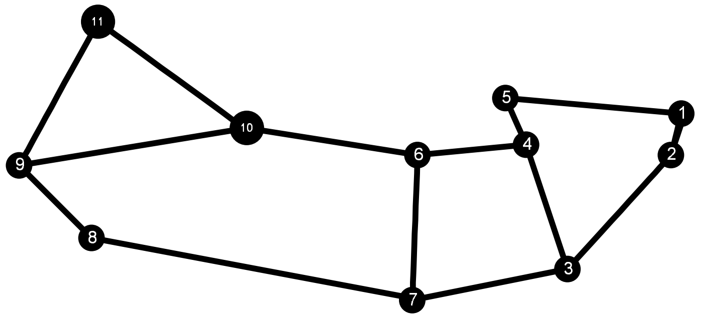


(b) N nodes, 1 destination

Figure 3.2: Two N-node ring topologies with different traffic patterns.



(a) Access-Core topology



(b) Abilene topology

Figure 3.3: Two realistic topologies.

We first illustrate some interesting numerical observations before presenting theorems on stability and optimality in the next section. Our numerical experiments use a combination of the Matlab and MOSEK [66] environments.

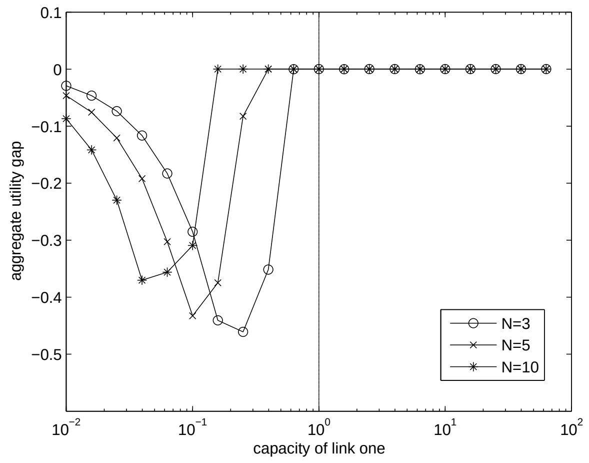
3.3.1 Simulation Set-up

We evaluate two variants of TCP congestion control: $\alpha = 2$ (*e.g.*, TCP Reno) and $\alpha = 1$ (*e.g.*, TCP Vegas). For the cost-function $f(u_l)$, we use an exponential function, which is the continuous version of the function used in various studies of traffic engineering [29, 74].

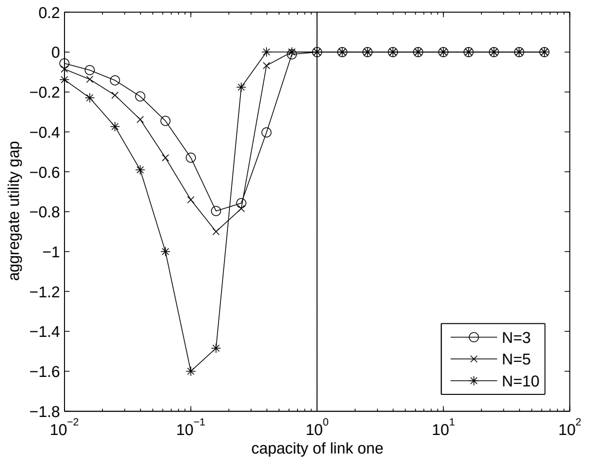
Our initial experiments evaluate a simple N-node ring topology, where we can easily scale the size of the network. To evaluate the influence of the traffic patterns, we consider two scenarios. In the first scenario, each node is a source sending to its clockwise neighbor; each source has two possible paths: a direct one-hop path and an indirect $(N-1)$-hop path. In the second scenario, node 1 is the destination and the remaining $N-1$ nodes are sources; each source x_i has an i-hop path and an $(N-i)$-hop path. Our experiments vary the number of nodes N and the capacity of link 1 (between nodes 1 and N).

To study realistic topologies with greater path diversity, we also experiment with the two networks in Figure 3.3. On the left is a tree-mesh topology, which is representative of a common network structure. In the middle is a full mesh representing the core of the network with rich connectivity. On the edge are three access tree subnetworks. Of the twelve possible source-destination pairs, $1-3$, $1-5$, $2-4$, $2-6$, $3-5$, and $4-6$ are chosen, and for each source-destination pair, the three minimum-hop paths are chosen as possible paths. On the right is the Abilene backbone network [6]. Of the many possible source-destination pairs, we choose $1-6$, $3-9$, $7-11$, and $1-11$. For each source-destination pair, we choose the four minimum-hop paths as possible paths. For the access-core and Abilene topologies, the simulations assume the link capacities follow a truncated (so as to avoid negative values) Gaussian distribution, with an average of 100 and a standard deviation that varies from 0 to 50. We simulate twenty random configurations for each value of the standard deviation. In all experiments, we start with an initial routing configuration that splits traffic evenly among the paths for each source-destination pair.

3.3.2 Suboptimality Gap Simulations

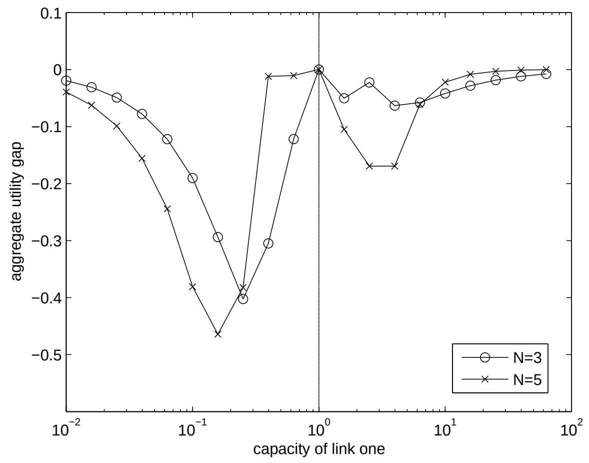


(a) $\alpha = 1$ (*e.g.*, TCP Vegas)

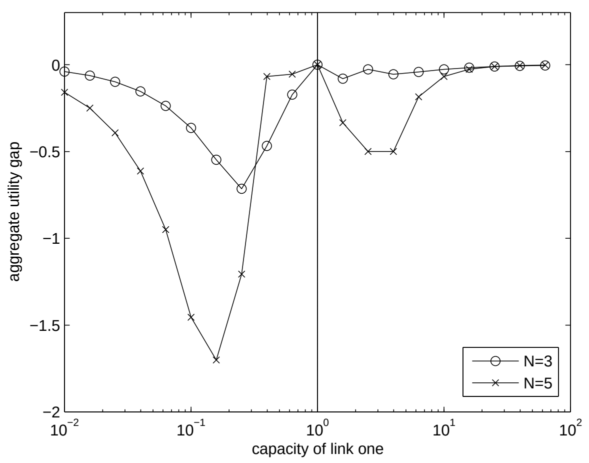


(b) $\alpha = 2$ (*e.g.*, TCP Reno)

Figure 3.4: Aggregate utility gap for the N-node, N-source ring.



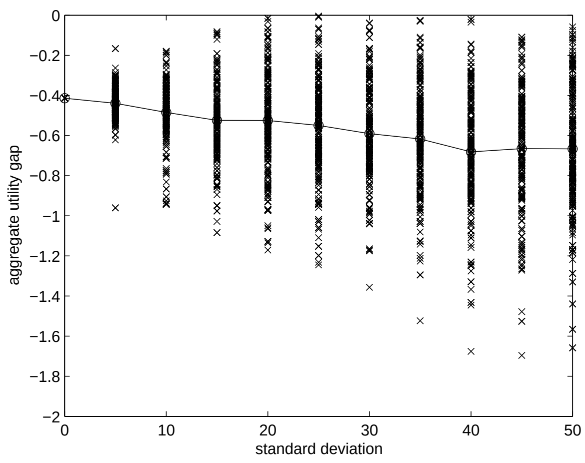
(a) $\alpha = 1$ (*e.g.*, TCP Vegas)



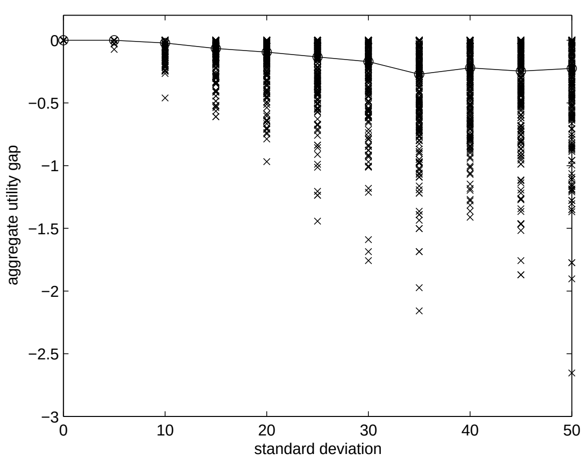
(b) $\alpha = 2$ (*e.g.*, TCP Reno)

Figure 3.5: Aggregate utility gap for the N-node, 1-destination ring.

Given the structure of (3.2), it is natural to wonder if the interaction of congestion control and traffic engineering maximizes aggregate user utility. Previous work [57, 89]



(a) Access-core topology



(b) Abilene topology

Figure 3.6: Aggregate utility gap for two realistic topologies (with $\alpha = 1$). A -x- marker denotes an individual test point and a -o- marker denotes the average.

has proposed the following joint optimization problem:

$$\begin{array}{ll} \text{maximize} & \sum_i U_i(x_i) \\ \text{subject to} & \mathbf{Rx} \preceq \mathbf{c},\ \mathbf{x} \succeq \mathbf{0} \end{array} \tag{3.6}$$

where *both* **R** and **x** are variables.

Our experiments quantify the gap in aggregate utility between that at the equilibrium of the joint system and the optimal aggregate utility of (3.6). Such a gap between the achieved utility and the maximum utility signifies a loss in user satisfaction, and often implies also a loss in fairness or efficiency. Table 3.2 summarizes the key observations from the numerical experiments.

Figure(s)	**Key Message**
3.4a versus 3.5a	Traffic pattern can have a significant effect.
3.4a versus 3.4b 3.5a versus 3.5b	TCP variants give the same trend.
All figures	Relatively small suboptimality gap.
All figures	Homogeneity minimizes suboptimality gap.

Table 3.2: Summary of results on suboptimality gap.

In Figure 3.4, we vary the capacity of link 1 and plot the gap in aggregate utility for ring topologies with three, five, and ten nodes, where each node communicates with its clockwise neighbor. The two graphs plot results for $\alpha = 1$ (*e.g.*, TCP Vegas) and $\alpha = 2$ (*e.g.*, TCP Reno), respectively. The graphs show trends that are very similar across a range of topology sizes, suggesting that the number of sources alone does not have a significant influence on the suboptimality gap. Similarly, the two TCP variants lead to very similar results.

The vertical line in the middle of the two graphs highlights the configuration where all links have unit capacity. The suboptimality gap is zero for a wide range of capacity configurations. When one link has much lower capacity than the other links, a suboptimality gap emerges. This occurs because the traffic-engineering step in the joint system stops making use of this low-capacity link, since the penalty for placing even a small amount of load on this link exceeds the cost of forcing the traffic on a longer path that places load on multiple links. When link 1 has an extremely low capacity, even the optimal solution cannot place much traffic on this link, leading to a small suboptimality gap.

The graphs in Figure 3.5 confirm that variations in link capacities affect the suboptimality gap. These graphs evaluate the N-node ring with one destination node, for two values of N and two TCP variants. In contrast to Figure 3.4, having either a smaller or a larger capacity on link 1 leads to a suboptimality gap. This is not surprising because link 1 is a bottleneck link for this traffic pattern. If the link has a small capacity, the traffic-engineering step does not make use of the link, making the left part of these curves closely resemble the plots in Figure 3.4. If the link has a high capacity, the traffic-engineering step tries to direct more sources through the link; however, this is not the best solution when the capacity of link 1 is just slightly larger than the other links because traffic traverses longer paths, placing load on a larger number of links. Comparing Figures 3.4 and 3.5 illustrates the important role the traffic pattern plays in determining whether the joint system successfully maximizes aggregate utility.

The graphs in Figure 3.6 illustrate the effects of a variation in link capacities on realistic topologies. We show how the suboptimality gap depends on the standard deviation of the link capacities, which are all varied according to a truncated Gaussian distribution. We plot separate points for each of the 500 experiments for each value of standard deviation, as well as a curve that highlights the mean values. The trend that a more homogeneous capacity distribution (smaller standard deviation) leads to a smaller suboptimality gap exists, but it is much more subdued than in the ring topology and it is dominated by the variance. This suggests that, with realistic topologies, the relationship between link capacity and utility gap is more complex. One possible explanation is that the bottleneck link on each path is what matters, and, while that is easily correlated with varying a single link in the ring topology, the effect is coupled in a more complex topology. In addition, for the Abilene topology, a suboptimality gap exists even for a homogenous capacity distribution. While the results for the ring topology suggest that network operators could favor network configurations that enable (near) optimal solutions, the results for the access-core and Abilene networks suggest this may be quite challenging for more realistic topologies.

3.4 Analysis of the TE Model

Our simulations showed that the TE Model (3.4,3.5) is stable and close to optimal for a range of topologies. We speculate, but have not yet shown it is provably stable for general topologies. In this section, we show how relaxing the constraint for (3.4) can lead to a *provably stable and optimal* joint system. By imposing conditions on f, the joint system can be *brought* arbitrarily close to optimality with respect to (3.6). This comes at the expense of robustness, however, and is not recommended for implementation.

Theorem 3.4.1. *If (3.4) is replaced by the following unconstrained problem:*

$$\mathbf{x}(t+1) = argmax_{\mathbf{x}\succeq 0} \sum_i U_i(x_i) - \sum_l f\left(\sum_i R_{li}(t)x_i/c_l\right), \tag{3.7}$$

then the TE Model converges to the optimum of

$$argmax_{(\mathbf{x},\mathbf{R})} \sum_i U_i(x_i) - \sum_l f\left(\sum_i R_{li}(t)x_i/c_l\right), \tag{3.8}$$

for sufficiently concave utilities (i.e., sufficiently elastic traffic): $U_i''(x_i) \leq -\frac{U_i'(x_i)}{x_i}$. In particular, it converges for α-fair utilities when $\alpha \geq 1$ and for arctan *utility of TCP Tahoe.*

Proof. First we show that the joint traffic engineering and modified congestion control system (3.5,3.7) is equivalent to a successive, alternating optimization of (3.8) over $\mathbf{R}$ and then $\mathbf{x}$. Then we provide a sufficient condition to guarantee convergence to optimality. Finally the condition is examined for α-fair utilities and arctan utility.

Consider the unconstrained minimization of

$$g(\mathbf{x},\mathbf{R}) = -\sum_i U_i(x_i) + \sum_l f\left(\sum_i R_{li}x_i/c_l\right), \tag{3.9}$$

which is equivalent to (3.8). The two steps in the alternating optimization method of Gauss-Seidel algorithm [16] are as follows:

$$\begin{aligned}
\mathbf{x}(t+1) &= \text{argmin}_{\mathbf{x}\succeq 0} - \sum_i U_i(x_i) + \sum_l f(\sum_i R_{li}(t)x_i/c_l) \\
\mathbf{R}(t+1) &= \text{argmin}_{\mathbf{R}} g(\mathbf{x}(t+1), \mathbf{R}(t)) \\
&= \text{argmin}_{\mathbf{R}} \sum_l f(\sum_i R_{li}(t)x_i(t+1)/c_l).
\end{aligned}$$

The minimization of $g(\mathbf{x},\mathbf{R})$ over $\mathbf{R}$ is clearly equivalent to (3.1).

So far we have constructed an optimization problem (minimization of $g(\mathbf{x},\mathbf{R})$)

whose Gauss-Seidel solution algorithm is equivalent to the system model of joint traffic engineering and modified congestion control. Now we will examine the conditions for convergence of this Gauss-Seidel Algorithm. From [16], the Gauss-Seidel Algorithm will converge to the minimizer of g if g is bounded from below, differentiable, marginally strictly convex in $\mathbf{x}$ and $\mathbf{R}$, and jointly convex in $\mathbf{x}$ and $\mathbf{R}$.

The first three conditions are already satisfied through the constraints placed in the system model definition. Condition 1 is satisfied since $\mathbf{x} \succeq \mathbf{0}$, $\mathbf{R} \succeq \mathbf{0}$ by definition. Condition 2 is satisfied since U and f are differentiable, so is g. The third condition is satisfied since U is strictly concave in $\mathbf{x}$, and f is marginally strictly convex in $\mathbf{x}$ and $\mathbf{R}$. The last condition is not satisfied in general since the function $f(\sum_l R_{li}x_i/c_l)$ is not jointly convex in $\mathbf{R}$ and $\mathbf{x}$.

In order to satisfy the condition on joint convexity in $\mathbf{x}$ and $\mathbf{R}$, consider a log change of variable. Let $\tilde{x_i} = \log x_i$, $\tilde{R_{li}} = \log R_{li}$, then $R_{li}x_i = \exp(\tilde{R_{li}} + \tilde{x_i})$. With the change of variable, it can be readily verified that f is still jointly convex in $\tilde{x_i}$ and $\tilde{R_{li}}$, but the utility function may no longer be concave in $\tilde{\mathbf{x}}$. If the utility function is concave in $\tilde{\mathbf{x}}$, then g would be strictly convex in $\tilde{\mathbf{x}}$ since f is strictly convex in $\tilde{\mathbf{x}}$. Denote the new utility function (after the log change of variable) as $W_i(\tilde{x_i})$. A sufficient condition for convergence of the Gauss-Seidel algorithm is for W to be concave in $\tilde{\mathbf{x}}$. A simple derivation shows that such a condition reduces to the following simple bound on the curvature of the utility function: $U_i^{''}(x_i) \leq -U_i^{'}(x_i)/x_i$.

Now we specialize to the α-fairness model for U which covers TCP Reno (currently deployed) and several proposed variants. In this case, $W_\alpha(\tilde{\mathbf{x}})$ can be written as follows:

$$W_\alpha(\tilde{\mathbf{x}}) = \begin{cases} \tilde{\mathbf{x}}, & \alpha = 1 \\ (1-\alpha)^{-1}\exp(\mathbf{x})^{1-\alpha}, & \alpha \neq 1. \end{cases} \tag{3.10}$$

Examining $W^{''}(\tilde{\mathbf{x}})$ shows that $W(\tilde{\mathbf{x}})$ is concave for $\alpha \geq 1$.

Finally, TCP Tahoe is examined. Recall that $U(\mathbf{x}) = \arctan(\mathbf{x})$ for TCP Tahoe, and $W^{''}(\tilde{\mathbf{x}}) = \arctan(\mathbf{x})$. It follows that $W^{''}(\tilde{\mathbf{x}}) = (\exp(\tilde{\mathbf{x}}) - \exp(3\tilde{\mathbf{x}}))/(1+\exp(2\tilde{\mathbf{x}}))^2$ and W is concave. Therefore, convergence of (3.8) is guaranteed for TCP algorithms with $\alpha \geq 1$ and TCP Tahoe. $\square$

A more general version of Theorem 1 states that convergence of the TE Model is guaranteed under the following two conditions.

Condition 1: g is marginally strictly convex in $\tilde{\mathbf{x}}$. Denote $f(\exp\tilde{\mathbf{x}})$ as $F(\tilde{\mathbf{x}})$, then algebraic manipulation shows this is equivalent to:

$$W'' < \gamma \sum_l F''(\tilde{\mathbf{x}}).$$

It is known that $F''(\tilde{\mathbf{x}}) > 0$, and in the proof of Theorem 1, the simple lower bound of 0 is used. If $W'' > 0$, γ is sufficiently large and f is sufficiently convex, $g(\mathbf{x})\mathbf{R})$ is still marginally strictly convex in $\tilde{\mathbf{x}}$.

Condition 2: g is jointly convex in $\tilde{\mathbf{x}}$ and $\tilde{R}$. This condition can be written mathematically as

$$v^T \bigtriangledown^2 g(\exp(\tilde{\mathbf{x}}), \exp(\tilde{R}))v \geq 0, \forall v.$$

Algebraic manipulation shows:

$$\begin{aligned} &v^T \bigtriangledown^2 g(\exp(\tilde{\mathbf{x}}), \exp(\tilde{R}))v = \\ &-v_1^T \bigtriangledown^2 W(\tilde{\mathbf{x}})v_1 + \gamma v^T \bigtriangledown^2 f(\exp(\tilde{\mathbf{x}}), \exp(\tilde{R}))v, \end{aligned} \tag{3.11}$$

where v_1 is the first N elements of v. Since f is jointly convex in $\tilde{\mathbf{x}}$ and $\tilde{R}$, we do require

$$v_1^T \bigtriangledown^2 W(\tilde{\mathbf{x}})v_1 \leq \gamma v^T \bigtriangledown^2 f(\exp(\tilde{\mathbf{x}}), \exp(\tilde{R}))v.$$

This can be rewritten as

$$v_1^T \bigtriangledown^2 W(\tilde{\mathbf{x}})v_1 \leq \gamma v_1^T \bigtriangledown^2 f(\exp(\tilde{\mathbf{x}}), \exp(\tilde{R}))v_1.$$

Again, for sufficiently large γ and/or sufficiently convex f, $g(\mathbf{x})\mathbf{R})$ is jointly convex in $\tilde{\mathbf{x}}$ and $\tilde{R}$ even if $U(\exp(\tilde{\mathbf{x}}))$ itself is not concave in $\tilde{\mathbf{x}}$.

Theorem 3.4.2. *The cost function f can be chosen so that (3.8) is arbitrarily close to (3.6). In this asymptote, the TE model (3.4, 3.5) converges to the global optimum $(\mathbf{R}^*, \mathbf{x}^*)$ of (3.6).*

Proof. From Theorem 1, the joint system (3.5,3.7) converges to (3.8). If (3.8) is arbitrarily close to (3.6), and (3.4) to (3.7) then (3.4, 3.5) will converge to the solution of (3.6).

By the penalty-function method (see [14] for details), there exists a penalty function P and a constant γ so that (3.6) is equivalent to (3.12):

$$\text{maximize}_{(\mathbf{x},\mathbf{R})} \sum_i U_i(x_i) - \gamma \sum_l P\left(\sum_i R_{li} x_i / c_l\right), \tag{3.12}$$

and (3.4) is equivalent to (3.7) provided that γ is sufficiently large, and P is convex, increasing, and zero for $\mathbf{Rx} \preceq \mathbf{c}$ (positive otherwise). Essentially, in (3.12) and (3.7), $-\gamma \sum_l P\left(\sum_i R_{li} x_i / c_l\right)$ in the objective function replaces the constraint $\mathbf{Rx} \preceq \mathbf{c}$ when γ is sufficiently large. If the operators choose a cost function f which is zero until $u_l = 1$, and sufficiently large afterwards, then it can match γP exactly. Such a function can be approximated as closely as desired, *e.g.* by choosing sufficiently large n in $f(u_l) = n u_l^n$ where $n > 0$ is a parameter that modulates the approximation accuracy with respect to the original TCP model. □

Simulations confirm that the joint model converges arbitrarily closely to the optimum of (3.6) when the TE penalty function in (3.5) is replaced by $f(u_l) = n u_l^n$ and n is allowed to go to infinity. For example, Figure 3.7 plots the utility gap as a function of n for the Abilene topology with a standard deviation of 50. For larger values of n, the utility gap grows arbitrarily small, in contrast to the large gap seen earlier in Figure 3.6(b). Although larger values of n narrow the optimality gap, we find that

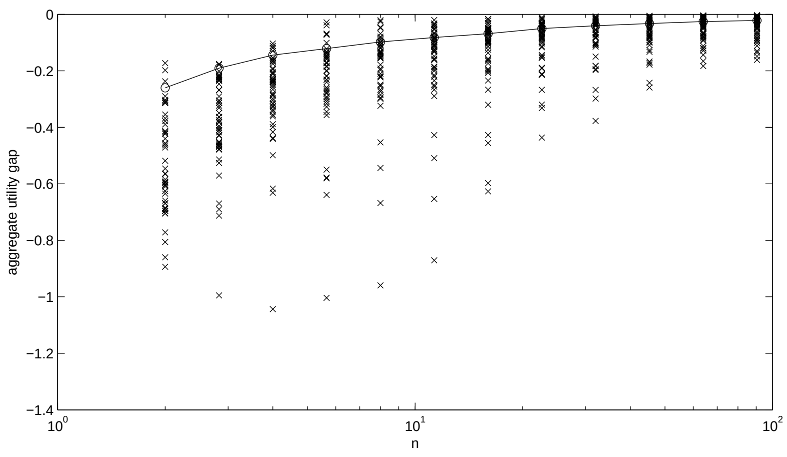


Figure 3.7: Using the function $f(u_l) = nu_l^n$ in the TE model (3.5) and alternating iterations with TCP (3.4) results in a arbitrarily small gap in utility as $n \to \infty$.

the joint system requires more iterations for convergence, leading to a sometimes substantial increase in convergence time. For simpler, more uniform topologies, we find that even small values of n suffice to close the optimality gap. These plots are omitted due to space limitations.

While modifying the traffic-engineering penalty function f will allow us to be arbitrarily close to the optimum of (3.6), it will also drive the network increasingly close to a solution with multiple links operating near capacity. This is a fragile point of operation for the network since a small burst in traffic would cause the traffic on certain links to exceed capacity. Once the traffic exceeds capacity, congestion is inevitable and so is the subsequent packet loss and delay increase. Here we have a trade-off between stability and optimality (with respect to (3.6)) on the one hand and robustness (with respect to short, high-volume traffic bursts) on the other.

3.5 Conclusion

In this chapter, we have studied the interaction of congestion control and traffic engineering from a network operator's perspective. Congestion control and traffic engineering both try to make efficient use of link bandwidth to improve network performance for end users. In today's IP networks, however, these two mechanisms operate independently, though they are coupled because they both adapt to network congestion. In this chapter, we find through simulation that congestion control and traffic engineering work effectively together to reach a stable equilibrium, with a small optimality gap. A modification to the operator's cost function leads to a provably stable and optimal system, but at the cost of robustness. This highlights the potential tension between performance and robustness, which the next chapter addresses.

Chapter 4

TRUMP: TRaffic-management Using Multipath Protocol

4.1 Introduction

Today's traffic management has three players: users, routers, and operators. Users run congestion control to adapt their sending rates at the edge of the network. Inside a single Autonomous System (AS), routers run shortest-path routing based on link weights. Operators monitor the network for congestion, and tune link weights to direct traffic away from congested links [30]. The current division of labor between the three players slowly evolved over time without any conscious design, resulting in a few shortcomings. First, the previous chapter observed today's traffic engineering and congestion control interact stably, but not always efficiently. Modifying the traffic engineering objective can lead to maximizing aggregate utility, but at the cost of robustness. Second, tuning link weights is an indirect way to control traffic flow through a network; further, the link-weight setting problem is NP-hard, forcing operators to resort to heuristics that can lead to highly suboptimal solutions [30]. Finally, since this offline optimization occurs at the timescale of hours, it does not adapt to changes in the offered traffic, causing an inefficient use of the underlying resources. These shortcomings motivate the need to rethink the division of labor between these three players.

In this chapter, we rethink Internet traffic management using optimization decomposition as a foundation. Optimization decomposition is the process of decomposing a single optimization problem into many sub-problems. The resulting distributed algorithm is provably stable and optimal, but two challenges remain. First, any mathematical modeling makes simplifying assumptions. Second, while multiple decomposition methods exist, it is unclear how to compare them. Our contributions are two-fold:

- **Protocol Design using Decompositions:** We demonstrate how to create a practical network protocol by deriving multiple distributed algorithms, comparing their practical properties, and synthesizing their best features into a practical protocol.

- **Redesigned Traffic Management:** We introduce TRUMP: an easy to manage distributed protocol, that performs well for diverse topologies, capacities, feedback delays and traffic loads.

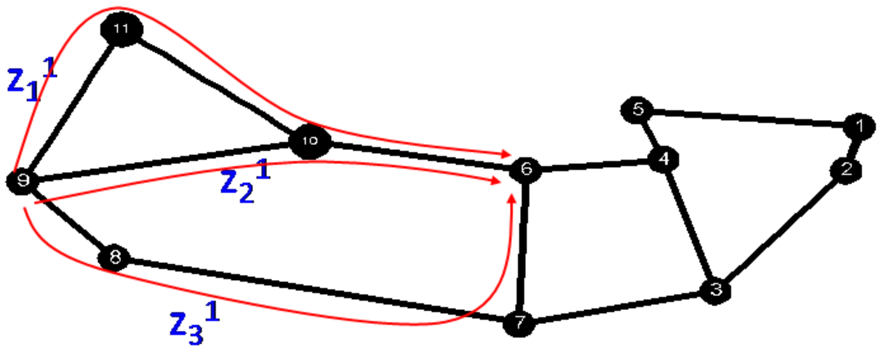


Figure 4.1: Three paths between source node 9 and destination node 6.

In our *top-down redesign* of traffic management, we start by selecting an intuitive and practical objective function in Section 4.2 that balances the objectives of users and operators. Using optimization decomposition techniques discussed in [70], Section 4.3 derives four specific distributed solutions where sources adapt sending rates on *multiple paths* to a destination. This is illustrated in Figure 4.1, source node 9 computes its sending rate on each of its three paths to destination node 6, based on feedback regarding the path congestion conditions. An advantage of our distributed algorithms is that they adapt at a single timescale (on the order of RTTs), and is able to respond quickly to traffic shifts. The four algorithms differ in the computation of path rates and congestion feedback. Optimization theory guarantees that these algorithms converge to a stable and optimal point, while simulations allow us to compare rate of convergence and robustness to tunable parameters in Section 4.4. Although these distributed algorithms work well, they can be sensitive to tunable parameters.

We combine the best features of each of these algorithms to construct a simple TRaffic-management Using Multipath Protocol (TRUMP) in Section 4.5. TRUMP converges faster than the four algorithms presented in Section 4.3, and has the fewest tunable parameters. Although TRUMP is not derived from a particular optimization decomposition, we are able to prove its convergence when the network is tuned to have low packet loss. As with any mathematical modeling, the TRUMP algorithm leaves many protocol details unspecified such as how to handle heterogenous feedback delays. We use engineering intuition to address these details in Section 4.5.

In Section 4.6, the TRUMP protocol is evaluated using packet-level simulations with a wide range of topologies and traffic loads. We use a simple heuristic to set TRUMPs parameter such that it converges smoothly for a wide range of topologies, capacities, feedback delays, and traffic loads. When many flows share the same bottleneck link, there is a small amount of packet loss during convergence, but TRUMP still converges within a few RTTs. In contrast, the other distributed solutions become unstable when such links exist. We also study the impact of number of paths on the performance of TRUMP. First, TRUMP is more likely to split traffic over multiple paths when there are fewer concurrent source-destination pairs. Second, while TRUMP can achieve higher throughput if all sources can access two paths rather than a single path, though access to more than two paths does not provide significant gains. This chapter discusses related work in Section 4.7 and concludes in Section 4.8.

4.2 Choosing An Objective Function

In this section, we use optimization as a modeling language to formalize traffic management. Every optimization problem consists of an objective function, a constraint set and variables. For traffic management, by having both routing and source rate as optimization variables, we have the most flexibility in resource allocation, such an idea was first proposed in Section 6.5.1. of [15] (we clarify the similarities and difference in 4.2.2). In our problem, the constraint is that link load does not exceed capacity. The objective function remains to be designed. We first propose an objective that maximizes aggregate user utility, but simulations reveal the solution converges slowly and is sensitive to stepsize. In addition, maximizing user utility leads to bottlenecks in the network, making the network fragile to traffic bursts. To address these practical challenges, we design an objective which balances maximizing user utility with minimizing operator's cost function. Table 4.1 presents a summary of the notation used.

4.2.1 Maximizing Aggregate Utility: DUMP

One natural objective for the traffic management system is to maximize aggregate user utility, where utility $U_i(x_i)$ is a measure of "happiness" of source-destination pair i (as introduced in the previous chapter) as a function of the total transmission

Symbol	**Meaning**
x_i	Rate of source i.
$U_i(x_i)$	Utility function for source i.
α	Parameterizing the TCP utility function.
$U_\alpha(x)$	Utility function modeling α-fairness.
R_{li}	Fraction of traffic on link l for source i.
c_l	Capacity of link l.
u_l	Utilization of link l.
$f(u_l)$	Cost function.
w	Weight between U and f
$\mathbf{H}$	Network topology as the set of all paths.
z_j^i	Rate of source i on its j^{th} path.
β_z	Step size for update of path rate.
y_l	Effective capacity on link l.
s_l	Feedback price on link l.
β_s	Step size for update of feedback price.
p_l	Consistency price on link l.
β_p	Step size for update of consistency price.
γ	Pacing parameter for TRUMP protocol.

Table 4.1: Summary of notation used in Chapter 4

rate x_i. U is a concave, non-negative, increasing and twice-differentiable function, *e.g.* $\log(x_i)$, that can also represent the elasticity of the traffic or determine fairness of resource allocation. This is the objective implicitly achieved by TCP congestion control today [48, 58]. We represent the routing by matrix R_{li} that captures the fraction of source i's flow that traverses link l, and we let c_l denote the capacity of link l. As proposed in [57, 89], the resulting optimization problem is:

$$\begin{array}{ll} \text{maximize} & \sum_i U_i(x_i) \\ \text{subject to} & \mathbf{Rx} \preceq \mathbf{c},\ \mathbf{x} \succeq \mathbf{0} \end{array} \tag{4.1}$$

where both R and x are variables.

A distributed solution to (4.1) can be derived through dual decomposition if (4.1) is a convex optimization problem. In its current form, (4.1) has a non-convex constraint set, which can be transformed into a convex set if the routing is allowed to be multipath. To capture multipath routing, we introduce z_j^i to represent the sending rate of source i on its jth path. We also represent available paths by a matrix $\mathbf{H}$ where

$$H_{lj}^i = \begin{cases} 1, & \text{if path } j \text{ of source } i \text{ uses link } l \\ 0, & \text{otherwise.} \end{cases}$$

$\mathbf{H}$ does not necessarily present all possible paths in the physical topology, but a subset of paths chosen by operators or the routing protocol. Then we can rewrite (4.1) as:

$$\begin{array}{ll} \text{maximize} & \sum_i U_i(\sum_j z_j^i) \\ \text{subject to} & \sum_i \sum_j H_{lj}^i z_j^i \leq c_l, \ \ \forall l. \end{array} \tag{4.2}$$

In this form, (4.2) is a convex optimization problem. A distributed solution to (4.2) can be derived using *dual decomposition* [57], where a *dual variable* is introduced to relax the capacity constraint. The resulting Dual-based Utility Maximizing Protocol (DUMP) is summarized in Figure 4.2. Similar to the reverse engineering of the congestion-control protocol in [58], $\mathbf{s}$ can be interpreted as link prices.

Feedback price update at link l:

$$s_l(t+1) = \left[s_l(t) - \beta_s(t) \left(c_l(t) - \sum_i \sum_j H_{lj}^i z_j^i(t) \right) \right]^+ ,$$

where β_s is the feedback price stepsize.

Path rate update at source i, path j:

$$z_j^i(t+1) = \text{maximize}_{z_j^i} \left(U_i \left(\sum_j z_j^i \right) - z_j^i \sum_l s_l(t) H_{lj}^i \right)$$

Figure 4.2: The DUMP algorithm.

Here t represents the iteration number and each iteration is at the same timescale as the longest Round Trip Time (RTT) of the network. At each link, s_l is updated based on the difference between the link load $\sum_i \sum_j H^i_{lj} z^i_j$ and the link capacity. As indicated by $[]^+$, s_l is only positive when the link load exceeds the link capacity, *i.e.* when the network is congested. Each source updates z^i_j based on explicit feedback from the links, in the form of feedback prices s_l. In particular, each source maximizes its own utility, while balancing the price of using path j. The path price is the product of the source rate with the price per load for path j (computed by summing s_l over the links in the path). DUMP is similar to the TCP dual algorithm in [58] except the local maximization is conducted over a *vector* $\mathbf{z}^i$, as opposed to only a scalar x_i, to capture the multipath nature of DUMP.

From optimization theory, certain choices of stepsizes, such as $\beta_s(t) = \beta/t$ where $\beta > 0$ is a constant, guarantee that DUMP will converge to the joint optimum as $t \to \infty$ [14]. However, such diminishing stepsize is difficult to implement in practice as it requires synchronization of time across the nodes, and particularly difficult to do with dynamic arrivals of new flows. Previous work indicates that even under the simplest of topologies and assuming greedy flows, DUMP has poor convergence behavior [57]; our own Matlab experiments [34] confirm this. When the stepsize is too large, DUMP will constantly overshoot or undershoot, never reaching the ideal utility. On the other hand, when the stepsize is too small, DUMP converges very slowly. Even at the optimal stepsize, DUMP only converges after about 100 iterations. This highlights that choosing an appropriate stepsize for DUMP is challenging.

4.2.2 New Objective for Traffic Management

Let us reflect for a moment on why DUMP has poor convergence behavior. If we look at the form for feedback price, we see it is only nonzero when links are overloaded, therefore, the feedback from the links is not fine-grained. This corresponds to the congestion control mechanism of TCP Reno where sources only reduce their sending rates once packets are already lost, causing the sawtooth behavior. In fact, the feedback price in DUMP has the same formulation as the congestion price in [58]. In addition, utility is only based on throughput, while having low delay is also important to traffic management. In addition, the authors of [35] suggest the network is driven to a solution where some links are operating near capacity when only utility is maximized. This is an undesirable operating point which is very fragile to traffic bursts. This indicates that maximizing the aggregate utility enhances performance of the individual users, but leaves the network as a whole fragile.

To avoid the poor convergence properties of DUMP, we look for an alternative problem formulation which also takes into account the operator's objective. Today, traffic engineering solves (3.1) as introduced in the previous chapter. f is a convex, non-decreasing, and twice-differentiable function that gives increasingly heavier penalty as link load increases, *e.g.* $e^{\sum_i R_{li}x_i/c_l}$. The intuition behind choosing this f is two-fold. First, f can be selected to model M/M/1 queuing delay. Second, network operators want to penalize solutions with many links at or near capacity and do not care too much whether a link is 20% loaded or 40% loaded [30]. If we solve (3.1) with both $\mathbf{x}$ and $\mathbf{R}$ as variables, then the solution would end up with zero throughput, which is also undesirable.

A better traffic management objective could be to combine performance metrics (users' objective) with network robustness (operator's objective), leading to the following formulation as a joint optimization over $(\mathbf{x}, \mathbf{R})$:

$$\begin{array}{ll} \text{maximize} & \sum_i U_i(x_i) - w \sum_l f(\sum_i R_{li}x_i/c_l) \\ \text{subject to} & \mathbf{Rx} \preceq \mathbf{c},\ \mathbf{x} \succeq \mathbf{0}. \end{array} \tag{4.3}$$

This objective favors a solution that strikes a trade-off between high aggregate utility and a low overall network congestion, to satisfy the need for performance and robustness. Similar problem formulations were proposed in [35, 33], though without

w. Here w is a parameter which adjusts the balance between the utility function and the cost function. When w is small, (4.3) is very close to (4.1) since the utility term dominates. When w is large, the solution is more conservative and avoids high link utilization. Today, operators perform traffic engineering by adjusting link weights depending on the instantaneous traffic load. In our case, they can adjust a single parameter w.

Aside from performance, fairness is another important consideration. From a theoretical perspective, the solution to (4.3) is α-*fair* as $w \to 0$, where α-fairness is defined in [65]. While this does not hold for general values of w, our experimental results in Section 4.6.6 are encouraging.

Before generating distributed solutions in Section 4.3, we first transform (4.3) to a convex optimization problem:

$$\begin{array}{ll} \text{maximize} & \sum_i U_i(\sum_j z_j^i) - w \sum_l f(y_l/c_l) \\ \text{subject to} & \mathbf{y} \preceq \mathbf{c}, \\ & y_l = \sum_i \sum_j H_{lj}^i z_j^i, \quad \forall l. \end{array} \tag{4.4}$$

Note that to decouple the objective which contains U (a per-source function) and f (a per-link function), we introduce an extra variable y_l to provide feedback before link load exceeds the actual capacity.

Our formulation (4.3) bears several similarities and differences to the classic joint optimal flow control and multipath routing presented in Section 6.5.1. of [15]. In both cases, there are two functions in the objective: a convex per link cost function, and a concave per source utility function. In (4.3), the cost function is a function of link utilization, where as [15] considers a function of link load. In (4.3), we consider the family of α-fair utility functions [65], whereas [15] has $-(a/x)^b$, where a and b are constants. In [15], rates are upper bounded by a target rate for a source-destination pair, there is no upper bound in (4.3). The formulation in both cases are general enough to be equivalent under certain conditions. The key difference lies in how the problem is solved. In [15], they analyze optimality conditions, and the existence of flow control depends heavily the upper bound for the throughput. Basically, the rate is equal to the desired rate until it exceeds the capacity of the system, in which case congestion control kicks in. In the next section, we derive *distributed* solutions to (4.3) using optimization decomposition.

4.3 Multiple Decompositions

In this section, we describe the distributed algorithms generated from optimization decompositions of (4.3) (the decomposition techniques are surveyed in [70, 48]). All four resulting algorithms update the path rates based on feedback prices from links. There are a number of other similarities between the four algorithms. First, the operations performed by links including measuring the link load present only a small overhead. Second, all four algorithms incur the same small message passing overhead: only the sum of the link prices on the end-to-end path needs to be communicated. Third, while computations can involve solving a local optimization problem and taking derivatives, U and f are twice differentiable, and therefore closed-form solutions exist and they are just simple function evaluations. Finally, the computational complexity of all four algorithms is constant per link and linear per source. The main difference, then, is the number of tunable parameters of each algorithm, which varies from one to three. Optimization decomposition leads us to three constructs that are generally applicable: effective capacity, consistency price and direct path-rate update.

4.3.1 Effective Capacity

The first three algorithms (partial-dual, primal-dual, and full-dual) prevent link loads from reaching link capacity by providing feedback based on *effective capacity* rather than actual capacity. In the resulting algorithms, the sources update the path rates based on feedback price just as in Figure 4.2. The feedback price is similar to that in Figure 4.2, except it is based on effective capacity y_l:

$$s_l(t+1) = s_l(t) - \beta_s \left(y_l(t) - \sum_i \sum_j H^i_{lj} z^i_j(t) \right). \tag{4.5}$$

As in Section 4.2.1, we consider constant stepsize for practical reasons, thus we remove the dependence on t from all stepsizes.

Local Optimization: Partial-Dual

The derivation process for the **partial-dual** algorithm is identical to Section 4.2.1 except with *effective capacity* $\mathbf{y}$ as an additional primal variable. The constraint $\mathbf{y} \preceq \mathbf{c}$ is enforced, resulting in the following equation for updating effective capacity:

$$y_l(t+1) = \text{minimize}_{(y_l \le c_l)} w f(y_l/c_l) - s_l(t) y_l. \tag{4.6}$$

In (4.6), y_l is updated by solving a local optimization using information from the feedback price and the cost function f. An economic interpretation is that the effective capacity balances the cost of using a link (represented by f) and revenue from traffic transmission (represented by the product of feedback price with the effective capacity). There is an explicit solution to (4.6). Note that the effect of the cost function is proportional to w.

Subgradient Update: Primal-Dual

The **primal-dual** decomposition first decomposes (4.4) into two subproblems, one responsible for each primal variable. The master problem solves for $\mathbf{y}$ assuming a given $\mathbf{x}^*$, while the subproblem solves for $\mathbf{x}$ assuming a fixed $\mathbf{y}$. The master problem is as follows:

$$\begin{aligned} &\text{maximize} \quad \textstyle\sum_i U_i(\mathbf{x}^*) - w \sum_l f(y_l/c_l) \\ &\text{subject to} \quad \mathbf{y} \preceq \mathbf{c}, \end{aligned} \tag{4.7}$$

where $\mathbf{x}^*$ is a solution to the following subproblem:

$$\begin{aligned} &\text{maximize} \quad \textstyle\sum_i U_i(x_i) \\ &\text{subject to} \quad \mathbf{Rx} \preceq \mathbf{y}. \end{aligned} \tag{4.8}$$

Note that (4.8) is identical to (4.2) except the constraint is on $\mathbf{y}$ rather than $\mathbf{c}$. The solution to the subproblem is then identical to that presented in Figure 4.2 except for the feedback price update which uses the effective capacity $\mathbf{y}$ rather than actual capacity $\mathbf{c}$.

The master problem can be solved through an iterative update of effective capacity :

$$y_l(t+k) = \min(c_l, y_l(t) + \beta_y(s_l(t) - wf'(y_l(t)))), \tag{4.9}$$

where β_y is the effective capacity stepsize. Taking a closer look at (4.9), the minimization ensures effective capacity stays below the actual capacity. The parameter k is an integer greater than 1 since (4.7) is updated less frequently than (4.8). The subgradient update itself consists of balancing the price the link can charge (s_l), and the cost that link must pay ($f_l'(y_l)$). In a nutshell, the primal-dual decomposition is identical to the partial-dual decomposition except that the effective capacity is updated iteratively through (4.9) rather than by solving a local minimization problem.

4.3.2 Consistency Price: Full Dual

The **full-dual** decomposition is quite similar to the partial-dual decomposition in Section 4.3.1, but a second dual variable $\mathbf{p}$ is introduced to relax the constraint $\mathbf{y} \preceq \mathbf{c}$. This dual variable can be interpreted as *consistency price* as it ensures *consistency* between the effective capacity and the capacity constraint at the equilibrium point. As with the feedback price, the consistency price is updated over time using a subgradient method:

$$p_l(t+1) = [p_l(t) - \beta_p(c_l - y_l(t))]^+,$$

where β_p is the stepsize for consistency price. Consistency price only comes into play when the capacity constraint is violated, therefore, it is mapped to a non-negative value. The effective capacity update is based on both link prices:

$$y_l(t+1) = \text{minimize}_{y_l} w f(y_l/c_l) - (s_l(t) + p_l(t))y_l.$$

The path rate update and feedback price update are identical to that of the previous two algorithms. The full-dual algorithm closely resembles an algorithm presented in [35], though our objective contains w as a weighing factor. Appendix 2 of [35] also shows a complete derivation of the full-dual algorithm.

4.3.3 Direct Path Rate Update: Primal

In all the previous algorithms, auxiliary dual variables were introduced to relax the constraints. In this **primal** decomposition, we find a direct solution by introducing a penalty function, as in the appendix of [49]. Let the penalty function $g_l(\sum_i \sum_j H^i_{lj} z^i_j)$ replace the capacity constraint $\mathbf{Hz} \preceq \mathbf{c}$. The penalty function is a continuous, increasing, differentiable and convex function that is sufficiently steep such that link loads will not overshoot capacity. If it is also sufficiently close to zero for values less than capacity, it will not affect the optimal point [16]. If we add g and the cost function f to get a penalty-cost function $P_l(\sum_i \sum_j H^i_{lj} z^i_j)$, then (4.4) can be transformed into the following:

$$\text{maximize} \sum_i U_i(\sum_j z^i_j) - w \sum_l P_l(\sum_i \sum_j H^i_{lj} z^i_j). \tag{4.10}$$

The derivative of (4.10) is:

$$\frac{dz_i}{dt} = \beta_z \frac{\partial U_i}{\partial z_j^i}(x_i(t)) - w \sum_l P_l'(\sum_i \sum_j H_{lj}^i z_j^i(t)), \tag{4.11}$$

where β_z is the stepsize for path rate. Converting (4.11) into a subgradient update form and separating link information from source information, we obtain the algorithm in Figure 4.3.

Path rate update:

$$z_j^i(t+1) = z_j^i(t) + \beta_z z_j^i(t)(\frac{\partial U_i}{\partial z_j^i}(x_i(t)) - \sum_l H_{lj}^i s_l(t))$$

Feedback price update:

$$s_l(t+1) = wP_l'(\sum_i \sum_j H_{lj}^i z_j^i(t))$$

Figure 4.3: The Primal algorithm.

The path rates are iteratively updated based on the difference between the rate of change of the utility function and the associated path feedback price. The feedback price here directly represents how quickly the penalty function is changing at a given link load. The primal algorithm in Figure 4.3 differs significantly from the first three decompositions. First, it uses direct subgradient update on the path rates. Second, it does not use the concept of effective capacity.

4.4 Convergence Properties

In this section, we study convergence properties of the four algorithms, and make key observations which will guide our design of a new protocol in Section 4.5. First, we find that there is a trade-off between the speed of convergence and the achievable aggregate utility. Second, we find algorithms which use local minimizations instead of iterative updates converge faster. Third, we find consistency price can aid convergence for small w.

4.4.1 Set-up of MATLAB Experiments

Due to the multitude of tuning parameters, finding the optimal values requires fine-grained sweeping of the parameter space. Thus we use MATLAB simulations along with simple topologies and simple traffic patterns to identify the key properties that improve convergence. For all algorithms, we update the source and link variables at each iteration based on link load from the previous iteration. For the utility function U, we use a logarithmic function commonly associated with proportional fairness and TCP Reno today [65]. For the cost-function f, we use an exponential function, which is the continuous version of the function used in various studies of traffic engineering [30].

We study three realistic topologies as shown in Figures 3.3 (introduced in previous chapter) and 4.4. Figure 3.3a is a tree-mesh topology, which is representative of a common access-core network structure. Figure 3.3b is the Abilene backbone network [6]. Finally, Figure 4.4 represents a multihoming topology where many multihomed stubs are all trying to reach the same destination through three ISPs. We select six source-destination pairs for access core and four pairs for Abilene. For each of these communicating pairs, three minimum-hop paths are available for access-core and four minimum-hop paths are available for Abilene. The simulations assume the link capacities follow a truncated (to avoid negative values) Gaussian distribution, with an average of 100 and a standard deviation of 10. For this set of experiments, we define convergence as reaching 99.9% of the optimal aggregate utility of (4.3). We found the convergence rates to be independent of initial rate assignments. We omit extra graphs when the same trends are observed across algorithms, topologies and

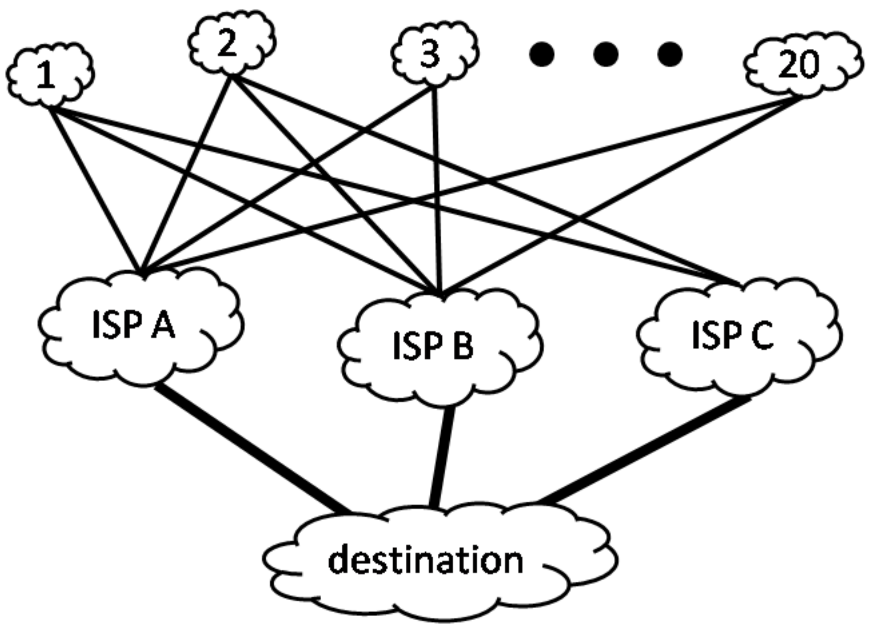
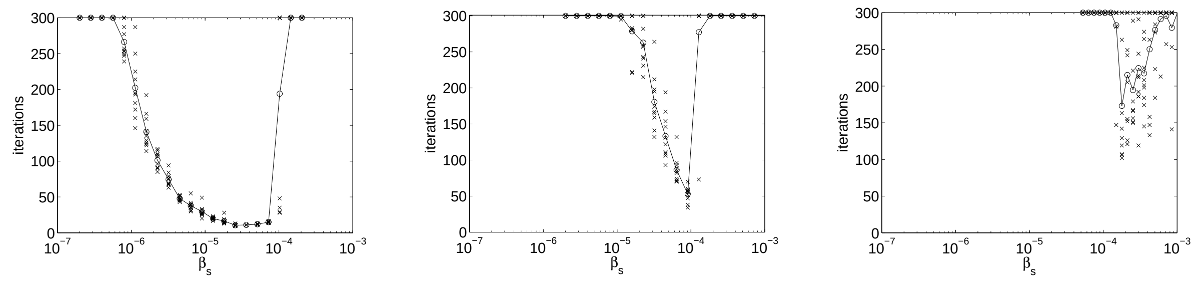


Figure 4.4: Topology capturing 20 stub networks connected to 3 ISPs. All networks are connected to ISP A, networks 1 through 10 are connected to ISP B and networks 1, 2, 3, 11, 12, 13 are connected to ISP C. The links connecting the ISPs to the destination have feedback delay from $30ms$ to $80ms$.

values of w, more detailed results can be found in [34].

4.4.2 Weighing User Utility and Operator Cost



(a) $w = 1$ (b) $w = 1/6$ (c) $w = 1/36$

Figure 4.5: Plots of partial-dual algorithm showing dependence of convergence time on stepsize. 'x' represent the actual data points and 'o' represent the average value. Access-core topology was used.

In this subsection, we study the *trade-off between aggregate utility and convergence* *time*. In Figure 4.5, we plot the number of iterations before convergence against stepsize β_s for three values of w for the partial-dual algorithm. For each stepsize, each point corresponds to a set of capacity values, and the average number of iterations before convergence is highlighted in a solid line. Comparing across Figure 4.5 from left to right, we see that as w decreases, the convergence time at the optimal stepsize increases and the range of stepsizes with a good convergence time shrinks.

In Figure 4.6, we plot the aggregate utility achieved by solving (4.3) as a percentage of maximal aggregate utility achieved by solving (4.1), for a range of w values. From the graph, we observe that there is a knee region for all three topologies. For the Abilene topology, this knee region is $w = [1/6, 1/10]$; for the access-core topology, this knee region is $w = [1/4, 1/6]$; for the multihoming topology, the knee region is $w = [5/4, 3/2]$. For w values smaller than the knee region, the algorithm achieves near maximal aggregate utility, since the cost function f is weighed sufficiently lightly to not change achieved aggregate utility. For w values larger than the knee region, the aggregate utility achieved decreases, as the cost function f becomes a significant part of the objective. The location of the knee region depends on whether bottleneck links are shared by many source-destination pairs, which is dependent on the topology, associated capacities, and the source-destination pairs chosen.

In this chapter, we define a *flow* to be the aggregate connections between a source-destination pair. In the multihoming topology, the three links connecting the ISPs to the destination are shared by many flows. If all links have equal capacities, then those three links are bottleneck links. Since the f function is a sum over all links, when a bottleneck link is shared by many flows, the penalty associated with pushing that link to full capacity is compensated by driving all sending rates higher. Consequently, for a given w value, the gap to maximal achievable utility is smaller when there is a single bottleneck link shared by multiple flows, than when there are many bottleneck links. So, in Figure 4.6, we observe the aggregate utility is at 100% even at $w = 1$ for the multihoming topology. Similarly, the access-core topology has a knee at a larger w than the Abilene topology, since several flows share the mesh at the center in the access-core topology.

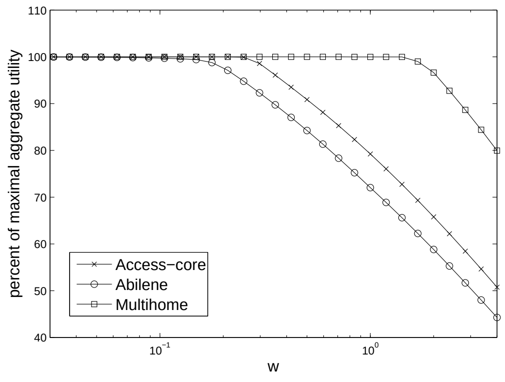


Figure 4.6: Plot of w versus percentage of maximal utility achieved.

Looking at Figures 4.5 and 4.6 together, at $w = 1/36$, the maximal aggregate utility is achieved, but the partial-dual algorithm converges slowly and is very sensitive to stepsize choice. In comparison, for a mere 3% reduction in aggregate utility, the convergence properties improve significantly at $w = 1/6$. As w increases even more, there is a clear trade-off between aggregate utility achieved, the rate of convergence and sensitivity to stepsize. At $w = 1$, the convergence properties are much nicer than at $w = 1/6$, but there is a 20% drop in utility. For a given w value, if the topology, capacities and traffic pattern causes multiple flows to share a single bottleneck link (*e.g.*, mulithoming topology instead of access-core topology), then there is a higher utility achieved, but a slower rate of convergence.

4.4.3 Comparing the Algorithms

In this subsection, we do a series of comparisons between convergence time and stepsize sensitivity of the four algorithms, and find partial-dual in Figure 4.5 is the best overall, with a good convergence profile and fewest tunable parameters. We summarize our observations in Table 4.2.

Algorithm	Partial-Dual	Primal-Dual	Full-Dual	Primal
$w = 1$, Access-Core	15	25*	15	25
$w = 1/6$, Access-Core	50*	75**	125*	150*
$w = 1$, Abilene	15	25*	15	25
$w = 1/6$, Abilene	125*	100**	50*	150*

Table 4.2: Summary of average number of iterations to convergence for best chosen tuning parameters. Here * denotes sensitivity to stepsize variation and ** denotes extra sensitivity to stepsize variation.

Comparing the primal-dual algorithm to the partial-dual algorithm, we find *the two extra tunable parameters do not improve the convergence properties.* The convergence times of primal-dual and partial-dual algorithms are almost identical for well-chosen β_y and k. For other values of β_y, however, we find the primal-dual algorithm converges more slowly than the partial-dual algorithm.

Comparing the full-dual algorithm in Section 4.3.2 to the partial-dual algorithm, we find *consistency price may improve convergence properties.* From Table 4.2, we note that β_p has no effect on the convergence time when $w = 1$. This is because the effective capacity stays far below actual capacity when w is high, so consistency price p_l stays at 0 and its stepsize plays no role. For $w = 1/6$ (which is the edge of the knee region seen in Figure 4.6), we find that the full-dual algorithm can converge faster than the partial-dual algorithm. This is because if we allow the capacity constraint to be violated during transient periods, the algorithm can take more aggressive steps and potentially converge faster.

Comparing the primal algorithm in Section 4.3.3 to the partial-dual algorithm, we find *local minimization update has better convergence properties than subgradient update.* This is intuitive as the subgradient update with a constant stepsize is constrained to react with the same strength each time, while local minimization can react more flexibly. From Table 4.2, the primal algorithm takes longer to converge at the optimal stepsize (25 iterations versus 15 iterations). In addition, the primal algorithm also requires operators to tune a second parameter g.

4.5 TRUMP

While the algorithms introduced in Section 4.3 converge faster than DUMP, we seek an algorithm with even better convergence properties. In this section, we introduce Traffic-management Using Multipath Protocol (TRUMP) with only one easy to tune parameter.

4.5.1 The TRUMP Algorithm

Our simulations in the previous section suggest that simpler algorithms with fewer tunable parameters converge faster, although having a second link price can help for small w. Using those observations, we *combine the best parts of all four algorithms* to construct the TRUMP algorithm described in Figure 4.7.

In TRUMP, the feedback price has two components as in the *full-dual* algorithm: p_l and q_l. Since we observed that local optimization worked better than subgradient update, we use the feedback price update from *primal* algorithm in Figure 4.3 as our q_l. This has the additional benefit of removing one tuning parameter from the protocol since the update of q_l involves no stepsize. By a similar argument, we use a local optimization for the path rate update as in the dual-based algorithms. The value of w is only known at the sources where the z's are computed, and there is only a single value for the network. The packet-level simulations in Section 4.6.3 reveal that TRUMP performs well for a large range of w-values when an appropriate stepsize is chosen.

Through simulations, we find that TRUMP indeed converges to the optimum of (4.3) for both topologies and a range of w values. When we plot the achieved aggregate utility at equilibrium versus w, we obtain a plot identical to Figure 4.6. In

Feedback price update:

$$s_l(t+1) = p_l(t+1) + q_l(t+1),$$

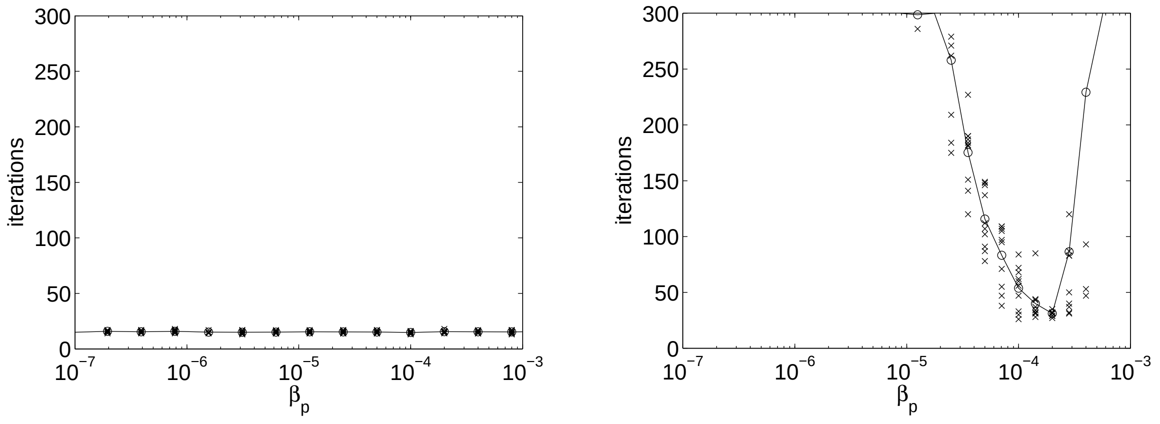
$$p_l(t+1) = [p_l(t) - \beta_p(c_l - \sum_i \sum_j H^i_{lj} z^i_j(t))]^+,$$

$$q_l(t+1) = wf'\left(\sum_i \sum_j H^i_{lj} z^i_j(t)/c_l\right),$$

Path-rate update:

$$z^i_j(t+1) = \text{maximize}_{z^i_j} U_i\left(\sum_j z^i_j\right) - \sum_l s_l(t) \sum_j H^i_{lj} z^i_j$$

Figure 4.7: The TRUMP algorithm.



(a) $w = 1$ (b) $w = 1/6$

Figure 4.8: Plots of TRUMP algorithm showing dependence of convergence time on stepsize. 'x' represent the actual data points and 'o' represent the average value. Access-core topology.

Figure 4.8, we plot convergence time versus stepsize for TRUMP. When the network sources are reacting strongly to the price q (e.g., $w = 1$ and the traffic engineering part is dominating), the price p is unnecessary as observed in Figure 4.8a. In the region where the network is being less conservative ($w = 1/6$), price p is a more definitive indicator of performance than price q, and can be helpful for source rate adjustments.

Comparing Figure 4.8 to Figure 4.5, we see that TRUMP has nicer convergence properties than the partial-dual algorithm, while having fewer parameters.

4.5.2 TRUMP Convergence Proof

Unlike the algorithms from Section 4.3, TRUMP is a heuristic and does not correspond to a known decomposition. Consequently, the convergence and optimality is not automatically guaranteed by optimization theory. Theorem 4.5.1 below guarantees convergence of TRUMP when the network is lightly loaded. We consider the region where w is sufficiently large for $p = 0$ (as seen in Figure 4.8a), and find a contraction mapping on z. Overall, TRUMP is simpler than any of the algorithms presented in Section 4.3, with only one tunable parameter that only needs to be tuned for small w.

Theorem 4.5.1. *TRUMP converges to the optimal value of (4.3) under the following conditions:*

1. $p_l = 0$, $\forall l$
2. $n_l < \alpha \frac{f_l'(u_l)^{(1/\alpha+1)}}{f_l''(u_l)}$, $\forall l$

where n_l is the number of flows sharing link l and α refers to α-fair utility [65].

Proof. If $p = 0$, then the z update is:

$$z_j^i = U_i'^{-1}(\sum_l H_{lj}^i f_l'(\sum_{i,j} z_j^i H_{lj}^i / c_l)/c_l).$$

We look for a *contraction mapping* for z as outlined in [16]. First we compute the Jacobian for z_j^i:

$$\begin{aligned} J_{ij,st} = \quad & (U_i'^{-1})'(\textstyle\sum_l H_{lj}^i f_l'(\sum_{i,j} z_j^i H_{lj}^i/c_l)/c_l) \\ & (\textstyle\sum_l H_{lj}^i \sum_{s,t} H_{lt}^s f_l''(\sum_{i,j} z_j^i H_{lj}^i/c_l)/(c_l)^2). \end{aligned}$$

Let $u_l = \sum_{i,j} z_j^i H_{lj}^i/c_l$ be the link utilization, then:

$$||J||_\infty = \max_{ij}(U_i'^{-1})'(\sum_l H_{lj}^i f_l'(u_l))(\sum_l H_{lj}^i \sum_{s,t} H_{lt}^s f_l''(u_l)).$$

Let $n_l = \sum_{s,t} H_{lt}^s$ represent the number of flows sharing link l, then:

$$||J||_\infty = \max_{ij}(U_i'^{-1})'(\sum_l H_{lj}^i f_l'(u_l))(\sum_l H_{lj}^i n_l f_l''(u_l)).$$

For convergence of z, $||J||_\infty < 1$ is a *sufficient* condition. For α-utility, we have $(U_i'^{-1})' = -\frac{1}{\alpha}x^{-1/\alpha-1}$ for $\alpha > 1$. For $U = \log(x), \alpha = 1$, $(U_i'^{-1})' = -x^{-2}$, so the same equation holds. So we can rewrite $||J||_\infty$ as:

$$||J||_\infty = \max_{ij} \frac{1}{\alpha} \frac{\sum_l H_{lj}^i n_l f_l''(u_l)}{(\sum_l H_{lj}^i f_l'(u_l))^{(1/\alpha+1)}}.$$

$||J||_\infty < 1$ holds if $\frac{n_l}{\alpha} f_l''(u_l) < f_l'(u_l)^{(1/\alpha+1)}$, $\forall l$. $\square$

4.5.3 TRUMP: Transition to Network Protocol

The transition from a mathematical algorithm to a network protocol requires relaxation of several simplifying assumptions. First, the algorithm in Figure 4.7 assumes feedback is signaled explicitly from links to sources. The explicit feedback could be piggy-backed on acknowledgment packets [47], attached to probe packets [46] or flooded throughout the network [63]. In all cases, there is delay associated with the feedback. Second, the algorithm assumes traffic flows fluidly, while real traffic consists of *packets*. Third, while an algorithm can be broadly defined with a family of functions U and f, a *specific* U and f must be selected. We address these concerns in the TRUMP protocol.

The time between iterations of the TRUMP algorithm depends on RTT^i_j, the time it takes for source i to receive feedback along all the links of path j. To transition to a packet-based protocol, the link prices are calculated based on the estimated local link load: N_T the number of bits which arrived in period $(t, t+T)$ divided by length of the period. Choosing f as an exponential function, each link updates its prices as:

$$p_l(t+T) = [p_l(t) - \beta_p(c_l - \tfrac{N_T}{T})]^+, \tag{4.12}$$

$$q_l(t+T) = \tfrac{w}{c_l} * \exp\left(\tfrac{N_T}{Tc_l}\right), \tag{4.13}$$

$$s_l(t+T) = p_l(t+T) + q_l(t+T). \tag{4.14}$$

Choosing a logarithmic function for U and solving the local minimization, we obtain the following source rate update:

$$z^i_j(t+T^i_j) = z^i_j(t) - \gamma \sum_j z^i_j(t) + \frac{\gamma}{\sum_l H^i_{lj} s_l(t)}. \tag{4.15}$$

At time 0, the prices are initialized to a constant before real prices are available after one RTT. New flows after time 0 are set at the calculated path rates according to the latest (delayed) price, collected by a probe before the flow starts. To control the rate of convergence for flows with varying RTTs, as commonly done in congestion control mechanisms, *e.g.* [90], we introduce a parameter $0 < \gamma < 1$. In general, path rates are updated every $\gamma\mathrm{RTT}^i_j$, but the path rate is recalculated at most once for any given price update. Thus the path rate adaptation will happen every $T^i_j = \max(T, \gamma\mathrm{RTT}^i_j)$. Note that the extra parameters γ and T are necessary for any packet-level protocol.

4.6 TRUMP: Packet-level Evaluation

In our MATLAB simulations, we had made a number of simplifying assumptions. Moving to packet-level simulations, we study the impact of relaxing the following assumptions: homogeneous feedback delay, no flow dynamics and no packet-level burstiness. In addition, we test TRUMP under realistic traffic loads and link failures. Finally, we examine whether TRUMP shares bottleneck links fairly.

4.6.1 Experimental Set-up

We implement the TRUMP protocol in NS-2 as described in Section 4.5.3. In particular, the link prices are updated every $5ms$ and feedback is piggybacked from the links to the sources. The path rates are updated with $\gamma = 0.1$. Most of the experiments are performed with $w = 1$, where there is no packet loss. The calculated source rates are compared to the ideal rates, which are determined using MOSEK optimization software.

Our simulations use both synthetic and realistic topologies, which are summarized in Tables 4.3 and 4.4 respectively. For the topologies that were previously simulated in MATLAB (Figure 3.3), we use the same paths with link capacities of 100Mb/s. Link delays on the Abilene topology were selected to approximate the realistic values. Links in Access-Core topology have a one-way propagation delay of $50ms$, a value chosen to test TRUMP under long feedback delay. Figure 4.4 contains three heavily loaded links, and hence was chosen for tuning of β_p under varying capacities, with link delays varying from $30ms$ to $80ms$. Specific paths and link delays are selected in the Share topology (Figure 4.14a) to test the fairness of TRUMP. Links in the Share topology have a capacity of 200Mbps, except for the bottleneck link from node 7 to node 8, which has a capacity of 100Mbps.

Since TRUMP with explicit feedback is most easily deployed inside a single AS,

Topology	**Nodes**	**Links**	**Flows**	**Paths**
Abilene	11	28	4	4
Access Core	10	24	6	3
Multihome	24	40	20	1-3
Share	9	16	3	1

Table 4.3: Summary of synthetic topologies.

we obtained intra-AS topologies, along with link delays from the Rocketfuel topology mapping engine [80, 81]. The link capacities are 100Mbps if neither endpoint has degree larger than 7, and 52Mbps otherwise. As summarized in Table 4.4, between 10 and 50 flows were randomly selected. For each source-destination pair, multiple paths were computed by first selecting a third transit node, then computing the shortest path containing all the three nodes, and finally removing cycles in the path. The RTTs on the paths range from $1ms$ to $400ms$.

ISP(AS Number)	**Cities**	**Links**	**Flows**	**Paths**
Genuity(1)	42	110	50	1-4
Telstra(1221)	44	88	20	1-4
Sprint(1239)	52	168	500	1-4
Tiscali(3257)	41	174	25	1-4
Abovenet(6461)	19	68	10	1-4
AT&T(7018)	115	296	1000	1-4

Table 4.4: Summary of ISP topologies.

4.6.2 Tuning Stepsize of TRUMP

We observed in section 4.4.2 that the links connecting the ISPs to the destination are fully utilized even for the conservative choice of $w = 1$. Previous MATLAB results indicate choosing β_p is challenging when there are bottleneck links, since packet loss can easily occur. In this section, we study the impact of link capacity on β_p, using the multihoming topology in Figure 4.4 where there are three bottleneck links.

In our first set of experiments, we vary the capacity of the links *uniformly* from

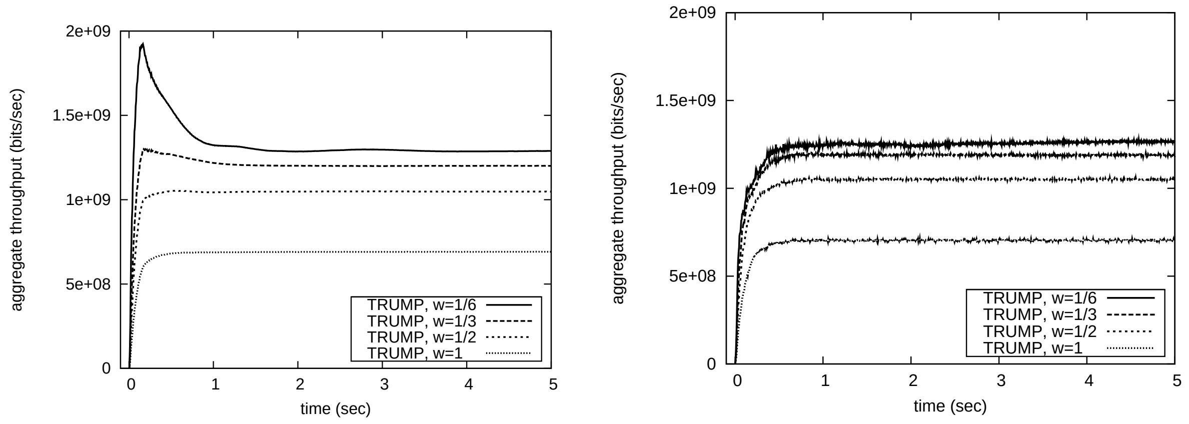
	10 Mbps	**100 Mbps**	**1000 Mbps**
$\beta_p = 1 * 10^{-11}$	no	no	no
$\beta_p = 1 * 10^{-13}$	no	no	no
$\beta_p = 1 * 10^{-15}$	yes	no	no
$\beta_p = 1 * 10^{-17}$	slow	yes	no
$\beta_p = 1 * 10^{-19}$	slow	slow	yes
$\beta_p = 1 * 10^{-21}$	slow	slow	slow
$\beta_p = 1 * 10^{-23}$	slow	slow	slow

Table 4.5: Rate of convergence of TRUMP for different β_p values and different link capacities.

10Mbps to 1000Mbps. In Table 4.5, we observe the best β_p for fast convergence is $1{*}10^{-15}$ for 10Mbps, $1{*}10^{-17}$ for 100Mbps and $1{*}10^{-19}$ for 1000Mbps. More precisely, the appropriate β_p value decreases by two orders of magnitude when the link capacities increase by one order of magnitude. Taking a closer look at (4.12), we observe that for $\beta_p = 0.1$, p_l is larger than q_l by c_l^2. Therefore, we let β_p equal $0.05/c_l^2$ and repeat the experiments with different capacities. We find that convergence is achieved with this setting of β_p. To confirm our choice holds in networks with heterogeneous capacities, we repeated the experiment with topology from Figure 4.4 with randomly assigned capacities ranging from 10Mbps to 1000Mbps. We confirmed that $\beta_p = 0.05/c_l^2$ results in a smooth convergence even in this challenging scenario.

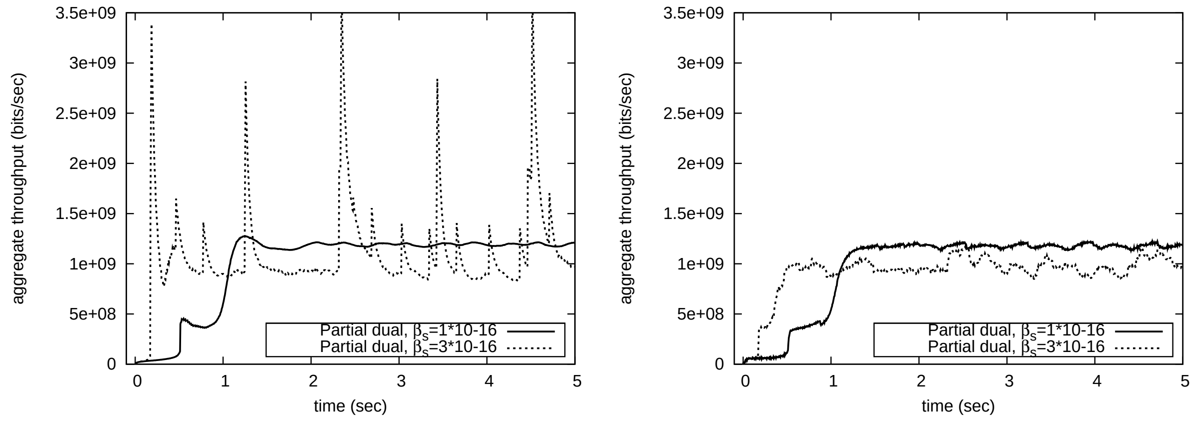
4.6.3 TRUMP versus Partial-Dual

We confirm our MATLAB results from Section 4.5.1: TRUMP has better convergence properties than partial-dual under heterogeneous feedback delay and for a range of w values. In Figures 4.9 and 4.10, we plot the aggregate throughput in the Sprint network with 50 greedy flows. The paths chosen had RTTs ranging from $3ms$ to $327ms$, with an average of $127ms$ and a standard deviation of $76ms$. Similar to the MATLAB experiments, we observe TRUMP converges slower for smaller w, though to higher aggregate rates, as shown in Figure 4.9. When $w = 1$, the TRUMP aggregate rates



(a) Sending rates at source (b) Measured rates at destination

Figure 4.9: Aggregate throughput of TRUMP with $\beta_p = 0.05/c_l^2$, in the Sprint network with 50 greedy flows.



(a) Sending rates at source (b) Measured rates at destination

Figure 4.10: Aggregate throughput of partial-dual with $w = 1/3$, in the Sprint network with 50 greedy flows.

increase from 0 at time $0s$ (when the flows are established), to close to the target value within $500ms$ — about 4 times the average RTT. When $w = 1/6$, the TRUMP aggregate rates take longer to converge, though they still converge smoothly. Comparing Figure 4.9a with Figure 4.9b, for the first second or so, the actual throughput is lower than the sending rates for small w. This is because if TRUMP's sending rates are above the bandwidth in the network, packets are lost. TRUMP converges for a range of w values with a single β_p value, chosen in the previous subsection.

Similar to the MATLAB experiments, we observe in Figure 4.10 the partial-dual is quite sensitive to the choice of stepsize at $w = 1/3$. In Figure 4.10a, we observe for a stepsize of 10^{-16}, the partial-dual sending rates converge slowly; but for a stepsize of 3×10^{-16} (only three times larger), we find the partial-dual sending rates oscillate significantly. In Figure 4.10b, we observe when the sending rates oscillate, there are heavy packet losses. In fact, the actual throughput for a stepsize of 3×10^{-16} is lower than when the stepsize is 10^{-16}. In addition, the same stepsize does not work across different values of w. By comparing Figures 4.9 and 4.10, we confirm our MATLAB results from Section 4.5.1: TRUMP has better convergence properties than partial-dual under heterogeneous feedback delay and for a range of w values.

4.6.4 Topology and Traffic Dynamics

First we consider the impact of a link failure in the Sprint Network. Path failures and recoveries are detected through active probing. All 50 greedy flows are established at 0 sec. At 5 sec the link between Pennsauken, NJ and Roachdale, IN fails, and it recovers at 10 sec. Flows 20 and 39 contain the affected link in at least one of their paths. In Figure 4.11, we plot the path rates of the flow 20. We observe that immediately after the failure, traffic is assigned to an alternate path unaffected by the failure. After the link is repaired at time 10 sec, traffic returns to the original path quickly. Similar behavior is observed for flow 39.

Second, we study the performance of TRUMP under realistic traffic loads by using 10 stochastic ON-OFF flows in the Abovenet network. As suggested by [27], the OFF periods are Pareto with shape 2.0 and average of $0.2s$. We consider three file size distributions: exponential, Pareto with shape 1.2 and Pareto with shape 1.8. In Figure 4.12, we plot the average file size against the *efficiency*: fraction of the actual throughput over the ideal throughput for a $10s$ period. The ideal throughput is found by solving (4.4). First, TRUMP's behavior is *independent* of the variance

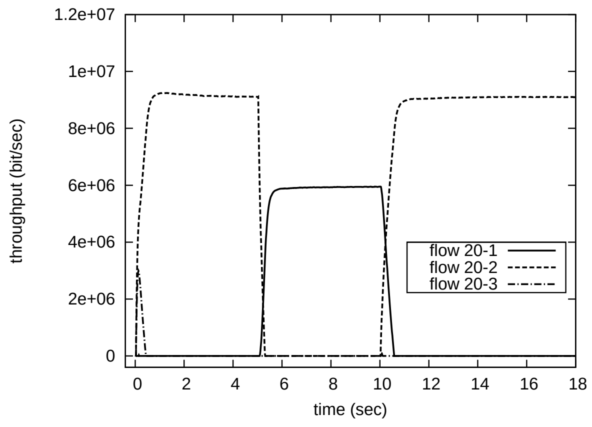


Figure 4.11: Plot of affected path rates for a link failure in the Sprint network.

of the file-size distribution, since all three curves overlap. Second, TRUMP is more efficient for larger files as it takes a few RTTs to converge to the ideal throughput. On the surface, TRUMP performs poorly for small files, only achieving 50% of the ideal rate. However, given those files are transmitted within a single RTT, achieving 50% of the ideal rate is much better than TCP today. In addition, TRUMP is optimized for logarithmic utility, for example $\log(20,000)/\log(40,000) = 0.93$. This means TRUMP achieves close to ideal utility even for short-lived flows.

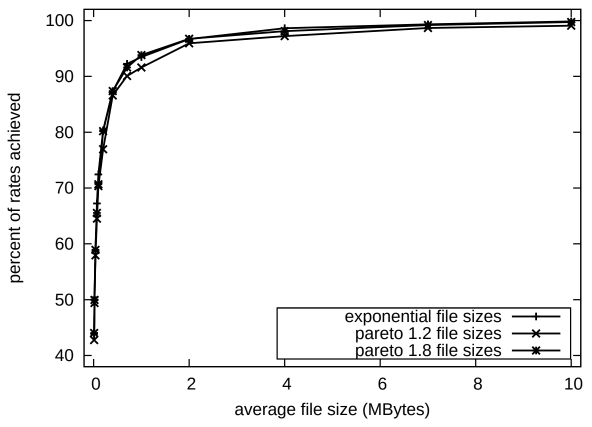


Figure 4.12: Plot of average file size versus efficiency for three distributions.

4.6.5 Selecting the Multiple Paths

There are often many paths available between each source-destination pair. In this subsection, we study how many paths to provide TRUMP, and how to select such paths.

Flows	**Single path**	**Two paths**	**Three paths**	**Shortest path**
5	56.6%	36.8%	6.6%	59.8%
10	66.8%	23.8%	6.7%	51%
25	67.2%	32.4%	0.4%	50.2%
50	76.8.4%	20.4%	2.8%	72.6%
100	70%	27.8%	2.2%	64.4%
250	88.6%	11%	0.4%	70.8%
500	91.8%	8%	0.1%	69%

Table 4.6: Impact of varying number of flows on the Sprint network.

We begin by studying how the number of flows (source-destination pairs) affects whether traffic splits over multiple paths, and whether shortest-hop paths are used. For the Sprint topology, we summarize in Table 4.6 the number of paths used by flows at equilibrium and percentage of flows using shortest-hop paths. The number of flows in the network impacts the likelihood of a flow being split amongst multiple paths. If there is a single flow in the network, it will use all the paths available to it. So when there are very few flows, a large percentage of flows place load on multiple paths simultaneously since there are many uncongested paths. As the number of flows increases, a larger percentage of flows will just select a single path, since most of the links are used by at least one flow already.

Further, we observe 50% to 73% of the flows send all their traffic on the shortest-hop path(s). Given the penalty function f is summed over all links, shorter-hop paths are generally preferred because if a longer-hop is taken, then more links are loaded. So if there are two equally loaded paths, the one with fewer hops is preferred. Longer-hop paths are more likely to be used when the network is under-utilized, because a flow might split traffic over two or three paths that are not used by any other flow. Longer-hop paths are also more likely to be utilized when the network is very congested, because a slightly longer-hop path that is much less congested is still attractive. Another advantage of shortest-hop paths is that they can be chosen a priori, while congestion levels depend on dynamic traffic patterns.

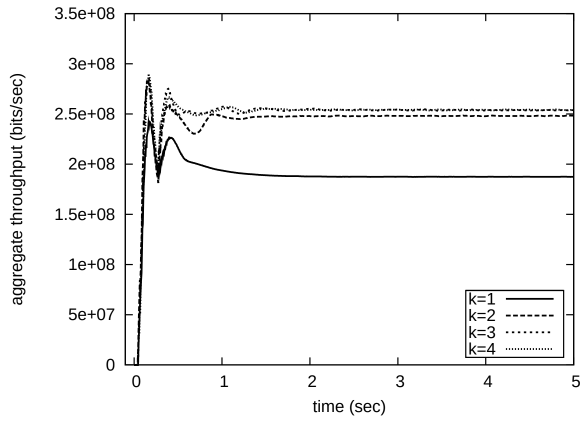
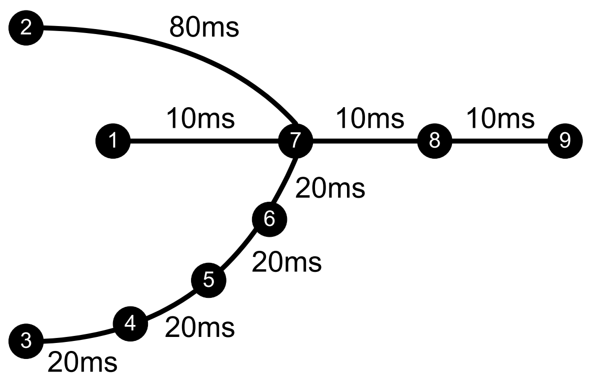


Figure 4.13: Plot of aggregate throughput versus time for the Abilene topology with four flows. Different lines correspond to different number of paths available per flow.

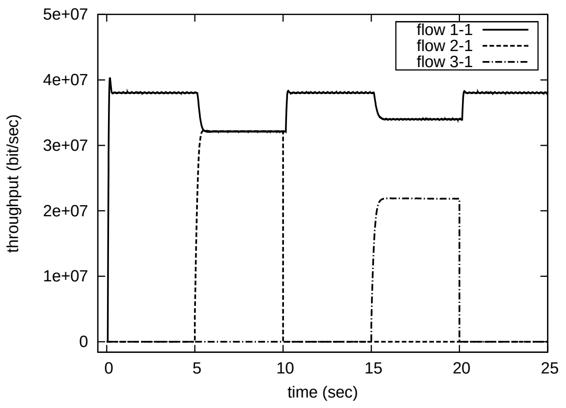
Next, we study how the number of paths available to each flow affects the utility achieved and rate of convergence. In Figure 4.13, we vary the number of paths available to each flow from one to four. For the Abilene topology with four flows, the aggregate throughput increases by 25% when there is more than one path available to a flow, though the gains are much more modest when more than two paths are provided for each source. This observation is inline with research illustrating the power of two choices [64, 51]. The number of flows has no visible impact on the rate of convergence. Looking at Figure 4.13 and Table 4.6 together, we conclude selecting two (or three) shortest-hop paths per source-destination pair is sufficient for TRUMP to perform well.

4.6.6 Fairness of Bandwidth Sharing

As mentioned in Section 4.2.2, TRUMP is α-fair as $w \to 0$, but its fairness for general w values is unknown. For $w = 1$, we construct a simple topology (Figure 4.14a)



(a) Share topology with delays



(b) Source rates

Figure 4.14: Fairness of bandwidth sharing.

to illustrate whether the bottleneck link is shared fairly. In Figure 4.14b, we plot throughput of two pairs of flows which differ in RTT or hop-count. All flows have a shared destination (node 9), and the sources are nodes 1, 2 and 3 respectively. We observe that flows 1 and 2, which have very different RTT (30ms and 100ms) but the same number of hops on their paths, share bandwidth fairly. Unlike most congestion control proposals, TRUMP does not discriminate against long RTTs since (4.3) has no dependency on RTT. While RTTs does indeed affect the transient behavior as indicated in the distributed algorithm of (13), fairness is an equilibrium property. On the other hand, flow 3 with twice as many hops receives roughly half the bandwidth of flow 1. This is inline with network operator's goals to penalize against longer-hop paths since that would require more usage of network resources. If the unshared links are lightly loaded, the bandwidth sharing would be less unfair since the amount of penalty depends on link load. It is also possible to change the source rate adaptation for TRUMP to react to path prices normalized by hop length of that path, to ensure fair bandwidth sharing for diverse hop lengths.

4.7 Related Work

Optimization theory is used in traffic management research in areas such as reverse engineering of existing protocols [48, 58], tuning configuration parameters of existing protocols [30], and guiding the design of new protocols [90] (for more references see [21]). In turn, such a broad use has encouraged innovations in optimization theory, for example, [70] introduced multiple decomposition methods. Our chapter takes advantage of the recent advancements and applies multiple decompositions to design traffic management protocols.

Most of the proposed traffic management protocols consider congestion control or traffic engineering alone. Several proposed dynamic traffic engineering protocols also load balance over multiple paths based on feedback from links [46, 24, 28], but they do not adapt the source rates. From the methodology perspective, our work bears the most resemblance to FAST TCP [90]. Other congestion control protocols that use control theory to prove stability include [47, 55, 84].

According to recent research, congestion-control and traffic-engineering practices may not interact well [8, 35, 31]. In response, many new designs are proposed. Some of them start with a different objective than this chapter, and find poor convergence properties [89, 57]. Algorithms similar to two of the decomposition solutions (Section 4.3) are described briefly in [35] and Appendix of [32], though neither considers possible design alternatives, nor present a packet-level protocol (and associated experiments).

Some research analyzes stability of joint congestion control and routing algorithms using theory [33, 69, 88], while we use optimization decomposition to guide the design of a practical protocol. Some of our evaluation is inspired by [88, 33], which prove that multipath congestion control can be stable under heterogeneous feedback delay. In particular, [33] shares a similar problem formulation and analyzes an algorithm similar to the primal-driven algorithm presented in Section 3.3, however, TRUMP offers extra flexibility through the tuning parameter w and faster convergence through an universal stepsize. In [51], they study coordinated path selection in the context of multipath congestion control, while in this chapter we find selecting two or three shortest-hop paths is sufficient for TRUMP to perform well.

4.8 Conclusions

This chapter presented a traffic-management protocol which is distributed, adaptive, robust, flexible and easy to manage. We followed a top-down design process starting with an objective which balances the goals of users and operators. We generated four provably optimal distributed solutions using known decomposition techniques. Using insight from simulations comparing the four algorithms, we combined the best parts of each algorithm to construct TRUMP. TRUMP is easy to manage, with just one optional tunable parameter. Our packet-level evaluations confirmed TRUMP is effective in reacting to topology changes and traffic shifts on a small timescale, with realistic feedback delay. We also found TRUMP's performance is only weakly dependent on the properties of file size distribution. In addition, TRUMP performs very well with access to just two or three shortest-hop paths.

This chapter started from an abstract model, and ended with a practical traffic management protocol based on feedback from the links along each path. In our ongoing work, we are exploring a version of TRUMP where the sources adapt the path rates based on observations of end-to-end delay and loss. We show that using optimization decompositions as a foundation, simulations as a building block, and engineering intuition as a guide can be a principled approach to protocol design.

Chapter 5

Supporting Multiple Traffic Classes with DaVinci

5.1 Introduction

In the previous chapter, our redesign of traffic management assumes the same traffic-management protocol for all end-host applications. In practice, different applications can have different performance objectives, for example, voice-over-IP and gaming traffic perform better over low-delay paths, whereas large file transfers perform better over high-bandwidth paths. In this context, TRUMP in the previous chapter was optimized for throughput-sensitive traffic. This chapter performs a top-down redesign of the traffic-management system to support *multiple traffic classes.*

A natural objective for an ISP is to maximize aggregate performance across multiple traffic classes. This chapter starts with an optimization formulation which captures this objective, then decomposes it into a subproblem for each traffic class, and an algorithm for dynamically adjusting the fraction of each link's bandwidth allocated to each traffic class. At the timescale of RTTs (tens or hundreds of milliseconds), each traffic class runs its own traffic-management protocol to optimize for its own objective, and allocate resources amongst its users. At a longer timescale (tens of seconds), the ISP adjusts the *bandwidth shares* (per link bandwidth resources allocated to each traffic class) based on information regarding its performance relative to its objective function. Using optimization theory, the following properties hold for the overall system:

- **Stability:** The bandwidth shares converge to a stable value, without requiring information about the bandwidth shares of other links.
- **Efficiency:** The bandwidth-share adaptation maximizes the aggregate performance of all traffic classes, if each traffic class optimizes for its own performance objective.
- **Independence:** Although bandwidth shares are changing over time, each traffic class can design and run its traffic-management protocols as if it had dedicated resources.

An interpretation of the mathematics naturally leads to a novel architecture DaVinci: Dynamically Adaptive Virtual Networks for a Customized Internet. In DaVinci, each virtual network runs its own *traffic-management protocols* to adjust how much traffic traverses each path in its virtual topology. In DaVinci, each substrate link has a *coordinator* that periodically computes the bandwidth share for each virtual network, and a *shaper* that limits the amount of traffic sent over the link based on these shares. Two technology trends today makes DaVinci feasible. Network virtualization can provide a platform for running multiple protocols in parallel, each with dedicated resources; router programmability enables customized protocols.

In general, DaVinci provides a platform for running the different algorithms presented in this dissertation. In particular, Section 5.2 presented an algorithm for adapting bandwidth shares between multiple classes, and Chapter 4 showed one example of a distributed traffic-management protocol optimized to maximize throughput. The method presented in Chapter 4 can also be used to design other distributed traffic-management protocols, such as one optimized for delay-sensitive traffic presented in Section 5.3.

The key features of DaVinci are customized protocols, separate queues, and dynamic adaptation of bandwidth shares. For a quantitative understanding of DaVinci's features, we consider a concrete example: a system with two traffic classes, optimized for *low delay* and *high throughput*, respectively. The delay-sensitive traffic prefers low propagation-delay paths and small queues. The throughput-sensitive traffic prefers high-bandwidth paths and tolerates longer queues in exchange for more efficient bandwidth usage. For a basic understanding of the system properties, we focus on a simple topology that emphasizes the interaction between delay and throughput. Using numerical examples, we illustrate that the bandwidth shares converge quickly under a wide range of settings. Using analysis, we illustrate that customized traffic-management protocols and separate queues are both important.

The rest of the chapter is organized as follows. The first few sections use optimization theory in three distinct ways. First, optimization theory is used to *prove* that the overall system converges to maximize aggregate performance (Section 5.2). Second, optimization is used to *design* both the bandwidth allocation across traffic classes (Section 5.2) and the distributed traffic-management protocols running inside each traffic class (Section 5.3). Numerical experiments on the adaptation of bandwidth shares are presented in Section 5.4. Third, optimization is used to *model* alternative architectures in Section 5.5 (customized protocols, but a single queue; a single routing protocol and separate queues). Section 5.6 provides a high-level overview of the DaVinci architecture. Section 5.7 places DaVinci in the context of other research trends: QoS, overlays, and network virtualization. Finally, we conclude the chapter and point to several directions for future work in Section 5.8.

5.2 ISP's Model of Multiple Traffic Classes

In this section, we first formalize the ISP's optimization problem. Next, we decompose this into an optimization problem for each traffic class and the bandwidth shares on each link. Then we prove stability and optimality of the distributed solution under certain conditions and discuss its benefits and limitations. We assume there are N traffic classes (TC), indexed by k, the notation is summarized in Table 5.1.

Symbol	**Meaning**
$\mathbf{s^{(k)}}$	Congestion prices in TC k. Used by TC k to compute path rates. Computed by ISP. Used by ISP to compute $\mathbf{y^{(k)}}$
$\beta_s^{(k)}$	Step size for updating $\mathbf{s^{(k)}}$.
$\mathbf{y^{(k)}}$	Bandwidth assigned to TC k. Computed by ISP.
β_y	Step size for updating $\mathbf{y^{(k)}}$.
C_l	Capacity of link l.
$U^{(k)}(\cdot)$	Performance objective function for TC k.
$w^{(k)}$	Weight the ISP assigns to $U^{(k)}(\cdot)$.
$\mathbf{z^{(k)}}$	Path rates for TC k.
$\mathbf{H^{(k)}}$	Mapping from links to paths for TC k.
p_l	Propagation delay of link l.
$x^{(1)}$	Demand of the delay-sensitive traffic.
u_l	Utilization of link l.
$f(u_l)$	Penalty function.
r_j^i	Fraction of traffic source i places on path j

Table 5.1: Summary of notation.

5.2.1 ISP's Objective

Each traffic class has a performance objective function $U^{(k)}(\cdot)$, with source rates and routing as variables controlled by that traffic class. Performance objective functions can include maximizing utility (*e.g.*, logarithmic function of throughput), and minimizing system congestion (*e.g.*, exponential function of link utilization [30]). The objective function $U^{(k)}(\cdot)$ may depend on the traffic volume exchanged between a source-destination pair i, as well as the routing that determines the fraction of traffic between each source-destination pair i that traverses each link l. In general, a source-destination pair may have *multiple paths* between each other, indexed by j.

$$H_{lj}^{(k)i} = \begin{cases} 1, & \text{if path } j \text{ of source-destination pair } i \\ & \text{in traffic class } k \text{ uses link } l \\ 0, & \text{otherwise.} \end{cases}$$

Then each traffic class k can adapt the amount of traffic source-destination pair i places on path j, captured by $z_j^{(k)i}$. We let $z_j^{(k)i}$ to take on any non-negative value, which implicitly assumes there is flexible splitting between the multiple paths. Then $\mathbf{H^{(k)}z^{(k)}}$ is the portion traffic class k contributes to the link loads. Finally, $U^{(k)}(\cdot)$ may also depend on the share of each link's capacity that the physical network allocated to class-k traffic $\mathbf{y^{(k)}}$, which could also affect the performance achieved.

The ISP's objective is to maximize social welfare: the aggregate performance of all traffic classes, subject to the capacity constraint on the physical links. This is captured by (5.1):

$$\begin{array}{ll} \text{maximize} & \sum_k w^{(k)} U^{(k)}(\mathbf{z^{(k)}}, \mathbf{y^{(k)}}) \\ \text{subject to} & \sum_k \mathbf{H^{(k)}z^{(k)}} \preceq \mathbf{C}, \\ & \mathbf{z^{(k)}} \succeq \mathbf{0}, \\ \text{variables} & \mathbf{z^{(k)}}, \mathbf{y^{(k)}} \end{array} \tag{5.1}$$

where $w^{(k)}$ is the weight the ISP assigns to represent the relative importance of traffic class k. If the ISP wants to give traffic class k strict priority, then $w^{(k)}$ can be assigned a value several orders of magnitudes larger than the other weights. Here, $\mathbf{y^{(k)}}$ is a vector capturing the share of each link's capacity that the physical network allocated to class-k traffic, which could also affect the performance achieved.

Using primal decomposition [56, 14], (5.1) can be decomposed into subproblems for each traffic class, and a bandwidth share adaptation algorithm which allocated resources between the traffic classes. The traffic-management protocols running in each network can be viewed as maximizing $U^{(k)}(\cdot)$, as captured by (5.2) below:

$$\begin{array}{ll} \text{maximize} & U^{(k)}(\mathbf{z^{(k)}}, \mathbf{y^{(k)}}) \\ \text{subject to} & \mathbf{H^{(k)}z^{(k)}} \preceq \mathbf{y^{(k)}}, \\ & \mathbf{z^{(k)}} \succeq \mathbf{0}, \\ \text{variables} & \mathbf{z^{(k)}} \end{array} \tag{5.2}$$

Given its share of the bandwidth resources $\mathbf{y^{(k)}}$, each traffic class can run distributed protocol(s) to maximize its own performance objective, as demonstrated in Chapter 4. A distributed protocol that implicitly maximizes (5.2) can be derived through a standard optimization method, *dual decomposition.* The bandwidth constraints to each traffic class can be relaxed through the introduction of a *congestion price* $s_l^{(k)}$. In economic terms, $s_l^{(k)}$ represents the price traffic class k would pay for violating the bandwidth constraint on link l, similar to interpretations of congestion control [58].

5.2.2 Adaptation of Bandwidth Shares

To determine the bandwidth share assignments $\mathbf{y_l}$ amongst traffic classes at each link l, the ISP needs to perform a two-step process described in Figure 5.1. The ISP first monitors link load and computes $\mathbf{s_l}$ as in (5.3). At each link, $s_l^{(k)}$ is updated for traffic class k based on the difference between the link load (due to that traffic class) $\sum_i \sum_j H_{lj}^{(k)i} z_j^{(k)i}$ and its share of the bandwidth $y_l^{(k)}$. The amount of the update is based on its previous value is moderated by β_s. The $[\ ]^+$ implies s_l is only positive

<u>Small timescale</u> (every T): Update $s_l^{(k)}$ for each k

$$s_l^{(k)}(t+T) = \left[s_l^{(k)}(t) - \beta_s^{(k)} \left(y_l^{(k)}(t) - \sum_i \sum_j H_{lj}^{i(k)} z_j^{i(k)}(t) \right) \right]^+, \qquad (5.3)$$

where t is time and T is at the same timescale as the longest Round Trip Time (RTT) of the network, *e.g.*, $100ms$. Let the converged value of $s_l^{(k)}$ be denoted $s_l^{*(k)}$, this is the input for updating $\mathbf{y_l}$.

<u>Large timescale</u> (every $T_y >> T$): Update $\mathbf{y_l}$

$$\begin{aligned} v_l^{(k)}(t+T_y) &= [y_l^{(k)}(t) + \beta_y(w^{(k)}(\lambda_l^{(k)}(t)) + \nabla_{\mathbf{y^{(k)}}} U^{*(k)}(\mathbf{z^{*(k)}}, \mathbf{y^{(k)}})]^+ \\ y_l^{(k)}(t+T_y) &= \text{argmin}_{y_l^{(k)}} \|y_l^{(k)} - v_l^{(k)}(t+T)\|, \\ &\quad \text{subject to} \sum_k y_l^{(k)} \le C_l \end{aligned} \qquad (5.4)$$

where T_y is the time period between bandwidth assignments, usually several orders of magnitude larger than T, *e.g.*, 10s.

Figure 5.1: The bandwidth share computation at each link l is a two step process: where $\mathbf{s_l}$ is updated at a smaller timescale than $\mathbf{y_l}$.

when the link load exceeds or achieves $y_l^{(1)}$, *i.e.* when the traffic class is fully utilizing its assigned bandwidth.

Then the bandwidth share allocated to traffic class k is updated with (5.4), in two steps. First, we introduce $v_l^{(k)}$ as an intermediate variable to facilitate the computation of $y_l^{(k)}$. The increase in $v_l^{(k)}$ is proportional to how much the ISP weighs the importance of traffic class k (denoted by $w^{(k)}$), and how much traffic class k needs extra bandwidth (indicated by $(s_l^{(k)*}(t) + \nabla_{\mathbf{y^{(k)}}} U^{*(k)}(\mathbf{z^{*(k)}}, \mathbf{y^{(k)}}))$. The parameter β_y dictates how much the new bandwidth allocation is based on the dual variables computed in the last time period. The $[\ ]^+$ ensures that all traffic classes are allocated a nonnegative amount of bandwidth. Then to enforce the capacity constraint $\sum_k y_l^{(k)} \le C_l$, where $\mathbf{C}$ is a vector of link capacities, the vector $\hat{\mathbf{y}}_\mathbf{l}$ needs to be projected onto the feasible region. The standard projection is to minimize the Euclidean distance between the feasible region and the original point $v_l^{(k)}$. From Figure 5.1, the bandwidth shares at each link are computed with only local link loads and the computation is simple.

Depending on the traffic class, the computation of the bandwidth shares could be simplified. In particular, whenever the objective function of a traffic class does not depend on $\mathbf{y^{(k)}}$, $\nabla_{\mathbf{y^{(k)}}} U^{*(k)}(\mathbf{z}^{*\mathbf{(k)}}, \mathbf{y^{(k)}})$ is zero, so the bandwidth share update is only dependent on the dual variables. In practice, this case corresponds commonly to the notion of utility functions for elastic traffic which only depend on the sending rates. Of course, some traffic classes may have objective functions which naturally depend on $\mathbf{y^{(k)}}$. For example, queuing delay depends on the available capacity, so an objective which minimizes delay depends on $\mathbf{y^{(k)}}$.

5.2.3 Convergence and Optimality

So far, we have derived the optimization problems solved by each traffic class and the adaptation of bandwidth shares by decomposing (5.1), now we take a closer look at the conditions for convergence.

Theorem 5.2.1. *The bandwidth allocation algorithm (5.4), together with each traffic class solving (5.2), converges to maximize (5.1) under the following conditions:*

1. *The problem (5.1) is a convex optimization.*

2. *The bandwidth allocation is updated after convergence of the primal variables* $\mathbf{z}$ *and the dual variables* $\mathbf{s}$ *in each subproblem (5.2).*

3. *The parameters* β_y *are diminishing with time.*

Proof. Starting with the master optimization problem (5.1), the first step in primal decomposition separates the capacity constraints into N parts. More specifically, the capacity constraint $\sum_k \mathbf{H^{(k)}} \mathbf{z^{(k)}} \preceq \mathbf{C}$ is rewritten as two sets of constraints: $\mathbf{H^{(k)}} \mathbf{z^{(k)}} \preceq \mathbf{y^{(k)}}$, $\forall k$, and $\sum_k \mathbf{y^{(k)}} \leq \mathbf{C}$. Then we can apply standard primal decomposition techniques [56, 14] to decompose (5.1) into N subproblems, each in the form of (5.2), keeping $\sum_k \mathbf{y^{(k)}} \leq \mathbf{C}$ as a constraint in the master problem. Each subproblem can be solved for both its original variables and Lagrangian multipliers (technical term for congestion prices) $s_l^{(k)}$ introduced to relax the capacity constraint per link for traffic class k. Then the master problem can update $\mathbf{y^{(k)}}$ based on a gradient update, which corresponds to (5.4). The weighing factors $w^{(k)}$ do not affect the optima of (5.2), but they do scale the Lagrangian multiplier as well as $\nabla_{\mathbf{y^{(k)}}} U^{*(k)}(\mathbf{z}^{*\mathbf{(k)}}, \mathbf{y^{(k)}})$ at the optima. Consequently, the weights $w^{(k)}$ in (5.1) are reflected in (5.4). From [56, 14], the bandwidth share allocation algorithm (5.4) converges to the maximum of (5.1) if the problem is convex and the parameters $\beta_y(t)$ are diminishing with time. $\square$

Theorem 5.2.1 has several interesting implications. As implicitly assumed in the problem formulation, each traffic class requires a separate queue, in order for the bandwidth shares to be meaningful. At equilibrium, (5.1) maximizes the aggregate performance across all traffic classes, *i.e.*, the social welfare. The adaptive bandwidth allocation algorithm is *distributed* per link and only relies on *local information.*

There are three major limitations of the theorem: convexity, selection of β_y, and timescale of adaptation. Convex objective functions apply to most performance objectives [21], so the problem formulation is still broad. Convexity of the constraint can be obtained if traffic can be split flexibly amongst multiple paths (when they exist). As shown in Chapter 2 multipath routing has become increasingly feasible over the years and certainly deployable within a single ISP. The choice of β_y will impact the speed of convergence: a smaller β_y means convergence is slower, but a larger β_y can cause divergence. We take a closer look at how to select β_y in the next section. Finally, the timescale of adapting the bandwidth shares must be chosen judiciously. If the timescale is too close to that of the distributed protocols in each traffic class, then the system could be unstable. On the other hand, if the resources are adapted very slowly, then the system might be operating inefficiently.

5.3 Two Traffic Classes

Most of the traffic in the Internet falls naturally into one of two categories: delay-sensitive and throughput-sensitive. Delay-sensitive traffic includes Voice-over-IP, video conferencing, online gaming and live video streaming. Throughput-sensitive traffic includes file transfers, Web browsing, e-mail and peer-to-peer file sharing.

Since we already studied the throughput-sensitive traffic in Chapter 4, we present only the distributed protocol for the delay-sensitive traffic here. Let the delay-sensitive traffic be allocated a bandwidth share of $\mathbf{y}^{(\mathbf{1})}$, we drop the superscripts for the other variables and constants since they are local to the delay-sensitive traffic class.

In the delay-sensitive traffic class, we assume the traffic is inelastic, *i.e.*, it does not change its sending rate based on network conditions. The delay-sensitive traffic wants to minimize the end-to-end delay it experiences. The delay along a path is the sum of the delays experienced in traversing each link. Delay on a link is the sum of propagation delay and queueing delay at that link. The delay-sensitive traffic would like to choose low propagation-delay paths, and keep the queues small to limit queuing delay.

Propagation delay p_l on a link is independent of the traffic load on it and depends primarily on the length of the link and its physical properties. To keep the queues small, we heavily penalize large queues in the objective function at link l with a convex function $f(u_l)$, where $u_l = (\mathbf{Hz})_l / y_l^{(1)}$ represents the link utilization. An example function for f could be the exponential function, which also approximates M/M/1 queueing delay. Since the traffic is inelastic, we let the source rates x^i be fixed and each source-destination pair i can decide on the percentage of traffic to place on each path based on the path delay. So the overall problem solved by the delay-sensitive traffic class is:

$$\begin{array}{ll} \text{minimize} & \sum_i \sum_j z_j^i H_{lj}^i (p_l + f((\mathbf{Hz})_l / y_l^{(1)})) \\ \text{subject to} & \mathbf{Hz} \preceq \mathbf{y^{(1)}} \\ & \mathbf{1^T z} = \mathbf{x^{(1)}} \\ & \mathbf{z} \succeq \mathbf{0} \\ \text{variables} & \mathbf{z} \end{array} \tag{5.5}$$

where $\sum_j z_j^i = x^i$ enforces that all the delay-sensitive traffic will be delivered. In reality, there may not be sufficient capacity to satisfy the demand, though in general, the delay-sensitive traffic should only be a small portion of the overall traffic in the Internet. Still, it is possible to use admission control to only accept delay-sensitive traffic if there is sufficient bandwidth to support it. Another approach would be to relax the $\sum_j z_j^i = x^i$ constraint as a penalty function as shown in [14], which would tolerate deviations from meeting the demand precisely.

Similar to [71], the distributed solution to (5.5) can be found using standard optimization techniques. As shown in Section 5.6, congestion price $\mathbf{s}$ is updated by the ISP and sent to each traffic class. The path rates are then updated via a local minimization:

$$\mathbf{z^i}(t+1) = \text{minimize}_{z_j^i} \sum_j z_j^i \left(\sum_l H_{lj}^i (p_l + f(u_l) + s_l(t)) \right) \tag{5.6}$$

where $u_l = (Hz)_l / y_l^{(1)}$. The local minimization is subject to the constraint $\sum_j z_j^i = x^i$. For a continuous, differentiable and convex f function, the local minimization has an analytical solution and is just a function evaluation. Each source minimizes the cost of using path j, which is dependent on the end-to-end propagation delay, plus the aggregate penalty of loading that path and the path price (which reflects how much the path is being used by other sources). The path price is the product of the source rate with the congestion price per unit load for path j (computed by summing s_l over the links in the path). In fact, (5.6) is like shortest-path routing, except the weights are dynamic: the propagation delay is constant, but the penalty function and price functions are load-sensitive. The distributed solution is provably stable and maximizes the performance objective of interest if the stepsize associated with the congestion price β_s is diminishing with time [16]. As with the work in the earlier chapters, the delay-sensitive traffic protocol can converge with well-tuned constant β_s, the experiments are summarized in [44].

5.4 Convergence Properties

The convergence of the overall system depends on the convergence of the individual traffic classes (on a small timescale) and the bandwidth shares (on a medium timescale). Previous work has shown how to tune protocols like the ones in Section 5.3 to converge within a few tens of iterations [40]. With the timescale separation between the convergence of $\mathbf{s}$ and the adaptation of bandwidth shares $\mathbf{y}$, we study the stability of $\mathbf{y}$ using the converged values of $\mathbf{s}$. Consequently, we do not simulate the distributed protocols for each traffic class, and instead use the values of $\mathbf{s}$ computed by solving the optimization problems directly. We use numerical experiments on a simple topology to study the convergence rate of $\mathbf{y}$ and associated sensitivity to tuning parameters.

5.4.1 Experimental Set-up

Our experiments evaluate the adaptation of the bandwidth shares on a medium timescale, based on the optimal values of $\mathbf{s}$. Since the bandwidth shares $\mathbf{y}$ are computed based only on local information, there are no feedback delays that could influence their adaptation. As such, we numerically simulate each iteration of computing

$\mathbf{y}$ using MATLAB. For the throughput-sensitive traffic, we use the same set-up as in Chapter 4, with the penalty weight set to 1. For the cost-function f, we use an exponential function, which is the continuous version of the penalty function used in various studies of traffic engineering [30].

To gain initial insight into the stability of the bandwidth shares, we study the two-link, two-node topology in Figure 5.2. The links have disparate delays and capacities, so that the delay-sensitive traffic clearly prefers the top link. The set-up is purposely simple so that it is easy to understand the intended steady-state behavior. In addition, this topology is useful for understanding the importance of customized protocols and separate queues in the next section. In our ongoing work, we leverage the same experimental framework to study larger and more realistic topologies.

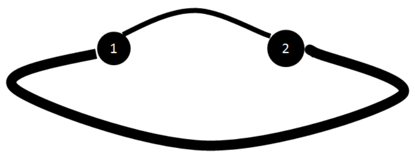


Figure 5.2: Topology with two paths between a pair of nodes. The two links have very different properties: the top link has propagation delay 5ms and bandwidth 100Mbps; the bottom link has propagation delay 50ms and bandwidth 1Gbps.

5.4.2 Sensitivity to Tunable Stepsize

The tunable stepsize β_y controls how quickly bandwidth shares $\mathbf{y}$ are reassigned between traffic classes, by reacting to changes in $\mathbf{s}$. Optimization theory allows us to prove the convergence of the bandwidth shares under the condition that β_y diminishes with each iteration of bandwidth-share computation. We relax this condition by studying the convergence rate of bandwidth shares for *constant* β_y, where convergence is defined as being within 0.001% of the optimal bandwidth shares, obtained by solving the master problem directly. In particular, we are interested in studying

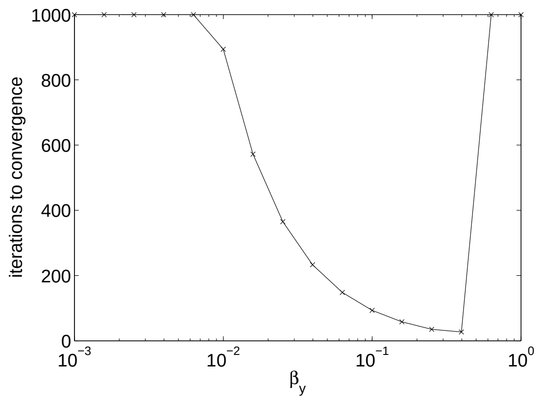


Figure 5.3: The rate of convergence versus β_y.

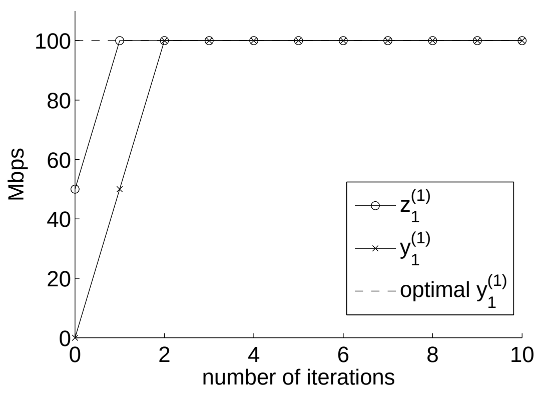
the sensitivity to β_y.

This experiment sweeps the values of β_y to observe the rate of convergence. Since s_l is independent of the actual link capacity C_l, we propose to scale β_y by C_l so that the bandwidth shares are adjusted quickly. First, we observe that the bandwidth shares do converge to their ideal values for constant β_y. Second, from Figure 5.3 we observe that convergence in under 250 iterations occurs for β_y values between 0.5 and 0.05. In particular, above a β_y value of 0.5, the bandwidth shares may not converge. The reason being as β_y gets large, there is a tendency to overshoot beyond the feasible region every single iteration.

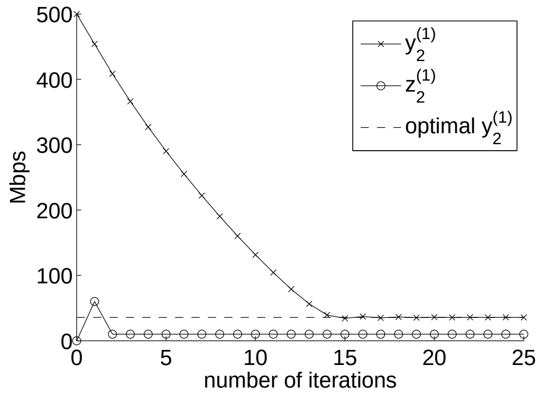
Below β_y value of 0.5, the rate of convergence slows as we move to smaller values of β_y. In practice, simulations should be run to tune the β_y value for a rate of convergence. If oscillatory behavior is observed at some point, β_y can be decreased to ensure convergence.

5.4.3 Delay-sensitive and Throughput-sensitive Traffic

In this experiment, we set the volume of the delay-sensitive traffic to be 110 Mbps, so that it cannot be accommodated solely by the low-delay link. The ideal bandwidth share assigned to the delay-sensitive traffic is all of the low-delay link plus 32Mbps of



(a) Low-delay link



(b) High-delay link

Figure 5.4: Assigned bandwidth and actual rates for delay-sensitive traffic versus iteration number. The dotted lines are the optimal bandwidth allocations. β_y is set to 0.5.

the high-delay link. Although the delay-sensitive traffic will not use all of its assigned bandwidth, this allows it to keep the queues small and thus the end-to-end delays small. The extra 22Mbps of bandwidth makes very little difference to the throughput-sensitive traffic, since it already has 968Mbps of bandwidth. On the other hand, the extra penalty caused by a fully loaded link versus a 30% loaded link is significant for the delay-sensitive traffic. Since the goal is to maximize the aggregate performance of the two traffic classes, this is a reasonable trade-off for the system to make.

As seen in Figure 5.4, the delay-sensitive traffic is initially assigned 50% of the bandwidth, thus the y-axis intercepts are 500 Mbps and 50 Mbps respectively. The initial link loads are set at zero, then jumps to 50 Mbps and 60 Mbps respectively after one iteration to satisfy the demand of 110 Mbps. From Figure 5.4a, we observe that after one iteration, the delay-sensitive traffic is assigned all of the bandwidth on the low-delay link. This is due to the large difference between the delay properties of the two links. After two iterations, the link load on this link is maintained at 100 Mbps. From Figure 5.4b, we observe that the delay-sensitive traffic's bandwidth share is consistently reduced on the high-delay link until reaching the ideal value.

For the throughput-sensitive traffic, a mirroring trend occurs. The throughput-sensitive traffic is also initially assigned 500 Mbps on the high-delay link and 50 Mbps on the low-delay link. On the low-delay link, the bandwidth share for the throughput-sensitive traffic drops to 0 Mbps. On the high-delay link, the bandwidth share for the throughput-sensitive traffic is increased to consume most of the idle bandwidth. Overall, the bandwidth in the ISP is efficiently utilized by the two traffic classes, independent of the initial conditions.

5.5 Alternative Designs

In this section, we illustrate the importance of both customized protocols and separate queues through the example topology in Figure 5.2. First, we study the case of separate queues, but a single routing protocol. Second, we study the case of customized traffic-management protocols, with a single shared queue.

5.5.1 Single Routing; Separate Queues

To understand the importance of customized protocols, we model a system with a single routing protocol and separate queues for each traffic class. The resources can be shared between the two queues via strict priority or weighted-fair queuing. For weighted-fair queuing, it is unclear which class of traffic dictates the routing when there is one routing matrix. As a result, we look at the strict priority case where the delay-sensitive traffic is given strict priority over the throughput-sensitive traffic, and dictates how all traffic is routed.

We model the system: let r_j^i be the percentage of traffic source-destination pair i places on the jth path, where $\sum_j r_j^i = 1$ for flow conservation. Since there is only one routing protocol we have $z^{(1)i} = x_i^{(1)} r^i$ and $z^{(2)i} = x_i^{(2)} r^i$. Since the delay traffic is given strict priority, the capacity constraint is $\mathbf{Hrx^{(1)}} \preceq \mathbf{C}$, where C is the total capacity in the system.

$$\begin{array}{ll} \text{minimize} & \sum_l (\mathbf{Hrx^{(1)}})_l \left(p_l + f\left(\frac{(\mathbf{Hrx^{(1)}})_l}{C_l} \right) \right) \\ \text{subject to} & \mathbf{Hrx^{(1)}} \preceq \mathbf{C} \\ & \mathbf{1^T r} = \mathbf{1^T} \\ & \mathbf{r} \succeq \mathbf{0} \\ \text{variables} & \mathbf{r} \end{array} \tag{5.7}$$

The solution of (5.7) determines the routing percentages **r**, which are then considered to be a constant for throughput-sensitive traffic. The remaining bandwidth is then given to the throughput-sensitive traffic, so the capacity constraint for this traffic is $\mathbf{Hrx^{(2)}} \preceq \mathbf{C} - \mathbf{Hrx^{(1)}}$.

$$\begin{array}{ll} \text{maximize} & \sum_i U(x_i^{(2)}) \\ \text{subject to} & \mathbf{Hrx^{(2)}} \preceq \mathbf{C} - \mathbf{Hrx^{(1)}} \\ & \mathbf{x^{(2)}} \succeq \mathbf{0} \\ \text{variables} & \mathbf{x^{(2)}} \end{array} \tag{5.8}$$

Note in (5.8), the throughput-sensitive traffic can control the amount of traffic it places on the network, but not the paths taken by the traffic.

Consider the simple topology presented earlier in Figure 5.2; by varying the delay-sensitive traffic volume, we can make the following observations analytically:

- Efficiency of resource usage is highly dependent on the volume of delay-sensitive traffic.

- The throughput-sensitive traffic volume is highly dependent on the volume of delay-sensitive traffic.

Since there is only one source-destination pair in Figure 5.2, there is no need to index the sources by i. The delay-sensitive traffic volume falls into three possible scenarios (units in Mbps): $0 \leq x^{(1)} \leq 100$, $100 \leq x^{(1)} \leq 1100$ and $x^{(1)} \geq 1100$. Considering each case separately:

- *Lightly loaded* ($0 \leq x^{(1)} \leq 100$): The delay-sensitive traffic routes all its traffic on the shorter path, so $r_2 = 0$. Then the throughput-sensitive traffic, being elastic, fills up the rest of the shorter path, *i.e.*, $x^{(2)} = 100 - x^{(1)}$. In this case, the network is highly under utilized, since the longer path, with capacity of 1000 Mbps is left entirely idle.

- *Critically loaded* ($100 \leq x^{(1)} \leq 1100$): The delay-sensitive traffic first fills up the shorter path, then routes the remaining amount on the longer path, so $r_2 = (x^{(1)} - 100)/1000$. Since there is no space left on the shorter path to accommodate the throughput-sensitive traffic, in order to maintain the routing ratio r, throughput-sensitive traffic must be zero.

- *Overloaded* ($x^{(1)} \geq 1100$): In this case, the system lacks the bandwidth to support the delay-sensitive traffic.

Compared with (5.1) where all the resources would be consumed at equilibrium regardless of the delay traffic volume, strict priority queuing is inefficient. Our simple example serves to highlight the importance to separate routing.

5.5.2 Single Queue; Customized Protocols

To understand the importance of separate queues, we model a system with a single shared queue and customized protocols for each traffic class. The topologies can be different, captured by $\mathbf{H^{(1)}}$ and $\mathbf{H^{(2)}}$ respectively. With a single queue, the queueing delay is based on the sum of both loads over the entire capacity. We model the delay-sensitive traffic in (5.9) and the throughput-sensitive traffic in (5.10):

$$\begin{array}{ll} \text{minimize} & \sum_l (\mathbf{H^{(1)}z^{(1)}})_l (p_l + f(\sum_{k=1,2} (\mathbf{H^{(k)}z^{(k)}})_l / C_l)) \\ \text{subject to} & \mathbf{H^{(1)}z^{(1)}} + \mathbf{H^{(2)}z^{(2)}} \preceq \mathbf{C} \\ & \mathbf{1^T z^{(1)}} \succeq \mathbf{x^{(1)}} \\ & \mathbf{z^{(1)}} \succeq \mathbf{0} \\ \text{variables} & \mathbf{z^{(1)}} \end{array} \tag{5.9}$$

$$\begin{array}{ll} \text{maximize} & \sum_i U(\sum_j z_j^{i(2)}) \\ \text{subject to} & \mathbf{H^{(1)}z^{(1)}} + \mathbf{H^{(2)}z^{(2)}} \preceq \mathbf{C} \\ & \mathbf{z^{(2)}} \succeq \mathbf{0} \\ \text{variables} & \mathbf{z^{(2)}} \end{array} \tag{5.10}$$

In (5.9), there is a constraint that the delay-sensitive traffic must be satisfied. In reality, without explicit coordination between the two networks, this constraint might not be satisfiable.

Consider the simple topology presented earlier in Figure 5.2; by varying the delay-sensitive traffic volume, we can make the following observations analytically:

- The performance of the delay-sensitive traffic is highly dependent on initial conditions for the throughput-sensitive traffic.

- Even when the initial conditions favor the delay-sensitive traffic, the overall delay experienced is still larger than when the queues are separate.

For simplicity, we assume that the delay-sensitive traffic is less than 100 Mbps, since it is more common for delay-sensitive traffic to be a relatively small part of the overall traffic. This means the delay-sensitive traffic has a clear preference for the shorter path and can be satisfied by using only the shorter path. The throughput-sensitive traffic can route in three different ways: using both paths, using the shorter path only or using the longer path only. Considering each case separately, we find:

- *Using both paths*: The throughput-sensitive traffic can fill up both paths (since it is elastic), thus not leaving enough room for the delay-sensitive traffic. Even if there is sufficient space for delay-sensitive traffic across the two links, part of the delay-sensitive traffic might be forced onto the longer path.
- *Using the shorter path only*: The throughput-sensitive traffic will fill up the shorter path, leaving the longer path for the delay-sensitive traffic.
- *Using the longer path only*: In this case, the delay-sensitive traffic gets its preferred path, but since there is only one queue for both types of traffic, the queuing delay is slightly longer than when there are separate resources.

Since the throughput-sensitive traffic is elastic, it can easily overwhelm the delay-sensitive traffic without the separation of resources. In (5.1), the delay-sensitive traffic can always be supported as long as it is less than system capacity. In addition, the delay-sensitive traffic will be routed on its preferred path(s), irrespective of the initial routing configuration of the throughput-sensitive traffic. Our simple example serves to highlight the importance of resource separation.

5.6 DaVinci Architecture

This section introduces the basic building blocks of DaVinci and how they work together. The physical network, which we refer to as the substrate, can leverage existing router-virtualization technology to run several virtual nodes on the same substrate node. In addition, virtual links can be established between virtual nodes using standard tunneling techniques. Each virtual network runs its own traffic-management protocols: the adaptation of source rates and routing to efficiently utilize resources. The substrate, in turn, monitors the load on each virtual link to adapt the bandwidth shares. The virtual networks and the substrate network, collectively, maximize the aggregate performance over all virtual networks.

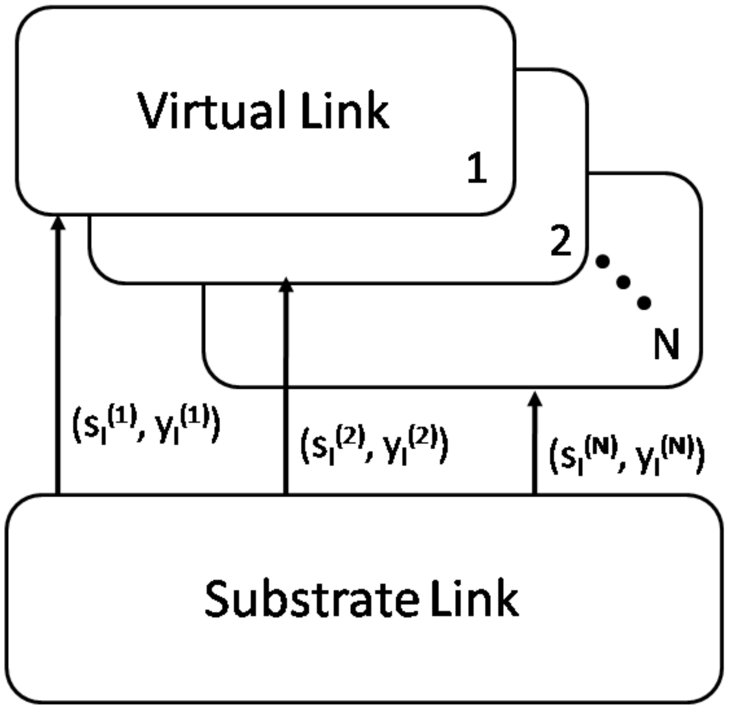


Figure 5.5: Single-link view of DaVinci.

Consider an instance of DaVinci with N virtual networks, denoted by superscript (k), where $k = 1, 2, ..., N$. Each virtual network consists of virtual nodes that each have a share of the CPU and memory of the corresponding substrate nodes for running the distributed traffic-management protocols. We assume that the virtual nodes have sufficient processing and memory resources, and instead focus on how the virtual networks share the bandwidth of the underlying substrate links. Each virtual link knows its current "capacity" $y_l^{(k)}$, as well as a measure of the current level of congestion $s_l^{(k)}$ (which we refer to as congestion prices), as shown in Figure 5.5. The virtual network uses $\mathbf{s^{(k)}}$ and $\mathbf{y^{(k)}}$ in its traffic-management protocols to make efficient use of the allocated resources.

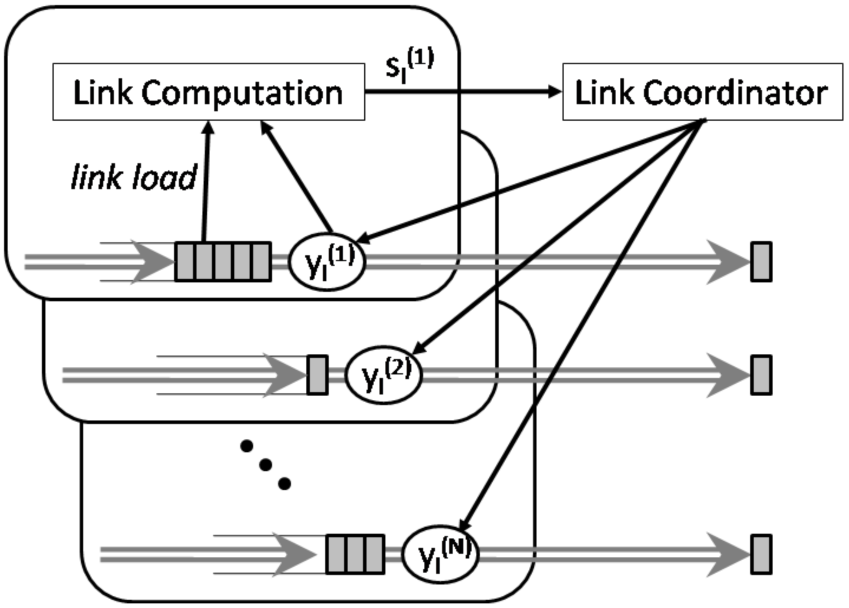


Figure 5.6: Each substrate link computes bandwidth shares for the virtual links that traverse it.

Substrate link l monitors the load on each virtual link that traverses it to compute $s_l^{(k)}$ and feeds this information to a *link coordinator*, as shown in Figure 5.6. The link coordinator periodically computes the bandwidth shares y_l, using its knowledge of the objective functions of the virtual networks. In addition, the substrate network ensures that $\sum_k y_l^{(k)} = C_l$, where C_l is the capacity of the substrate link. At a smaller timescale, the substrate has a *shaper* for each virtual link that serves incoming data packets based on the bandwidth share $y_l^{(k)}$. The shaper is *non-work-conserving*, *i.e.*, if one virtual link has idle resources, the substrate does not transmit extra packets on behalf of the remaining virtual links. Instead, the incoming packets for busy virtual links accumulate in queues awaiting service. Using a non-work-conserving shaper ensures that isolation between virtual networks is maintained between updates of bandwidth shares.

All data packets are handled by the substrate at the behest of the virtual networks.

At an edge node of the substrate, data packets are directed to the appropriate virtual network using packet classification or some other form of "user opt-in". In particular, the substrate has separate forwarding tables for each virtual network, so a packet is forwarded onto the appropriate outgoing link. At each outgoing link, there are separate queues corresponding to each virtual link.

Within the same virtual network, each edge virtual node may have multiple (virtual) paths for reaching another virtual node. The virtual nodes can compute these paths in either a central or a distributed fashion. To distinguish between multiple paths within the same virtual network, packets are encapsulated with *labels* at the edge. Virtual nodes can then populate *label tables* based on the paths they computed. In particular, the virtual node i computes a *path rate* z_j^i that determines the amount of traffic directed over path j.

The main building blocks of DaVinci—such as router virtualization, packet encapsulation, traffic shapers, and forwarding engines—are readily available today. The main novelty of DaVinci is (i) the way these components are combined and (ii) how the link coordinator adapts the bandwidth shares to ensure that the system maximizes aggregate performance.

5.7 Related Work

DaVinci is related to several trends in networking research, from the long history of research on QoS, to recent work on overlay networks and the emerging interest in network virtualization. Similar to past work on differentiated services, network virtualization has separate queues for different classes of traffic. Yet, DaVinci's goals are quite different than that of traditional QoS. Rather than offering performance guarantees (at the cost of reserving resources), DaVinci maximizes the aggregate performance across all traffic classes. Similar to overlays, DaVinci enables multiple customized protocols to run on top of a shared substrate. Yet, unlike overlays, DaVinci's virtual networks run directly on the substrate and have their own shares of the underlying resources.

5.7.1 Quality of Service (QoS)

There is a rich body of research on Quality-of-Service techniques with the goal of providing performance guarantees. At one extreme, the IntServ [18] architecture offers per-flow performance guaranees by having the routers direct traffic to separate queues (for each flow) and employ packet-scheduling algorithms that arbitrate access to the shared links. The routers may also run QoS-routing protocols [20, 52] to identify paths that can satisfy the performance requirements, as well as signalling protocols to reserve resources along these paths. In contrast, DaVinci does not perform any resource reservation or any per-flow operations. In that sense, DaVinci is more similar to DiffServ [17] and Type-of-Service (ToS) routing [61], which manage resources at the granularity of traffic classes, rather than flows.

DiffServ first classifies packets at the edge of the network based on their Type of Service (ToS) bits or other header fields (such as IP addresses and port numbers) to map data packets to the appropriate traffic classes. Then the routers schedule amongst the queues based on static priority or fixed weights. In contrast, DaVinci has a link coordinator that assigns the scheduling weights *dynamically*, to maximize aggregate performance. In addition to having per class queuing, the routers could compute separate forwarding tables for each traffic class [61]; this idea has recently enjoyed renewed interest at the IETF in the form of "multi-topology routing," where routers can run multiple instances of the same routing protocol [72]. Recent research has shown that running two instances of OSPF, with link weights tuned to different application performance objectives, offers significant performance benefits over a single protocol instance [54]. Similar to [54], we consider two traffic classes:delay-sensitive traffic and throughput-sensitive traffic. In contrast, we do not consider the delay-sensitive traffic to have higher priority, and we model the throughput traffic as elastic. DaVinci takes these ideas one step further by allowing each virtual network to run its own suite of traffic-management protocols.

5.7.2 Overlay Networks

Overlay networks are a broad concept, including essentially any network constructed on top of another network. In recent years, the term has been used to describe networks composed of end hosts or middleboxes that communicate over tunnels that span the underlying Internet. In sharp contrast to research on QoS, overlay networks tend not to require any support from, or changes to, the underlying network. As such, when an overlay node forwards traffic to another overlay node, the packets traverse the physical links without QoS support from the network. In comparison, DaVinci's virtual networks run directly on the substrate nodes and have their own queues and shares of the bandwidth resources.

Overlay networks have been applied to achieve a wide-range of goals, such as improving routing reliability [7], running new congestion-control schemes [90], and deploying new services like multicast [43] or conferencing [22]. Users may connect to an overlay in a variety of ways, such as establishing a tunnel to an overlay node, configuring a Web browser to use an overlay node as a proxy, or DNS redirection [45, 60]. In addition to running customized routing or congestion-control protocols, an overlay can perform its own admission control and packet scheduling to manage its own traffic [83]. The research on overlay networks has, among other things, demonstrated the value of customized traffic-management protocols. In DaVinci, we take these ideas to their logical conclusion by running the customized protocols inside the network and providing separate shares of the underlying resources.

5.7.3 Network Virtualization

Network virtualization builds on several earlier "virtualization" technologies. In order to connect geographically disparate sites, ISPs establish Virtual Private Networks (VPNs) [76], where each VPN has its own virtual links and forwarding tables. Separately, server virtualization has become increasingly common as a way to consolidate services on a single machine, and support migration to alleviate hot-spots in data centers. Recently, router vendors have started supporting virtualization to enable router consolidation [62, 1]. In addition to having separate forwarding tables and links, virtual routers also run their own instance of the routing software. Though limited today to running existing vendor-specific routing software, recent press releases by Cisco and Juniper indicate an emerging commercial interest in supporting router programmability [4, 5]. DaVinci can leverage these technologies to build customized virtual networks.

There are two broad usages of virtual networks: experimental research facilities and platforms for commercial services. The research community has proposed to build experimental test beds that run multiple virtual networks in parallel [13, 86]. Programmability is a key feature of experimental test beds, so that researchers can run their own custom protocols. To ensure experiments are repeatable, static resource partitioning is the natural choice for experimental test beds. ISPs, on the other hand, are interested in improving the user experience. In this scenario, efficient usage of the underlying resources becomes more important, thus DaVinci proposes to dynamically adapt bandwidth shares. Though recent research papers have proposed that ISPs can run multiple virtual networks in parallel, each with customized protocols [26, 9, 87], they focused on high-level issues such as economic incentives. In contrast, this chapter presents a concrete design for efficiently managing resources between multiple virtual networks.

5.8 Conclusions and Future Work

We present DaVinci: a simple, flexible, and efficient architecture for supporting multiple traffic classes. In DaVinci, each virtual network runs customized traffic-management protocols, and a per-link bandwidth coordinator adjusts bandwidth shares across virtual networks. The substrate computes bandwidth shares entirely based on local link loads, imposing no message-passing overhead. A non-work-conserving shaper at each link ensures the virtual networks are isolated between bandwidth share computations.

Though this paper used optimization theory to design and analyze DaVinci, optimization theory is one of many possible tools to enable a grounded discussion of adaptive network virtualization. We believe adaptive network virtualization shows promise as a future architecture and hope our results will encourage more researchers to explore this topic. In the rest of this section, we discuss future work involving relaxing the stability and optimality properties of DaVinci.

5.8.1 Impact of System Dynamics

In the problem formulation of (5.1), we implicitly assumed there is a fixed number of virtual networks, a fixed number of source-destination pairs, and there are no failures in the physical topology. In a real network, system dynamics can potentially impact the stability of the bandwidth allocation algorithm and the distributed protocols that run inside each virtual network. In this subsection, we discuss which of the system dynamics will likely impact DaVinci, thus providing the direction for further experimentation.

The convergence condition in Theorem 5.2.1 states that the adaptation of bandwidth allocation between the virtual networks only occurs when the distributed protocols that run inside each virtual network converge. Adapting the bandwidth shares too frequently can lead to instability, but adapting bandwidth shares too infrequently can lead to inefficiency. To avoid devolving to static resource allocation, it is essential for the distributed protocols inside each virtual network to be tuned for quick convergence.

While feedback delay can affect the convergence properties of the distributed traffic-management protocols, previous work has extensively evaluated the effect of feedback delay and found methods to ensure stability, *e.g.*, [90, 49]. Since the bandwidth shares are adapted based only on local information, their stabilities are unaffected by feedback delay.

Since some of the system dynamics occur on a very long timescale, they do not impact the stability of the bandwidth allocation algorithm as long as the distributed traffic management protocols are well-tuned to converge quickly. At the timescale of days, ISPs may embed new virtual networks, or modify existing virtual network topologies. At the timescale of hours, links and routers in the substrate network might be upgraded or experience failure. These events act as effective "resets" to the initial conditions of the system, thus the distributed traffic-management protocols running in each virtual network will need to converge after such an event occurs.

The key factor that impacts the convergence of the bandwidth shares to an equilibrium value is traffic fluctuations on the timescale of seconds. This includes both flows arriving and departing as well as traffic bursts. In general, there may not be sufficient time for the distributed traffic-management protocol to converge to an equilibrium value. Consequently, the bandwidth shares might not converge to an equilibrium value. The non-work-conserving shaper at each link maintains isolation between virtual networks at some timescale, so that transient traffic bursts in one virtual network do not compromise the other virtual networks. For flow-level dynamics, recent advances in optimization theory have shown that the queue lengths in the networks will not grow to infinity [82]. In addition, past studies of distributed traffic-management protocols derived from optimization decomposition [90, 40] have performed packet-level evaluations with realistic topologies, delay values and traffic models. These studies have found that distributed traffic-management protocols can stay close to the achievable performance for each instance of time (which has a specified traffic pattern). Still, extensive evaluation of DaVinci is required to understand whether it tracks the ideal bandwidth shares closely under traffic shifts.

5.8.2 Extensions to DaVinci

An even more interesting direction is to study the trade-offs faced by DaVinci when important assumptions are relaxed.

What if a traffic class wants guarantees? So far, we have assumed the traffic in DaVinci is elastic and therefore does not require hard guarantees. In reality, some traffic is inelastic and might want minimum guarantees. In response, the bandwidth reservation mechanisms could be applied to DaVinci at two different levels. At a higher level, some virtual networks might be allocated a minimal guaranteed bandwidth with the excess bandwidth shared between remaining virtual networks. Within each virtual network, providing per-flow-guarantees is also possible by running per-flow admission control, signaling and bandwidth reservation protocols. To support per-flow-guarantees, however, the substrate would need to provide per-flow queues, which is significantly less scalable than per traffic-class queues.

What if some virtual networks are running non-optimized protocols? In DaVinci, we assume the distributed traffic-management protocols inside each virtual network are optimized for a distinct performance objective. In reality, an ISP might want to use simpler traffic-management protocols such as OSPF or TCP Reno, and simply tune them for a particular traffic class. In fact, recent work has studied running two instances of OSPF in parallel, each tuned to delay-sensitive and throughput-sensitive traffic respectively, with strict priority given to the delay-sensitive traffic [54].

In our ongoing work, we are interested in exploring the trade-off between simplicity of the traffic-management protocols and the aggregate performance of the system. In particular, a concrete starting point would be to evaluate running the two instances of OSPF with our link coordinator computing the bandwidth shares, and compare the results to [54].

What if the virtual networks span several ISPs? So far, we assume that DaVinci is deployed by a single ISP with control over both the substrate and all the virtual networks. In reality, ISPs might host virtual networks as a service and therefore virtual networks might span several substrate networks [26]. In this environment, virtual networks belonging to different entities can coexist, raising concerns about greedy and malicious behaviors. Fortunately, the bandwidth shares in DaVinci are directly dependent on the virtual link loads, making it impossible for a virtual network to acquire more bandwidth without inflating its own virtual link load, which would negatively impact its own performance. Still, a malicious virtual network can reduce the bandwidth available to other virtual networks by inserting bogus traffic or introducing instability. To protect against a malicious virtual network, each virtual network could be charged a price for the bandwidth it is allocated and the instability it introduces.

Chapter 6

Epilogue: Configuration Complexity

In today's traffic management, operators *configure* link weights every few hours, which can be a cumbersome task. This dissertation has kept configuration complexity in mind from the beginning. The algorithms proposed in this dissertation do not require operators (or the management systems) to tweak routing protocols in response to congestion, so the parameters are set on a much longer timescale. Further, we used the sensitivity of tuning parameters as a comparison metric in comparing between multiple distributed protocols.

In the process of designing and analyzing traffic-management protocols, we have identified two distinct types of protocol parameters: control variables and granularity parameters. *Control variables* are inputs to a protocol which directly affect the equilibrium behavior of the protocol, *e.g.*, link weights in OSPF. In contrast, *granularity parameters* affect the dynamic behavior of the protocol, *e.g.* stepsizes in TRUMP. In the context of an optimization problem, the control variables appears as part of the problem formulation, while the granularity parameters (such as stepsizes and timescales) are only part of the distributed solution. Consequently, the control variables affect efficiency (at equilibrium), and the granularity parameters affect rate of convergence towards equilibrium. Table 6.1 summarizes the control variables and granularity parameters which have appeared earlier in this dissertation.

Algorithm	**Control Variable**	**Granularity Parameters**
DUMP	U	β_s
Partial-dual	U, f, w	β_s
Primal-dual	U, f, w	β_s, β_y, T_y
Full-dual	U, f, w	β_s, β_p
Primal-driven	U, f, w	β_z
TRUMP	w	β_p, γ
Bandwidth-share adaptation	$U^{(k)}$, $w^{(k)}$	β_s, β_y, T_y

Table 6.1: Control variables and granularity parameters of algorithms presented in this dissertation.

The results presented in this dissertation also hint at a few potentially promising future directions for research on configuration complexity. As a start, control variables and granularity parameters face different configuration challenges. In today's traffic management, configuration of control variables is indirect: link weights are tuned so that congestion is minimized when shortest path routing is used. The *indirectness* of this process makes configuring link weights more challenging than tuning U, w, and f in Chapter 4. On the other hand, today's traffic engineering is centralized, which sidesteps the convergence issues of a distributed algorithm.

In contrast, some of the distributed algorithms (such as DUMP) in Chapter 4 have granularity parameters that are difficult to tune. For algorithms derived from optimization decomposition, the general guideline is that convergence requires small step-size, while fast convergence requires the step-size to be large (subject to convergence). Because tuning the granularity parameters devolves to running simulation experiments that sweep their values, having fewer of them might reduce configuration complexity. Comparing the primal-dual to the partial-dual, we saw that having more granularity parameters did not improve convergence speed, but increased configuration complexity.

Of course, the number of granularity parameters is not the only factor. For example, TRUMP and DUMP both contained a single stepsize, but with dramatically different sensitivities. Though Chapter 4 traded configuring control variables (link weights) with configuring granularity parameters (stepsizes), the overall configuration complexity is reduced. First, these stepsizes are set much less often than link weights, and may have a range of acceptable values. Second, we found a setting for TRUMP's stepsize that works well for a range of topologies, capacities, feedback delays and traffic patterns. Still, more research is require to understand how to find distributed solutions that avoid sensitive parameters.

Although TRUMP is easier to configure than traffic management today, it requires multipath routing and forwarding, which imposes extra overhead on the system. In general, solutions which reduce configuration complexity may pose trade-offs with scalability, though such trade-offs could evolve with time. As technology advances with Moore's Law, previously challenging computations can be done faster. As theory develops, there could be better understanding of configuration complexity, and improved guidance for protocol design.

Bibliography

[1] Juniper Networks: Intelligent Logical Router Service. `http://www.juniper.net/solutions/literature/white_papers/200097.pdf`.

[2] Wikipedia: Congestive Collapse. `http://en.wikipedia.org/wiki/Congestion_collapse`.

[3] Wikipedia: History of the Internet. `http://en.wikipedia.org/wiki/History_of_the_Internet`.

[4] Cisco opening up IOS, December 2007. `http://www.networkworld.com/news/2007/121207-cisco-ios.html`.

[5] Partner Solution Development Platform Opens Opportunity to Accelerate the Pace of Network Innovation with JUNOS Software, December 2007. `http://www.juniper.net/company/presscenter/pr/2007/pr-071210.html`.

[6] Abilene Backbone. `http://abilene.internet2.edu/`.

[7] D. Andersen, H. Balakrishnan, M. Kaashoek, and R. Morris. Resilient Overlay Networks. In *Proc. Symposium on Operating Systems Principles*, October 2001.

[8] E. Anderson and T. Anderson. On the stability of adaptive routing in the presence of congestion control. In *Proc. IEEE INFOCOM*, April 2003.

[9] T. Anderson, L. Peterson, S. Shenker, and J. Turner. Overcoming the Internet impasse through virtualization. *Computer*, 38(4):34–41, April 2005.

[10] D. Awduche, L. Berger, D. Gan, T. Li, V. Shrinivasan, and G. Swallow. RSVP-TE: Extensions to RSVP for LSP Tunnels. RFC 3209, December 2001.

[11] S. Balon, F. Skive, and G. Leduc. How well do traffic engineering objective functions meet TE requirements? In *Proc. IFIP Networking*, May 2006.

[12] T. Bates and Y. Rekhter. Scalable support for multihomed multi-provider connectivity. RFC 2260, January 1998.

[13] A. Bavier, N. Feamster, M. Huang, L. Peterson, and J. Rexford. In VINI Veritas: Realistic and controlled network experimentation. In *Proc. ACM SIGCOMM*, August 2006.

[14] D. Bersekas. *Nonlinear Programming.* Athena Scientific, second edition, 1999.

[15] D. Bersekas and R. Gallager. *Data Networks.* Prentice Hall, second edition, 1992.

[16] D. Bersekas and J. Tsitsiklis. *Parallel and Distributed Computation: Numerical Methods.* Athena Scientific, second edition, 1997.

[17] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, and W. Weiss. An Architecture for Differentiated Services. RFC 2475, October 1998.

[18] R. Braden, D. Clark, and S. Shenker. Integrated Services in the Internet Architecture: An Overview. RFC 1633, June 1994.

[19] R. Callon. Use of OSI IS–IS for Routing in TCP/IP and Dual Environments. RFC 1195, December 1990.

[20] S. Chen and K. Nahrstedt. An overview of quality-of-service routing for the next generation high-speed networks: Problems and solutions. *IEEE Network Magazine*, 12(6):64–79, November 1998.

[21] M. Chiang, S. Low, R. Calderbank, and J. Doyle. Layering as optimization decomposition. *Proceedings of the IEEE*, 95(1):255–312, January 2007.

[22] Y. Chu, S. Rao, S. Sechan, and H. Zhang. Enabling conferencing applications on the internet using an overlay multicast architecture. In *Proc. ACM SIGCOMM*, August 2001.

[23] S. Dasgupta, J. C. de Oliveira, and J.-P. Vasseur. Path-computation-element-based architecture for interdomain mpls/gmpls traffic engineering: Overview and performance. *IEEE Network Magazine*, 21(4):38–45, July 2007.

[24] A. Elwalid, C. Jin, S. Low, and I. Widjaja. MATE: MPLS Adaptive Traffic Engineering. In *Proc. IEEE INFOCOM*, April 2001.

[25] D. Eppstein. Finding k shortest paths. *SIAM J. Computing*, 28(2):652–673, April 1999.

[26] N. Feamster, L. Gao, and J. Rexford. How to lease the Internet in your spare time. *ACM SIGCOMM Computer Communication Review*, 37(1):61–64, January 2007.

[27] A. Feldmann, A. Gilbert, P. Huang, and W. Willinger. Dynamics of IP traffic: A study of the role of variability and the impact of control. In *Proc. ACM SIGCOMM*, August 1999.

[28] S. Fischer, N. Kammenhuber, and A. Feldmann. REPLEX — Dynamic Traffic Engineering Based on Wardrop Routing Policies. In *Proc. CoNEXT*, December 2006.

[29] B. Fortz and M. Thorup. Optimizing OSPF weights in a changing world. *IEEE J. on Selected Areas in Communications*, 20(4):756–767, May 2002.

[30] B. Fortz and M. Thorup. Increasing Internet capacity using local search. *Computational Optimization and Applications*, 29(1):13–48, October 2004.

[31] R. Gao, D. Blair, C. Dovrolis, M. Morrow, and E. Zegura. Interactions of Intelligent Route Control with TCP Congestion Control. In *Proc. of IFIP Networking*, May 2007.

[32] R. Gibben and F. Kelly. On packet marking at priority queues. *IEEE Trans. Automatic Control*, 47(12):1016–1020, December 2002.

[33] H. Han, S. Shakkottai, C. Hollot, R. Srikant, and D. Towsley. Multi-Path TCP: A Joint Congestion Control and Routing Scheme to Exploit Path Diversity on the Internet. *IEEE/ACM Trans. Networking*, 14(6):1260–1271, December 2006.

[34] J. He, M. Bresler, M. Chiang, and J. Rexford. Rethinking Traffic Management: From Multiple Decompositions to A Practical Protocol, March 2007. Princeton University CS Tech. Report TR-774-07. `www.cs.princeton.edu/research/techreps/TR-774-07`.

[35] J. He, M. Bresler, M. Chiang, and J. Rexford. Towards Robust Multi-layer Traffic Engineering: Optimization of Congestion Control and Routing. *IEEE J. on Selected Areas in Communications*, 25(5):868–880, June 2007.

[36] J. He, M. Chiang, and J. Rexford. DATE: Distributed Adaptive Traffic Engineering. Poster session at INFOCOM 2005.

[37] J. He, M. Chiang, and J. Rexford. Can Congestion Control and Traffic Engineering Be at Odds? In *Proc. IEEE GLOBECOM*, November 2006.

[38] J. He, M. Chiang, and J. Rexford. TCP/IP Interaction Based on Congestion Price: Stability and Optimality. In *Proc. International Conference on Communications*, June 2006.

[39] J. He and J. Rexford. Towards Internet-wide Multipath Routing. *IEEE Network Magazine*, 22(2):16–21, March 2008.

[40] J. He, M. Suchara, M. Bresler, M. Chiang, and J. Rexford. Rethinking Internet Traffic Management: From Multiple Decompositions to A Practical Protocol. In *Proc. CoNEXT*, December 2007.

[41] G. Iannaccone, C.-N. Chuah, S. Bhattacharyya, and C. Diot. Feasibility of IP restoration in a tier-1 backbone. *IEEE Network Magazine*, 18(2):13–19, March 2004.

[42] S. Jamadagni and C. Praveen. OSPF extensions for flexible CSPF algorithm support. RFC 2026, October 2002.

[43] J. Jannotti, D. Gifford, K. Johnson, M. Kaashock, and J. O'Toole. Overcast: Reliable multicasting with an overlay network. In *Proc. Operating Systems Design and Implementation*, October 2000.

[44] U. Javed. Modelling and Simulating a Distributed Delay-Sensitive Traffic Protocol. Internal Tech Report.

[45] D. Joseph, J. Kannan, A. Kubota, K. Lakshminarayanan, I. Stoica, and K. Wehrle. OCALA: An architecture for supporting legacy applications over overlays. In *Proc. Networked Systems Design and Implementation*, May 2006.

[46] S. Kandula, D. Katabi, B. Davie, and A. Charny. Walking the Tightrope: Responsive Yet Stable Traffic Engineering. In *Proc. ACM SIGCOMM*, August 2005.

[47] D. Katabi, M. Handley, and C. Rohrs. Congestion Control for High Bandwidth-Delay Product Networks. In *Proc. ACM SIGCOMM*, August 2002.

[48] F. Kelly, A. Maulloo, and D. Tan. Rate control for communication networks: Shadow prices, proportional fairness and stability. *J. of Operational Research Society*, 49(3):237–252, March 1998.

[49] F. Kelly and T. Voice. Stability of end-to-end algorithms for joint routing and rate control. *ACM SIGCOMM Computer Communication Review*, 35(2):5–12, April 2005.

[50] R. Keralapura, C.-N. Chuah, N. Taft, and G. Iannaccone. Can coexisting overlays inadvertently step on each other. In *Proc. International Conference on Network Protocols*, November 2005.

[51] P. Key, L. Massouli, and D. Towsley. Path selection and multipath congestion control. In *Proc. IEEE INFOCOM*, May 2007.

[52] F. Kuipers, T. Korkmaz, M. Krunz, and P. V. Mieghem. Overview of constraint-based path selection algorithms for QoS routing. *IEEE Communication Magazine*, 40(12):50–55, December 2002.

[53] A. Kvalbein, T. Cicic, and S. Gjessing. Post-failure routing performance with multiple routing configurations. In *Proc. IEEE INFOCOM*, May 2007.

[54] K.-W. Kwong, R. Guerin, A. Shaikh, and S. Tao. Improving service differentiation in ip networks through dual topology routing. In *Proc. CoNEXT*, December 2007.

[55] A. Lakshmikantha, N. Dukkipati, R. Srikant, N. McKeown, and C. Beck. Performance Analysis of the Rate Control Protocol. In submission. `http://yuba.stanford.edu/rcp/`.

[56] L. Lasdon. *Optimization Theory for Large Systems*. Macmillian, 1970.

[57] X. Lin and N. Shroff. Utility Maximization for Communication Networks with Multi-path Routing. *IEEE Trans. Automatic Control*, 51(5):766–781, May 2006.

[58] S. Low. A duality model of TCP and queue management algorithms. *IEEE/ACM Trans. Networking*, 11(4):525–536, August 2003.

[59] S. Low, L. Peterson, and L. Wang. Understanding Vegas: A duality model. *J. of the ACM*, 49(2):207–235, March 2002.

[60] H. V. Madhyatha, A. Venkataramani, A. Krishnamurthy, and T. Anderson. Oasis: An overlay-aware network stack. In *ACM SIGOPS Operating System Review*, January 2006.

[61] I. Matta and A. U. Shankar. Type-of-service routing in datagram delivery systems. *IEEE J. on Selected Areas in Communications*, 13(8):1411–1425, October 1995.

[62] D. McPherson, D. O'Leary, D. Ward, E. Brendel, O. Aruj, P. Agarwal, R. Hartani, and S. Poretsky. Core Network Design and Vendor Prophecies. In *Proc. NANOG*, June 2003.

[63] J. McQuillan and D. Walden. The ARPA network design decision. *Computer Networks*, 1(5):243–289, August 1977.

[64] M. Mitzenmacher. The power of two choices in randomized load balancing. *IEEE Trans. Parallel Distrib. Syst.*, 12(10):1094–1104, October 2001.

[65] J. Mo and J. Walrand. Fair End-to-end Window-based Congestion Control. *IEEE/ACM Trans. Networking*, 8(5):556–567, October 2000.

[66] MOSEK Optimization Software. `http://www.mosek.com/`.

[67] M. Motiwala, N. Feamster, and S. Vempala. Path splicing: Reliable connectivity with rapid recovery. In *Proc. SIGCOMM Workshop on Hot Topics in Networking*, November 2007.

[68] J. Moy. OSPF Version 2. RFC 2328, April 1998.

[69] F. Paganini. Congestion Control with Adaptive Multipath Routing Based on Optimization. In *Proc. Conference on Information Sciences and Systems*, March 2006.

[70] D. Palomar and M. Chiang. A tutorial on decomposition methods and distributed network resource allocation. *IEEE J. on Selected Areas in Communications*, 24(8):1439–1451, August 2006.

[71] J. Pongsajapan and S. Low. Reverse engineering TCP/IP-like networks using delay-sensitive utility functions. In *Proc. IEEE INFOCOM*, April 2007.

[72] P. Psenak, S. Mirtorabi, A. Roy, L. Nguyen, and P. Pillay-Esnault. Multi-Topology (MT) Routing in OSPF. RFC 4915, June 2007.

[73] Y. Rekhter and T. Li. A Border Gateway Protocol 4 (BGP-4). RFC 1771, March 1995.

[74] J. Rexford. Route optimization in IP networks. In *Handbook of Optimization in Telecommunications*. Springer Science + Business Media, February 2006.

[75] R.Gao, C. Dovrolis, and E. Zegura. Avoiding oscillations due to intelligent route control systems. In *Proc. IEEE INFOCOM*, April 2006.

[76] E. Rosen and Y. Rekher. BGP/MPLS VPNs. RFC 2547, March 1999.

[77] E. Rosen, A. Viswanathan, and R. Callon. Multiprotocol Label Switching Architecture. RFC 3031, January 2001.

[78] S. Savage, A. Collins, E. Hoffman, J. Snell, and T. Anderson. The end-to-end effects of Internet path selection. In *Proc. ACM SIGCOMM*, August 1999.

[79] S. Sinha, S. Kandula, and D. Katabi. Harnessing TCP's Burstiness with Flowlet Switching. In *Proc. SIGCOMM Workshop on Hot Topics in Networking*, December 2004.

[80] N. Spring, R. Mahajan, and D. Wetherall. Measuring ISP Topologies with Rocketfuel. In *Proc. ACM SIGCOMM*, August 2002.

[81] N. Spring, R. Mahajan, D. Wetherall, and T. Anderson. Inferring Link Weights using End-to-End Measurements. In *Proc. Internet Measurement Workshop*, 2002.

[82] R. Srikant. *The Mathematics of Internet Congestion Control*. Birkhauser, 2004.

[83] L. Subramanian, I. Stoica, H. Balakrishnan, and R. Katz. OverQoS: An overlay based architecture for enhancing Internet QoS. In *Proc. Networked Systems Design and Implementation*, September 2004.

[84] K. Tan, J. Song, Q. Zhang, and M. Sridharan. A Compound TCP Approach for High-speed and Long Distance Networks. In *Proc. IEEE INFOCOM*, April 2006.

[85] R. Teixeira, K. Marzullo, S. Savage, and G. M. Voelker. Characterizing and measuring path diversity of Internet topologies. In *Proc. ACM SIGMETRICS*, June 2003.

[86] J. Turner. A proposed architecture for the geni backbone platform. In *Proc. Architecture for Networking and Communications Systems*, 2006.

[87] J. Turner and D. E. Taylor. Diversifying the Internet. In *Proc. IEEE GLOBECOM*, November 2005.

[88] T. Voice. Stability of Multi-Path Dual Congestion Control Algorithms. *IEEE/ACM Trans. Networking*, 15(6):1231–1239, December 2007.

[89] J. Wang, L. Li, S. Low, and J. Doyle. Cross-layer optimization in TCP/IP networks. *IEEE/ACM Trans. Networking*, 13(3):582–595, June 2005.

[90] D. Wei, C. Jin, S. Low, and S. Hegde. FAST TCP: Motivation, architecture, algorithms, performance. *IEEE/ACM Trans. Networking*, 14(6):1246–1259, December 2006.

[91] D. Wendlandt, I. Avramopoulos, D. Andersen, and J. Rexford. Don't secure routing protocols, secure data delivery. In *Proc. SIGCOMM Workshop on Hot Topics in Networking*, November 2006.

[92] D. Xu, M. Chiang, and J. Rexford. DEFT: Distributed exponentially-weighted flow splitting. In *Proc. IEEE INFOCOM*, May 2007.

[93] W. Xu and J. Rexford. MIRO: Multi-path Interdomain ROuting. In *Proc. ACM SIGCOMM*, August 2006.

[94] X. Yang and D. Wetherall. Source selectable path diversity via routing deflections. In *Proc. ACM SIGCOMM*, August 2006.