# On Routing Wide-Area Network Traffic with High Utilization and Low Latency

*Nikola Gvozdiev*

A dissertation submitted in partial fulfillment
of the requirements for the degree of
**Doctor of Philosophy**
of
**University College London**.

Department of Computer Science
University College London

August 20, 2019

I, Nikola Gvozdiev, confirm that the work presented in this thesis is my own. Where information has been derived from other sources, I confirm that this has been indicated in the work.

# Abstract

An ISP's customers increasingly demand delivery of their traffic with low latency. The ISP's topology, routing, and traffic engineering, often over multiple paths, together determine congestion and latency within its backbone. In this thesis we first consider how to measure a topology's capacity to route traffic without congestion and with low latency. We introduce *low-latency path diversity* (LLPD), a metric that captures a topology's flexibility to accommodate traffic on alternative low-latency paths. We find, perhaps surprisingly, that topologies with good LLPD are precisely those where routing schemes struggle to achieve low latency without congestion. We examine why these schemes perform poorly, and offer a new routing system called Low Delay Routing (LDR)—an existence proof that a practical routing scheme can achieve a topology's potential for congestion-free, low-delay routing.

LDR dynamically places aggregates to avoid congesting the network, while also minimizing their completion times, by routing aggregates on paths that minimize end-to-end delay. LDR's centralized controller finds a latency-optimal placement of thousands of aggregates in less than a second, keeping pace with traffic dynamics seen in today's backbones. LDR also exhibits less churn in traffic placement when demand or the network topology changes. In simulations of real-world wide-area topologies, using real-world packet traces, we show that LDR is able to overcome both short and long-term variability in today's backbone traffic and can successfully run links at high utilization without incurring significant queuing delay.

# Impact Statement

Computer networks provide two fundamental resources to end users: *bandwidth*, or how much information the network can carry, and *delay*, or how quickly information propagates. Given that the capacity of desirable low-delay network paths is limited, those two resources are often at odds. While in the past propagation delay was largely ignored, nowadays a new crop of bandwidth-hungry, delay-sensitive applications is forcing network operators to be more flexible and dynamically adjust the routing in their networks in order to minimize delay while maximizing bandwidth.

Maintaining the precarious balance between bandwidth and delay in an environment where the network operator lacks direct control over senders and receivers is an exciting open research problem that we explore in this thesis. We provide an existence proof that it is possible to build a routing system which dynamically balances traffic to both achieve high utilization and low delay with off-the-shelf routing and switching hardware. In addition to obvious commercial benefits for network operators, such a system also has the potential to provide the network's end users with better end-to-end service. But, fundamentally, even the best routing system is limited by the physical topology that it operates on top of. We map out the design space of low-delay routing and explore what network topologies are most amenable to providing bandwidth without sacrificing delay. We expect this analysis to be beneficial inside, as well as outside academic research. We outline a number of new promising research directions that emerge from our work.

# Acknowledgements

I am deeply grateful to my supervisors Mark Handley and Brad Karp for their guidance. Without their experience and unyielding support this thesis would not have been possible. Their deep insights and attention to detail never ceased to amaze me and served to inspire me during my time at UCL. In addition, I would like to thank Stefano Vissicchio whose help and advice were invaluable in shaping the work, as well as our submissions' anonymous reviewers whose feedback was essential.

I offer my thanks to my wife Svetlana and to my parents Boriana and Lyubomir who were always encouraging and understanding. They were with me every step of the way and certainly without their moral support the journey would have been a lot more difficult.

And last, but not least, I would like to thank Lynne Salameh, Astrit Zhushi, Petr Marchenko and Georgios Nikolaidis, my fellow PhD students at UCL, whose friendship I value greatly.

# Contents


**1 Introduction** **16**

1.1 Cutting Latency by Increasing Connectivity . . . 19

1.2 Problem Statement . . . 21

1.3 Thesis Roadmap . . . 21

1.4 Contributions . . . 23

**2 Literature Review** **24**

2.1 Single-path Routing . . . 24

2.2 Routing Over Multiple Paths . . . 28

2.3 Solution Space . . . 32

**3 The Challenges of Routing for Low Latency** **35**

3.1 The Bandwidth-Propagation Delay Tradeoff . . . 35

3.1.1 Greedy Routing and Varied Link Capacities . . . 38

3.1.2 Greedy Routing and Local Aggregates . . . 38

3.1.3 The Need for Non-Greedy Routing . . . 39

3.2 Assessing Topologies' Potential for Low Latency . . . 41

3.2.1 Low-Latency Path Diversity . . . 41

3.2.2 Path Diversity is Hard to Use . . . 43

3.3 The Headroom Dial . . . 50

3.3.1 Headroom vs. Latency . . . 51

3.3.2 How Much Headroom is Needed? . . . 51

3.4 Summary of Findings . . . 58

**4 Routing Goals and Design Overview** **59**

4.1 Requirements . . . 60

4.1.1 Requirement: Explicitly Target Low Delay . . . 61

4.1.2 Requirement: Adapt to Variable Demand . . . 61

4.1.3 Target Behavior . . . 62

4.1.4 Greedy Heuristic or Closer-to-Optimal Solution . . . 63
4.2 LDR's Design . . . 64
4.3 Installing Network State . . . 65

**5 Optimization 67**
5.1 Objective . . . 67
5.2 Minimizing $O_{max}$ Across all Links . . . 71
5.3 Adding Paths Iteratively . . . 73
5.4 Path Addition Heuristic for Large Networks . . . 74
5.5 Prioritizing Traffic in LDR . . . 76
5.6 Reordering, Jitter and Control Plane Overhead . . . 80
5.6.1 Changes in Per-aggregate Demands . . . 81
5.6.2 Changes in Per-aggregate Flow Counts . . . 82
5.6.3 Limited Optimization . . . 82

**6 Characterizing Demand 84**
6.1 Counting Flows . . . 84
6.2 Aggregate Demand . . . 85
6.2.1 Adding Headroom . . . 85
6.2.2 Statistical Multiplexing . . . 86
6.2.3 Predicting Mean Traffic Level . . . 87
6.2.4 Assessing Link Multiplexing . . . 87
6.3 Dealing With Unexpected Variability . . . 91
6.3.1 Triggered Optimization . . . 91
6.3.2 Low-priority Marking . . . 92
6.3.3 Triggered Optimization and Headroom . . . 92

**7 Evaluation 94**
7.1 The Impact of Latency on Path Selection . . . 95
7.2 Generating Traffic Matrices . . . 98
7.3 Static Components of LDR . . . 101
7.3.1 Low vs. High LLPD . . . 101
7.3.2 Performance Under Varied Load and Locality . . . 103
7.3.3 Fraction of Flows Routed on Shortest Path . . . 105
7.3.4 Absolute delay . . . 106
7.3.5 Path Count . . . 108
7.3.6 Runtime . . . 111
7.3.7 Suboptimality of LDR . . . 112

7.3.8 Reordering and Jitter . . . 113
7.3.9 Prediction Algorithm . . . 115
7.4 Short and Long-Term Variability in Demand . . . 117
7.4.1 Performance of the Convolution Algorithm . . . 117
7.4.2 Predictability of Short-term Variability . . . 119
7.4.3 Long-term Variability and Headroom . . . 121
7.4.4 Triggered Optimization and Limited Optimization . . . 122

**8 Conclusions** **125**
8.1 Limitations in LDR's Evaluation . . . 126
8.2 Modern enterprise networks. . . . 126
8.3 Resilience to Failures . . . 127
8.4 Influence of Routing on Topology . . . 128
8.4.1 Multi-step Upgrade Using LLPD . . . 129
8.4.2 Single-step Upgrade . . . 129
8.5 Future Research . . . 131
8.6 Closing Remarks . . . 133

**Bibliography** **135**

# List of Figures


1.1 A subset of NTT's European network. Drawn to scale. . . . . 19

2.1 All links have the same bandwidth; may need to route packets from aggregates A→D and E→D over both the top path ($A \to B \to C \to D$) and the bottom path ($E \to B \to F \to G \to D$) to avoid congestion. . . . . 25

2.2 Figure 2.1 with one VRF per aggregate at $B$; each outgoing link is associated with two different costs, depending on the VRF. Two different shortest-path trees will be used; the paths taken by each aggregate are highlighted. . . . . 27

2.3 Figure 2.1 with link weights for ECMP; the ".5" labels indicate that traffic is split evenly among outgoing links; with ECMP packets from A→D and E→D will share $A \to B \to C \to D$ and $E \to B \to F \to G \to D$ which have the same weight. . . . . 28

2.4 Link-based MinMax formulation: $A$ and $L$ are the sets of all aggregates and links respectively, the demand of an aggregate $a$ is $B_a$ and the capacity of a link $l$ is $C_l$. We want to find the each aggregate's flow $f_a(i, j)$ that needs to be sent over each link $(i, j)$ which will minimize the maximum link utilization $U_{max}$. One variable per link, per aggregate. . . . . 30

2.5 All links are 10 Gbps; MinMax can fail to minimize utilization in links with utilization below the utilization of the link with minimum maximal utilization; the MinMax utilization is 0.9, the split of bottom aggregate is undefined. . . . . 30

2.6 All links are 10 Gbps; the MinMax utilization is 0.9 regardless of path choice and splits; with MinMax among the many solutions of the same minimal utilization, it is undefined whether packets take the high or the low delay path. . . . . 31

2.7 Path-based MinMax formulation; $P_{al}$ is the set of aggregate $a$'s paths that cross link $l$ and the variable $x_{ap}$ is the fraction of $a$ that goes on path $p$. One variable per path. 31

2.8 A map of the solution space of routing based on the routing system's objective; systems that directly control traffic sources in enterprise environments are colored red. The positions of the points are notional and do not correspond to specific quantitative values. . . . . 33

3.1 CDF of the fraction of links that, when removed from each topology, converts it into a spanning tree; 237 real-world topologies from the Topology Zoo [50] dataset. 36

3.2 Simple scenario drawn to scale; all links have capacity of one unit. SP routing with ECMP, MPLS-TE, and B4 exhibit congestion when a new link is added whether cost is hop count or propagation delay; these routing schemes fail to fit offered demand because they *greedily* place each aggregate's flows onto their shortest paths first. . . 37

3.3 Greedy routing gets stuck in local minimum, fails to avoid congestion; all links have the same delay and the same unit capacity, except for $A \rightarrow D$, whose capacity is 2 units. . . . . 38

3.4 Greedy routing yields high delay on mesh-like networks; all links have a capacity of one unit and the two aggregates have a demand of one unit. The long-haul aggregate (shown in blue) has a second best path that is of slightly longer latency than its shortest path, while the second best path of the local aggregate (show in red) is significantly worse. . . . . 39

3.5 Congestion-free solution that is unattainable by SP routing regardless of assignment of weights; thick lines have a capacity of 2 units. . . . . 39

3.6 CDF curves of APA for all networks, given path stretch limit of 40%. Five random curves are highlighted. The vertical line at 0.7 indicates PoPs 70% of whose shortest-path links can be routed around without excessive delay. . . . . 42

3.7 GTS's Central Europe topology . . . . . 44

3.8 Networks with high LLPD tend to concentrate traffic when using SP routing. The $x$-axis shows the topologies sorted according to their LLPD, and the $y$-axis reports the fraction of source-destination pairs that experience congestion. . . . . 44

3.9 Effects of active routing on congestion and delay. The top part of each graph shows the fraction of all non-zero demands in the traffic matrix that end up crossing at least one congested path, the bottom part shows latency stretch. For each $x$ value (different topology) we plot the median and 90th percentile from runs across a range of traffic matrices. The gray line indicates the span of the distribution. . . . . 46

3.10 Inadvertent congestion on GTS using B4. The greedy nature of B4's path allocation causes both directions of the first link on the $V \rightarrow G$ path to quickly become saturated, at which point there are no alternative paths for the $V \rightarrow G$ traffic. . . . 47

3.11 Excessive latency in the GTS topology using B4. As much as possible of each aggregate is routed on the shortest paths (the two solid lines) causing fully allocated links (like the one labeled) to be shared between the two aggregates; traffic from both aggregates is then sent on second-best paths. Note that even though the second best path of the red aggregate has comparable delay to its best path, the second best path of the blue aggregate is significantly longer—it would have been better to route more of the red aggregate on its second-best path. This real-world example is reminiscent of the synthetic one presented in Figure 3.4. . . . . 48

3.12 Link utilization in GTS. . . . . 50

3.13 Latency stretch as headroom is increased. . . . . 51

3.14 Minute to minute change of mean traffic level in the CAIDA dataset . . . . . 52

3.15 Minute to minute change of standard deviation. . . . . 53

3.16 A short-term spike in traffic level; from 2016 CAIDA packet trace of uncongested U.S. Tier-1 ISP link. . . . . 54

3.17 Hurst parameters at different chunk sizes . . . . . 56

3.18 Hurst parameters at 100ms bin size . . . . . 57

3.19 A minute from a combined hour-long trace with H = 0.8 . . . . . 57

4.1 MinMax and B4 do not explicitly target low delay, and so do not achieve minimal-delay paths. All links are of 1 Gbps capacity. . . . . 60

4.2 The solution to the left is the same as the ideal one in Figure 4.1d. The solution to the right has the same total propagation delay as the one on the left, but is less desirable due to excessive stretch. . . . . 62

4.3 High-level operation: controller and ingress points. . . . . 64

5.1 All links are 10 Gbps; $O_{max} = 2$, attained at the top links. If the optimizer uses the objective in 5.1 the bottom aggregate's traffic may end up congesting one of the bottom paths even though there is capacity for it to fit. . . . . 69

5.2 Same scenario as in Figure 5.1, but the bottom aggregate's demand is $30 Gbps$ and not satisfiable by the two bottom paths. This creates two regions of different oversubscription in the network. A single application of Equation 5.3 will minimize oversubscription in the top part of the network, but not in the bottom one. . . . . 72

5.3 Obtaining paths and per-path aggregate fractions, assuming each aggregate's demand is known. This is a two-stage process: the inner loop uses Equation 5.3 to minimize both delay and oversubscription for a given set of paths (Section 5.2) and the outer loop adds new paths (Section 5.3). . . . . 72

5.4 A simple pathological example. All links have the same 20$Gbps$ capacity and there is a single aggregate whose shortest path passes through $C$; the top path via $E$ has longer propagation delay than any other path from $A$ to $B$. LDR's iterative path addition process will have to add all paths that go through the densely connected region before the only viable path via $E$ is discovered. . . . . 74
5.5 Adding paths to a single aggregate, assuming the aggregate's set of paths is pre-populated to contain its shortest path. When the longest path in the set is oversubscribed, $k$ shortest paths are added until either a non-oversubscribed path is found, the aggregate's path set has grown to the per-aggregate limit ($L$) or the global soft path limit is hit ($L_{soft}$). After either of those limits is reached paths are skipped to avoid exploring densely connected regions as in Figure 5.4. No paths are added if the hard limit ($L_{hard}$) is reached. . . . . 75
5.6 Optimization process with extended aggregates; each aggregate is defined as a combination of $\langle ingress, egress, filter \rangle$. . . . . 77
5.7 Different priority modifiers. . . . . 78
5.8 Effect of a single large aggregate's demand decrease . . . . . 81
5.9 Effect of a single aggregate's flow count change . . . . . 82

6.1 Picking rates to cope with short-term variability . . . . . 86

7.1 Ladder topology . . . . . 95
7.2 Ladder topology MinMax / MinMaxK10 . . . . . 95
7.3 Ladder topology B4 . . . . . 95
7.4 Ladder topology LDR . . . . . 95
7.5 Ladder topology completion times . . . . . 96
7.6 Cumulative fraction of total volume in Cogent's topology that travels a given shortest-path distance. . . . . 100
7.7 Maximum flow stretch; LLPD $<$ 0.5; no headroom . . . . . 101
7.8 Maximum flow stretch; LLPD $>$ 0.5; where the CDF fails to reach 1.0, this indicates that in the remaining scenarios the routing system could not find a placement that would fit all the traffic . . . . . 102
7.9 Maximum flow stretch under different load and locality values; LLPD $>$ 0.5; no headroom; where the CDF fails to reach 1.0, this indicates that in the remaining scenarios the routing system could not find a placement that would fit all the traffic 104
7.10 Fraction of flows that are routed on the shortest path under different load and locality values; LLPD $>$ 0.5; no headroom . . . . . 106
7.11 Absolute stretch in median topologies; only showing the top 20% of each distribution. 107

7.12 Maximum path count under different load and locality values; LLPD $> 0.5$; no headroom . . . 109

7.13 Fraction of aggregates that have only one path under different load and locality values; LLPD $> 0.5$; no headroom . . . 110

7.14 Runtime of optimization algorithms. Each point is the runtime of running LDR with and without $k$ shortest paths caching on a traffic matrix from the set of results that are shown in Figures 7.8 to 7.13. We also present the runtime of a traditional link-based multi-commodity flow formulation. . . . 111

7.15 Sub-optimality with LDR. Plot shows median latency stretch at 39% and 40% headroom. The two curves are for LDR and MinMax, which is the optimal multi-commodity flow solution that minimizes link utilization. . . . 112

7.16 CDF of the fraction of total network volume that changed paths. Each point is a separate traffic matrix, the load of whose aggregates is randomly uniformly distributed +/- 5%. . . . 113

7.17 CDF of the fraction of total network volume that moved to shorter paths. . . . 114

7.18 CDF of the total number of paths updated. . . . 115

7.19 CDF of max single-aggregate volume change . . . 115

7.20 Predictions of mean traffic level (Tier-1 ISP) . . . 116

7.21 Perfect next-minute mean level prediction; convolution algorithm has access to current traffic counters. Mean link utilization (left) and maximum queue size (right) for the first minute. Links are ranked based on the utilization of LDR (NC). . . . 118

7.22 Perfect next-minute mean level prediction; convolution algorithm uses previous minute's traffic to assess short-term variability. Mean link utilization (left) and max queue size (right) for the first three minutes. Links are ranked based on the utilization of LDR (NC). . . . 119

7.23 Traffic that crosses link rank 20 from Figure 7.22; time is in milliseconds; traffic is binned in 100 ms bins and each point is the mean of a bin. In this experiment LDR is given the exact mean traffic levels for the upcoming minute, but this knowledge of the future is of little use as the unexpected change happens mid-minute. . . . 120

7.24 155 sec to 161 sec zoomed in from Figure 7.23; time is in milliseconds; bin size is 10ms. . . . 120

7.25 Increase in delay due to adding a fixed amount of headroom to all links. . . . 121

7.26 Mean link utilization (left) and max queue size (right) for the first ten minutes; 10% headroom target for the mean level estimation algorithm. Links are ranked based on the utilization of LDR (NC). . . . 122

7.27 Mean link utilization (left) and max queue size (right) for the first ten minutes; 5% headroom target for the mean level estimation algorithm. Links are ranked based on the utilization of LDR (NC). . . . . 123
7.28 Mean link utilization (left) and max queue size (right) for the first ten minutes; 10% headroom target for the mean level estimation algorithm; limited optimization enabled. Links are ranked based on the utilization of LDR (NC). . . . . 124

8.1 Same shortest-path routing data as in Figure 3.8, but with Google's topology (LLPD = 0.875) added. . . . . 126
8.2 Latency benefits of network growth; graph shows median and 90th percentile of path stretch before and after growing networks to increase their LLPD; each letter is a different topology: Packetexchange (P), Deutsche Telekom (D), Hurricane Electric (H) and Tinet (T). . . . . 128
8.3 Fraction of flows whose delay increases/decreases when an existing link is upgraded in Hurricane Electric's network . . . . . 130
8.4 Fraction of flows whose delay increases/decreases when a new link is added to Hurricane Electric's network . . . . . 131
8.5 AS-level topology, ingress and egress devices shown. . . . . 132

# List of Tables

5.1 Limits used when adding paths . . . . . 76

7.1 Route changes sent by the controller during the simulation from Figure 7.26. . . . . 123
7.2 Route changes sent by the controller during the simulation from Figure 7.26, but with limited optimization enabled. . . . . 124

## Chapter 1

# Introduction

In recent years, low-latency communication has taken on a new importance. In a widely publicized study a decade ago Amazon found that increasing latency by 100 ms reduces revenue by millions of dollars [74]. Today, as desires of users have evolved, lower delay has become even more crucial from financial services installing microwave towers across Europe [6] and willing to invest billions to reduce the round trip time between London and Tokyo [5], to online gaming companies building their own backbone just to shave off several milliseconds of the delay across the US [33]. Given this strong economic incentive, it is only natural that lowering delay has received a lot of attention from both industry and the research community [75]. Different approaches to reducing delay can be grouped into three main categories.

The first category consists of solutions that focus on making transport protocols better suited for low latency communication across the Internet. Much effort has been put into improving TCP loss recovery [28], and deploying congestion control that tries not to build queues [17]. New transport protocols for the web are rapidly being deployed [53], specifically tailored to reduce the number of round trip times small web requests make.

A second category looks at the datacenter. User requests will often trigger distributed computation (*e.g.,* a map reduce) across multiple machines physically located within the same datacenter, or even within the same rack. Due to the very low round trip times and high capacity between those machines, custom solutions have been devised to increase the utilization of the network while avoiding transient congestion due to effects such as incast [18]. There has been much research ranging from prioritizing certain types of traffic [2] to latency-minimizing datacenter-specific congestion control mechanisms [4] and protocols [10, 39].

A third category focuses on the wide area. While content providers have invested great effort to move static content closer to users, a lot of dynamically generated traffic is likely to still need to cross the Internet. This involves potentially traversing multiple ISPs' networks where the user's packets are at the mercy of the paths chosen by the routing scheme employed by the network's administrator. In this thesis we will focus on the latency that a user's traffic experiences while

traversing a single ISP's network. While this may seem like a straightforward routing problem, choosing low-propagation-delay paths is not enough: as queuing inflates latency, a low-latency placement of traffic also must not congest the ISP's network.

Fundamentally, the rate of any data transfer will be limited by the hop with the lowest available capacity along the end-to-end path. This hop, which we will refer to as the *bottleneck*, is where packets experience congestion, queuing delay and potentially drops. Usually the bottleneck hop is close to the end user, with access link speeds a couple of orders of magnitude lower than core links—*e.g.,* a typical 10-100 Mbps broadband access link versus a 40 Gbps core fiber link. By changing its own routing the ISP can influence the propagation delay experienced by packets that cross its network. As long as the bottleneck remains close to the end user, outside of the ISP's own network, such routing change will not have a significant effect on queuing delay. If the ISP, however, makes a routing change that congests one or more of its core links, the bottlenecks of a large number of its customers will be moved from their respective access links to the ISP's own core. In that case the affected customers customers will experience unpredictable queuing and loss. Clearly, ISPs have a strong incentive to avoid congestion within their backbones: customers may jump ship to a competitor if they find their transit traffic experiences significant loss. Given the importance of latency to user experience, competing on latency also ought to help ISPs attract customers. Can a wide-area ISP minimize *both* propagation and queuing delay to offer the lowest possible latency?

A natural approach taken by some network operators today is to use prioritization. Not all traffic is created equal and some traffic types (*e.g.,* non-interactive streaming video) are more tolerant to delay and packet loss than others (*e.g.,* web browsing). If the operator has an intimate knowledge of the traffic that crosses the network, and the traffic itself is readily classifiable, it is possible to give precedence to delay sensitive traffic so that it does not compete with other traffic for network resources, using techniques such as differentiated services [61].

Throughout the bulk of this thesis we will consider the more general problem of providing low latency when low-latency traffic *cannot* be separated from other traffic. As we discuss in Section 5.5, when it is possible to tease apart latency sensitive from latency insensitive traffic, the approaches discussed in this work can either be trivially adapted, or can be used complementary to prioritization-based approaches.

In general, an ISP has two design choices at its disposal that principally determine the congestion and delay experienced by traffic within its backbone: the topology and the placement of traffic on that topology, as determined by some combination of routing and traffic engineering.[1]

## Traffic Placement

As it turns out, ISPs' present-day approach to avoiding congestion within their backbones affords flexibility in path choice that can be harnessed to route traffic so as to minimize latency. ISPs avoid

[1] In the interest of brevity, we will often refer to this combination as routing.

congestion by overprovisioning—by ensuring that link capacities exceed demand. But as adding capacity over the wide-area is very costly, ISPs tend to avoid the high cost of provisioning a *single* path that can carry the entire *aggregate* between one ingress router and one egress router. Rather, they provision *multiple, shared* paths between an ingress and egress, each typically of *different delay,* and use traffic engineering to split aggregates across these paths.

Today's traffic engineering schemes do not minimize the delay experienced by traffic, though—they focus primarily on avoiding congestion in a fashion that can often *lengthen* delay relative to the shortest path. Approaches such as TeXCP [47] and MATE [29] spread traffic across all available paths to maximize the unused capacity on all those paths—and thus place traffic on paths longer than necessary. Approaches for within an enterprise, where demand is easier to predict, such as B4 [45] and SWAN [42] "pack" traffic on an enterprise WAN's links to achieve high link utilization. Normally, filling links to near their capacity risks congestion should demand increase. But B4 and SWAN do not need to cope with demand variability. As the operator controls both end hosts and the WAN's routers, these systems assume rate limits for sources that are known to the routing system. Alas, an ISP has no such control of demand. Shortest-path routing on delay-proportional link metrics concentrates traffic on the shortest path(s), and forces ISPs to overprovision more than they would need to when using the aforementioned traffic engineering schemes, which can fit traffic on multiple unequal-delay paths. (Indeed, the high overprovisioning cost of shortest-path routing is the very motivation for these traffic engineering schemes.) None of these approaches reliably finds delay-minimizing paths, as we illustrate with simple, pedagogical examples in Section 4.1 and in simulations of real-world ISPs' topologies in Chapter 7.

## The Interplay Between Topology and Routing

The ability of the routing system to minimize delay is closely related to the network's topology. Historically, Internet providers have run their backbones so as to provide end-to-end reachability with adequate capacity, with a measure of redundancy for resilience to backbone link failures. This arrangement falls out of the interaction between a backbone's topology and the intra-domain routing system the provider employs. For example, failure resilience requires some degree of path diversity and a routing system that can choose paths, while providing adequate capacity depends on whether the routing system avoids concentrating traffic on some of those paths and congesting them. Given this close interaction, it is natural not only that network topology has influenced routing system design, but also that routing system design influences topology: a provider will deploy links in light of what the routing system will *do* with them.

Early backbones used shortest-path (SP) intra-domain routing; first distance-vector [56], then link-state [57, 60, 16]. These algorithms worked well when relatively sparse topologies were run at low utilization. More recently, cost pressures have pushed ISPs toward higher link utilization. SP routing has a natural tendency to concentrate traffic and cause congestion, so ISPs have augmented

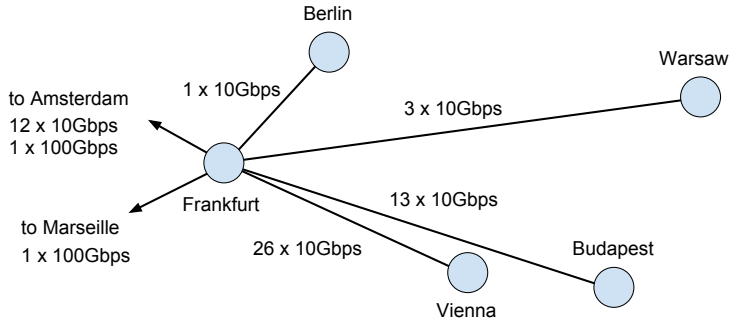



**Figure 1.1:** A subset of NTT's European network. Drawn to scale.

SP routing with traffic engineering (TE) mechanisms such as MPLS-TE [23] that offload traffic onto longer paths. However, the sparseness of topologies has not greatly changed over the last 15 years, as we will explore in Section 4.1. This status quo—SP routing augmented with TE, running over sparse topologies—does a good job of delivering capacity to end users. It also provides a clear path for upgrades: add capacity to links where TE is needed to reduce congestion.

There has been little systematic study of the interaction between a topology's design and the behavior and performance of routing schemes when run on it. A topology's designer must, even if only implicitly, take into account how the routing system will behave on that topology. Similarly, a new routing system's designer would have in mind (again, perhaps implicitly) topologies on which routing should perform well. Each of these approaches starts by fixing a "legacy" design (either the routing or the topology) and attempts to tailor the other to it. If either legacy design isn't a good fit with placing traffic to avoid congestion and achieve low latency, the ability of the ensemble to meet those aims will suffer.

One aim of this thesis is to break with this approach and develop an understanding of exactly which sorts of ISP topologies fundamentally have the potential to deliver traffic with both low propagation and low queuing delay under time-varying demand. As monetary cost is central to how ISPs develop their networks and we have no model for the economic and geopolitical constraints that gate new link deployment, throughout most of this work we will focus our analysis on existing ISP topologies.

## 1.1 Cutting Latency by Increasing Connectivity

One seemingly promising strategy for cutting latency is to introduce links that carry demand along a more direct geographic path, shortcutting a previous, more circuitous one. Unfortunately, legacy SP routing *directly hampers the introduction of latency-cutting links into a backbone's topology,* making it hard to build low-latency, more connected backbones. When augmented by TE, SP fares somewhat better, but as we will show, this combination still falls short.

To see where SP struggles, consider Figure 1.1, which depicts a portion of the European network of NTT, a large global ISP [1]. Most hops in this part of NTT's network consist of bundles of

individual 10 Gbps links. Clearly this structure is the result of incremental upgrades—once a hop starts to approach capacity, NTT adds a new 10 Gbps link.

In NTT's network, note that traffic from Budapest to Vienna must go via Frankfurt, and thus experience higher latency than strictly necessary. Suppose, for the sake of argument, that the traffic from Budapest to Vienna varies between 7 Gbps and 20 Gbps depending on time of day. Suppose further that traffic has grown such that the Budapest → Frankfurt link is running uncomfortably close to capacity. The operator must add capacity. One option is to add another 10 Gbps to the Budapest → Frankfurt link; another is to add a new 10 Gbps link direct between Budapest and Vienna. Both would help with capacity, but the new link would also improve latency and might well be cheaper, as it is shorter.

In practice, though, the routing system's limitations make using this direct link difficult. [2] At off-peak times, the 7 Gbps of traffic fits, but at peak times SP routing would result in heavy congestion as 20 Gbps of demand tries to fit down a 10 Gbps link. What the operator would like is for the 10 Gbps link to run at fairly high utilization all the time, but for excess traffic to take the indirect path—if only 10 Gbps are added there is no solution that satisfies peak capacity with the best possible latency. In this way, capacity is provided cheaply, and at least some traffic sees improved latency.

TE schemes [47, 42, 31, 29, 7] can, in principle, solve this problem. To do so, they would split the Budapest → Vienna traffic unequally among multiple, unequal delay paths as necessary, and do so automatically and dynamically depending on the time of day and level of traffic. However, such schemes primarily concern themselves with *capacity*; none places traffic within a backbone so as to minimize delay *and* fit offered load. Indeed, as we explore in Section 4.1, on mesh-like networks, neither SP routing (with or without ECMP) nor state-of-the-art TE schemes can achieve delay minimization while fitting user demands.

But WAN routing is not only about choosing paths and deciding how to split aggregates among paths. A central challenge when adapting wide-area intra-domain routing to traffic demands is *coping with variability in those demands.* A routing scheme that aims for low-latency traffic placement will have to, by definition, load paths that provide low propagation delay to a very high level of utilization, but it must at the same time remain congestion-free and must not congest links when traffic demands vary over time. A simple way to guard against overloading links when demands increase is to enforce *headroom*: to reserve some minimum fraction of each link's capacity to accommodate foreseen but rare demand increases. Putting capacity aside to soak up demand spikes, however, can be seen as *changing the topology*; a capacity-aware routing scheme may move some traffic to longer paths when the capacity of a short-delay path is "reduced." We explore this interplay, and show that one may view the design space of congestion-free, low-delay routing schemes

[2] NTT may, of course, have other reasons for arriving at this topology. Regardless, adding and using the direct link is difficult.

as falling along a continuum. At one extreme is a notional scheme that employs no headroom on any links—and thus achieves the lowest delay a given topology can offer, at the expense of risking congestion when demand increases. At the other are MinMax schemes, which by definition leave as much headroom on links as possible: as previously mentioned, they spread traffic across multiple paths so as to maximize the free capacity across links, in so-called MinMax fashion. This approach trades away high link utilization to hedge against demand variability: maximizing headroom reduces the chance that a change in traffic causes congestion. In an ISP setting, where demand isn't known perfectly in advance, it is an open question where on this continuum a practical routing scheme should lie. Ideally, there should be a sweet spot with enough headroom to cope with demand variability, yet not so much that paths are needlessly circuitous, incurring high latency.

## 1.2 Problem Statement

Today's routing schemes fall short at extracting path-diverse ISP topologies' potential for low-latency, congestion-free traffic delivery. Can we build a routing system that is able to fully harness this potential?

Delays in computer networks can either be propagation delays or queuing delays. Queuing delay is much less predictable than propagation delay. Can a latency-minimizing routing scheme *minimize propagation delay while avoiding queuing?*

Any delay-minimal routing scheme for the ISP setting must cope with *ever-changing demand*—it cannot rely on fully predictable, controlled demand, as can schemes tailored for the enterprise setting. As there is a limit to how frequently any demand-sensitive routing scheme can change the placement of traffic, to avoid congestion, can a delay-minimizing routing scheme also *be able to predict whether aggregates' variable demands will statistically multiplex on a path in between changes in traffic placement?*

To use low-delay paths in the ISP efficiently, a routing scheme must run a subset of the ISP's links at high utilization. Regardless of how good the routing system's traffic prediction is, doing so under variable, uncontrolled demand risks congestion. Can a routing scheme *adapt quickly when demand for a link bumps up against that link's capacity?*

## 1.3 Thesis Roadmap

We begin by examining, in Chapter 2, prior work in the field of routing and traffic engineering, and how it relates to low-latency routing.

In Chapter 3 we explore how connected today's topologies are. We demonstrate using small-scale synthetic examples that, as backbones get more connected, today's routing and TE systems fail to place traffic onto multiple alternative paths so as to satisfy user demands *and* minimize delay. We then develop a routing-agnostic, first-principles understanding of exactly which sorts of ISP topologies fundamentally have the potential to deliver traffic without congestion and with low

latency under time-varying demand: namely, those with *diverse low-latency paths.* We quantify the extent to which 116 real ISP backbone topologies from the Internet Topology Zoo [50] exhibit this potential. From there, we explore in detail how well today's widely known routing systems manage to exploit these same ISP topologies' inherent potential for congestion-free, low-latency traffic placement. We verify the results from the synthetic examples and find, somewhat surprisingly, that on topologies with diverse low-latency paths—*precisely* those with the greatest potential of this sort—status-quo schemes from shortest-path routing to B4 and MinMax traffic engineering (*e.g.,* TeXCP) arrive at traffic placements that suffer congestion or high latency stretch. We reveal why these routing designs encounter these poor outcomes on these promising topologies.

In Chapter 4 we present the overall goals and the design of Low Delay Routing (LDR), a routing system for ISP backbones that offers users low latency by placing aggregates on paths so as to minimize the total latency experienced by flows, while also avoiding congestion. Ingress routers report aggregate traffic demand measurements to LDR's centralized controller, which periodically computes minimal-delay placements of aggregates on paths. The controller reports these placements to the ingress routers, which forward aggregates' flows accordingly. Should demand abruptly increase beyond expectation, ingress routers trigger an early recomputation of traffic placement.

In Chapter 5 we describe LDR's optimization mechanism, which, given a traffic matrix and a set of demands, is able to quickly produce a solution that avoids congesting links while minimizing propagation delay.

Of course, real demand in a network is never fixed. As previously discussed, any solution which minimizes propagation delay will need to run some links at high utilization. We cannot simply apply LDR's optimization in an uncontrolled ISP-like environment and expect low queuing delay—we need to deal with variability. The mechanisms we describe in Chapter 6 allow LDR to deal with both short- and long-term variability.

In Chapter 7 we evaluate the performance of LDR. Extensive simulations of LDR on real-world ISP topologies [50] show that LDR consistently achieves lower delay than versions of MinMax and B4 that we extend to target delay minimization explicitly. We further explore how *stable* delays experienced by flows are under these schemes as an ISP increases capacity. Our results reveal that, somewhat surprisingly, adding capacity to an ISP's backbone often causes B4 and MinMax to *lengthen* many flows' delays, whereas LDR maintains minimal total delay *and* causes delay increases for far fewer flows. LDR achieves these feats despite the ever-changing traffic demands in the ISP scenario.

In Chapter 7 we additionally demonstrate that LDR's approach to routing can perform well on all topologies, but performs especially well where the topology offers a good diversity of low-delay paths. We speculate that such topologies may be rare today because they have been hard to use effectively with existing routing schemes. The adoption of techniques similar to those presented may eliminate this obstacle to building more "mesh-like" network topologies well suited to low-

latency, congestion-free traffic delivery.

## 1.4 Contributions

As we outline above, at the beginning of this thesis we explore the interplay between the diversity of low-latency paths in a topology and the ability of a routing scheme to exploit that diversity to achieve congestion-free, low-delay traffic delivery. Our main contributions there are:

- revealing the nuanced interaction between a topology's path diversity and routing schemes that aim to deliver low latency without congestion;
- revealing exactly why existing routing schemes cannot unlock the low-latency potential of path-diverse topologies;
- identifying the central role of headroom in effecting a necessary trade-off between avoiding congestion and reducing path latency when traffic demands vary; and
- characterizing an approach to routing that avoids the pathologies that existing approaches fall prey to on path-diverse topologies, that parsimoniously yet safely applies headroom to cope with demand variability, and that is computationally tractable at ISP scale.

We then design and implement LDR—a routing system that actively balances propagation and queuing delay in order to minimize latency. Key contributions in LDR's design include:

- a novel *iterative* formulation of linear optimization for traffic placement whose efficiency renders minimal-delay routing tractable at scale. LDR's optimizer can place thousands of aggregates optimally in a backbone of hundreds of links in less than a second;
- a novel method for predicting how traffic demands will statistically multiplex on a path at a granularity of tens of milliseconds, based on *convolving* aggregates' past demand distributions.

## Chapter 2

# Literature Review

There is a rich literature on protocol and system designs for picking paths and placing traffic on paths so as to meet varied objectives in varied settings. They can be broadly separated into two categories—systems that only route each aggregate's traffic over a single path through the network and systems that can use multiple paths simultaneously for each aggregate. In this chapter we will provide an overview of both categories. Keep in mind that in related work the word *aggregate* is often used to refer to any combination of 5-tuple flows, while in this thesis we use a narrower definition—the collection of flows between a given source-destination pair.

## 2.1 Single-path Routing

The simplest routing schemes always employ a single path between each ingress and egress point in the network. This path is computed so that it minimizes some constant per-link cost, also called a weight. Legacy protocols, such as ARPANET's original routing protocol RIP [57], are based on Bellman-Ford [11] distributed shortest-path computations, and aim to minimize the number of hops traversed by packets. These routing protocols use hop count as the metric, essentially treating all links in the network as having the same positive weight. During operation, network devices periodically exchange all or part of their routing state with their immediate neighbors. For a given destination each device will then send packets to the neighbor that has a lower hop count to the destination than the device's own hop count. Such protocols are called distance-vector protocols because each device needs to maintain a list, or a vector, of hop counts, or distances. Basic distance-vector protocols have the advantage of being simple and easy to implement on routers' often weak control-plane CPU, but also have a number of disadvantages:

- They do not scale well—a Bellman-Ford algorithm implementation needs to pick a maximum hop count value which to consider infinity. This hop count value imposes a limit on the diameter of the network, and it cannot be set to an arbitrarily high value because count-to-infinity problems [65] negatively affect convergence time.

- They can exhibit undesirable transient behavior during convergence—due to the distributed

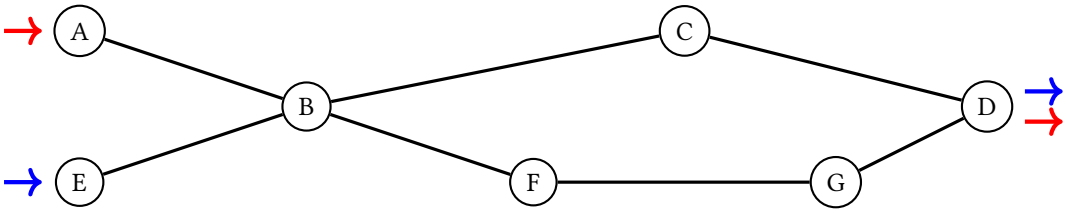



**Figure 2.1:** All links have the same bandwidth; may need to route packets from aggregates A→D and E→D over both the top path ($A \to B \to C \to D$) and the bottom path ($E \to B \to F \to G \to D$) to avoid congestion.

nature of the routing system when a change occurs different routers will learn about it at different times. Routing loops or blackholes may temporarily form until all network devices learn the same state.

Both of those problems are well known and have been extensively studied by previous literature.

A desirable property of distance-vector routing is that if a metric change is received that reduces the distance to the destination, a switch to the new shortest path will not cause a routing loop. If the neighbor sending the change is the new next hop, it must have already applied the update, and so must be closer to the destination. No loop can occur. Provably loop-free distance-vector algorithms such as DUAL [34], which is the basis for Cisco's EIGRP [3], make use of this property to safely apply updates that reduce the distance to a destination, and trigger a diffusing computation [26] for other updates.

More modern incarnations of shortest-path routing that are in wide deployment today (*e.g.,* OSPF [60], IS-IS [16]) address distance vector's scalability issues by maintaining the state of all links in the network in any of the network's devices (and are thus called link-state protocols). When this internal network map changes, each device independently runs Dijsktra's algorithm [25] to obtain the next hop it needs to send packets to each destination. Dijkstra's algorithm can handle networks of arbitrary size and is computationally less complex than Bellman-Ford. Moreover, maintaining the network map needed by the algorithm only requires the exchange of small link-status updates as opposed to distributing potentially large parts of the routing table to all neighbors.

All routing protocols described so far have focused primarily on achieving end-to-end reachability over the single best path, according to some per-link metric. There are lots of real-world scenarios where this is not enough. Consider the small example from Figure 2.1[1]. There are two transit aggregates that need to exit the network via $D$, having entered the network via $A$ and $E$ respectively. Regardless of the paths assigned by the routing system the two aggregates' traffic will combine at $B$. Shortest paths exhibit optimal substructure—a shortest path is composed of other shortest paths. Because of this property if traffic always follows the single shortest path *all* traffic that reaches $B$ will only follow one single exit path to $D$ regardless of how link weights are

[1] This is a common example in routing and traffic engineering known as the "fish problem".

assigned.

Keeping in mind that each link in Figure 2.1 has the same capacity, there are three possibilities for each one of the two aggregates:

1. The aggregate's flows are capped somewhere outside the network (*e.g.,* by low-speed access links) and when combined with the other aggregate's traffic the two aggregates will not cause significant queuing at either $B$ or any device downstream of $B$. In other words, the total sum of both aggregates' demands will not exceed the capacity of a link within the network.

2. The aggregate is similarly externally capped, and it does not congest the ingress port of its ingress node (either $A$ or $E$), but when combined with the other aggregate would cause a downstream queue to build up at $B$. In other words, the total sum of both aggregates' demands will exceed the capacity of a link within the network.

3. The aggregate is not externally capped and there is a persistent ingress queue at its ingress node. In this case the notion of an aggregate's "demand" is not well defined.

In case 1 single shortest-path routing will work well, as all traffic will follow the shortest path tree rooted at $D$ (which is a branch of the SP trees rooted at $A$ and $E$) without causing congestion. The administrator is free to choose whatever assignment of link weights they desire, to "steer" traffic to take either the top path via $C$ or the bottom one via $G$.

The other two cases, however, are more challenging. In case 2 it is possible to avoid queuing altogether—assuming that the $(A,E) \rightarrow B \rightarrow C \rightarrow D$ paths are more desirable, it should be possible to offload some fraction of their traffic onto the less desirable $(A,E) \rightarrow B \rightarrow F \rightarrow G \rightarrow D$, thus relieving the excessive load at $B$. In case 3 queuing cannot be avoided, but it is possible to maximize the amount of traffic the network can handle by spreading the aggregates evenly among the top and bottom paths. In the last two cases there is a fundamental tradeoff that the routing system must make—if all packets are routed on the lowest cost paths there will be congestion, but at the same time any alternative path the routing system routes traffic on in order to reduce congestion will have higher cost. Does the system choose to reduce congestion (or maximize network capacity in the case of 3), or does it choose risk congestion and reduce path cost?

Notice that which of the three cases from Figure 2.1 an aggregate is in depends on the traffic of not only that aggregate, but also on other aggregates in the network. The entity that is in charge of routing in the system, be it an automated system or a human operator, needs to be aware of the *traffic matrix*—the load of each one of all possible $N(N-1)$ aggregates in the network, where $N$ is the number of devices in the network. Without this knowledge it is difficult for a system to pick a non-extreme point in the design space between reducing congestion and minimizing path cost.

Single shortest path routing takes one extreme in this design space by routing everything on the path with the lowest possible cost. An alternative way to handle cases 2 and 3 would be to still

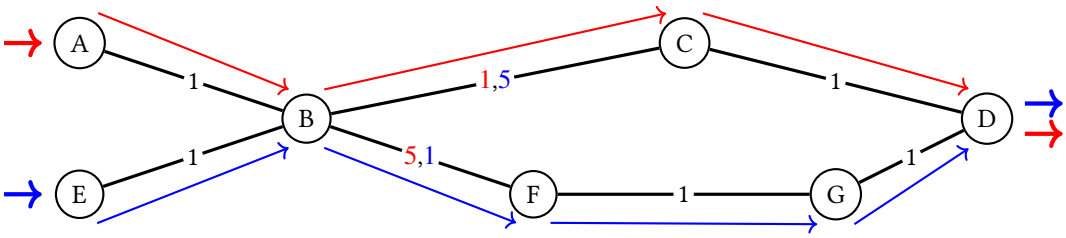



**Figure 2.2:** Figure 2.1 with one VRF per aggregate at $B$; each outgoing link is associated with two different costs, depending on the VRF. Two different shortest-path trees will be used; the paths taken by each aggregate are highlighted.

use shortest-path routing, but run a different routing table for each aggregate at $B$. By assigning different link weights on each outgoing link for each aggregate, the network administrator can then make sure the top path is preferred by one of the aggregates, and the bottom one by the other. Device $B$ will run two instances of the same routing protocol, and will behave as if it were two different devices as far as the routing protocol instances running in the rest of the network are concerned. Figure 2.2 shows how VLAN routing and forwarding (VRF) [69] can handle the simple example from Figure 2.1 in this way. In addition to running two instances of shortest-path routing, either a tagging mechanism (*e.g.*, MPLS [24]) or matching based on input port at $B$ will be needed to route the packets as shown. Note that each aggregate's traffic still takes only one path through the network, and the path is still the notionally shortest from the viewpoint of the algorithm, even though it is not the actual shortest path.

In addition to offering network administrators greater routing flexibility, VRFs also provide a natural extension of VLANs to level 3, allowing different customers' routes to be isolated. The obvious drawback is that they can be hard to configure and require intimate knowledge of the network's operation and traffic patterns. For example if one of the two paths on Figure 2.2 were to have lower capacity the network operator would have to know ahead of time which of the two aggregates is more likely to fit on that path, then closely monitor the network for signs of overload and change weights to adapt the routing as the traffic patterns of the two aggregates change.

What if we could sense the actual demand and have routing automatically reconfigure itself to avoid congestion? MPLS-TE [23] does just that. MPLS-TE combines MPLS tunnels, resource reservation via RSVP [83] and constrained shortest path computations (CSPF).[2]

MPLS-TE, as well as most traffic engineering solutions, routes traffic over tunnels. A tunnel is a unidirectional logical link from ingress to egress; packets are tagged, or labeled, by the ingress upon entering the network and a path is configured that drives the tagged packets to the egress on an arbitrary, not necessarily shortest, path. The label-switched tunnel does not necessarily need to have a globally (network-wide) unique label which will be put on every packet that traverses the path defined by the tunnel—in common labeling technologies, such as MPLS, labels only have

[2]In this thesis we assume automatic bandwidth allocation for MPLS-TE, as it is often used in practice [78, 76].

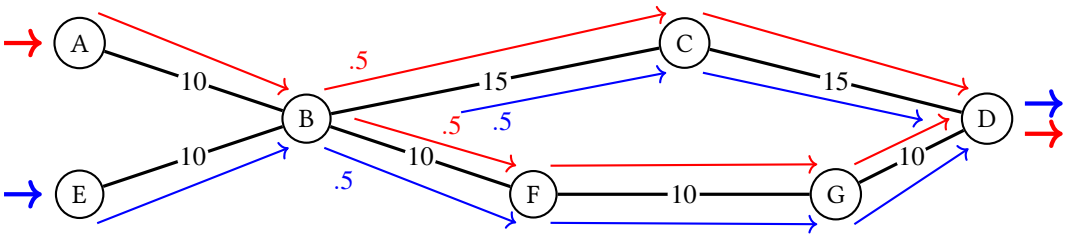



**Figure 2.3:** Figure 2.1 with link weights for ECMP; the ".5" labels indicate that traffic is split evenly among outgoing links; with ECMP packets from A→D and E→D will share $A \to B \to C \to D$ and $E \to B \to F \to G \to D$ which have the same weight.

a local meaning.

In MPLS-TE the ingress[3] at predetermined intervals polls interface counters to determine what the demand of an aggregate is. That information is then shared with other participating nodes using a slightly modified version of either OSPF or IS-IS. When a path computation needs to happen (*e.g.,* when a new tunnel is set up or when a link goes down) constraints are added to regular shortest-path Dijkstra that exclude links with unreserved capacity that is less than the demand of the aggregate. Once a path is found capacity along each hop is reserved for it using RSVP.

The mechanism described above lets MPLS-TE handle automatically the example from Figure 2.2. The network administrator does not have to manually tweak link weights, instead a non-shortest path computation is used to generate a single path for each aggregate. An obvious drawback is that each ingress only updates its estimate of aggregate levels at the end of every period, which in practice defaults to one day [19]. If an aggregate's level suddenly changes it may take a while for the network to adapt to the new level. To address this shortcoming some vendors support features called overflow and underflow, which can trigger a path recomputation at the ingress as soon as it detects that an aggregate's level significantly overshoots or undershoots the previous level. Due to the distributed nature of MPLS-TE, those features can be hard to properly configure [78].

## 2.2 Routing Over Multiple Paths

An alternative way to handle the simple scenario in Figure 2.1, would be to spread each aggregate's traffic across *both* the top and the bottom path. Equal cost multipath (ECMP) does just that, as shown on Figure 2.3. The network administrator has adjusted the weights of the top path, so that it appears to the routing system as desirable as the bottom one. Unlike previously discussed single-path schemes, ECMP can distribute an aggregate's traffic among a number of paths that have equal cost, using a total of four paths on Figure 2.3. Traffic from both aggregates will be split evenly among the top and bottom paths either at $B$, as shown, or at the ingress points $A$ and $E$.

How does $B$ split aggregates among paths? To achieve the best possible split $B$ should send

[3]Here we use "ingress" in the same sense that "head-end" is used in some MPLS literature; "egress" corresponds to "tail-end".

down each path half of the upstream packets it receives. In practice, however, different network paths will have different propagation and queuing delays; each aggregate consists of a number of flows and sending packets from the same flow down different paths can cause reordering when the flow's packets join at the egress of the network. Reordering behaves badly with TCP's congestion control [84], causing the TCP flow to unnecessarily reduce its congestion window. To avoid such undesirable behavior, most network devices will hash each packet's five tuple—a combination of header fields that uniquely identifies the flow the packet is part of—and then probabilistically send flows down paths depending on the hash value and the weight of the path. In the case of ECMP the weights are always $1/N$ where $N$ is the number of paths.

Notice that in the ECMP case the 50/50 splits from Figure 2.3 are always optimal with respect to avoiding queuing *regardless of the traffic matrix*. In all cases for this topology and those two aggregates the equal splits will minimize peak utilization—the less loaded any link in the network is the more headroom there is, and the less likely it is for a fluctuation in the volume of an aggregate to cause queues to build up.

Surprisingly, it is possible to generalize the ECMP observation and, given an arbitrary topology, come up with routing which will provide low peak utilization in a way that is oblivious of the traffic matrix [67, 8]. Unlike ECMP, oblivious routing achieves low utilization by splitting traffic *unevenly* among multiple paths available from an ingress. The utilization provided by such an oblivious system is not as low as the one that can be achieved by utilization-minimizing approaches that are load-dependent, but an oblivious routing solution is more deployable since it does not need to actively measure the traffic matrix.

If the routing system has access to the traffic matrix, it can further decrease peak utilization in arbitrary topologies. The routing system can direct traffic in a way that minimizes some arbitrary per-link cost function of the link's utilization ($f(u)$). Such a utilization-based cost function assumes that the cost of a link grows as the link becomes more and more used. This assumption holds when congestion is within the network (case 3 on Figure 2.1), and much less so when congestion is outside the network (other two cases)—as we will later show, when traffic is externally bottlenecked it is possible to drive a links' utilization close to 90% without incurring queuing at that link. For a specific $f(u)$ the actual routing solution can be found by treating routing as a multi-commodity flow problem [13] or, for continuous and convex functions, using a distributed algorithm [12] inspired by Newton's method.

A large volume of traffic engineering literature [47, 29, 79] picks the per-link cost function $f(u)$ to be the top utilization of any link in the network. This results in what is called a MinMax problem—minimizing the maximum link utilization. An optimal solution to the problem can be obtained by solving the mathematical optimization in Figure 2.4. Constraints 2.2, 2.3 and 2.4 are standard multi-commodity flow preservation constraints that make sure that for each aggregate the flow that enters a node is equal to the flow that exits a node, except for the source and the sink

$$
\begin{aligned}
&\text{minimize:} && U_{max} && \\
&\text{subject to:} && \sum_{a \in A} f_a(i,j) < U_l C_l && \forall (i,j) \in L && (2.1)\\
& && \sum_{v \in V} f_a(i,j) - \sum_{v \in V} f_a(j,i) = 0 && \forall a \in A, \forall (i,j) \in L, i \neq s_a, t_a && (2.2)\\
& && \sum_{v \in V} f_a(s_a,j) - \sum_{v \in V} f_a(j,s_a) = B_a && \forall a \in A, \forall (i,j) \in L && (2.3)\\
& && \sum_{v \in V} f_a(j,t_a) - \sum_{v \in V} f_a(t_a,j) = B_a && \forall a \in A, \forall (i,j) \in L && (2.4)\\
& && U_l < U_{max} && \forall l \in L && (2.5)\\
& && 0 \leq U_{max} \leq 1 &&
\end{aligned}
$$

**Figure 2.4:** Link-based MinMax formulation: $A$ and $L$ are the sets of all aggregates and links respectively, the demand of an aggregate $a$ is $B_a$ and the capacity of a link $l$ is $C_l$. We want to find the each aggregate's flow $f_a(i,j)$ that needs to be sent over each link $(i,j)$ which will minimize the maximum link utilization $U_{max}$. One variable per link, per aggregate.

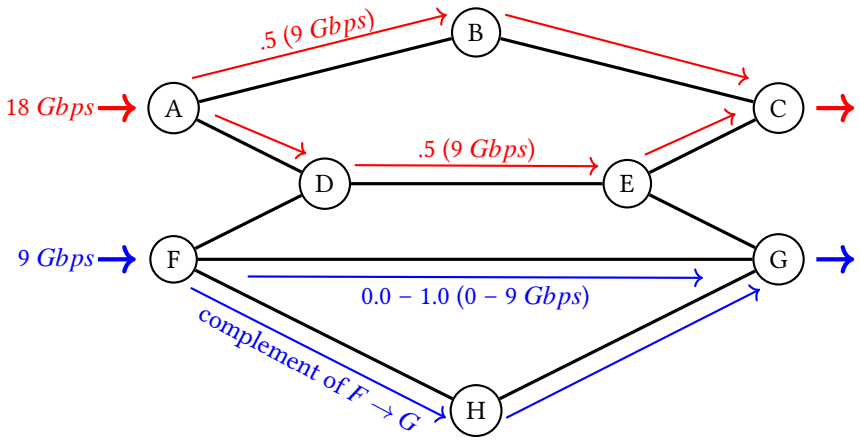



**Figure 2.5:** All links are 10 Gbps; MinMax can fail to minimize utilization in links with utilization below the utilization of the link with minimum maximal utilization; the MinMax utilization is 0.9, the split of bottom aggregate is undefined.

for that aggregate. Constraint 2.1 makes sure that no links exceed their capacity, and the constraint 2.5 makes all link utilization variables less than the maximum link utilization. Upon solving, each aggregate $a$'s paths can be obtained by performing DFS on the DAGs formed by links for which the aggregate's flow is non-zero—*i.e.,* links with positive $f_a$.

Notice that the utilization of links that are below $U_{max}$ is undefined, which may lead to both higher utilization and higher propagation delay than necessary. Figure 2.5 illustrates this point. In the simple topology all links have the same 10 Gbps capacity, and there are only two aggregates, with demands of 18 and 9 Gbps respectively. A MinMax solution will spread out the first aggregate across as many paths as possible, which in this case is the two top paths. The minimum link utilization is therefore 90%. None of the bottom aggregates' flows will be routed on the top links, because that would cause the utilization of those links to go up. The bottom aggregate will have to be then split among the two paths at the bottom. There exist multiple equally good solutions—as

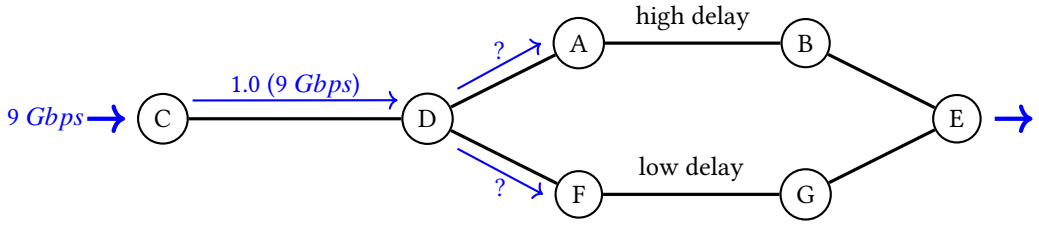



**Figure 2.6:** All links are 10 Gbps; the MinMax utilization is 0.9 regardless of path choice and splits; with MinMax among the many solutions of the same minimal utilization, it is undefined whether packets take the high or the low delay path.

$$
\begin{array}{lll}
\text{minimize:} & U_{max} & \\
\text{subject to:} & \sum_{a \in A} B_a \sum_{p \in P_{al}} x_{ap} < U_l C_l & \forall l \in L \\
& U_l < U_{max} & \forall l \in L \\
& 0 \leq U_{max} \leq 1 & \\
& 0 \leq x_{ap} \leq 1 & \forall a \in A, \forall p \in P
\end{array}
\tag{2.6}
$$

**Figure 2.7:** Path-based MinMax formulation; $P_{al}$ is the set of aggregate $a$'s paths that cross link $l$ and the variable $x_{ap}$ is the fraction of $a$ that goes on path $p$. One variable per path.

long as no more than 9 Gbps are routed on the $F \rightarrow G$ link, the maximum link utilization will not be affected. In reality, the network operator may prefer to also split the bottom aggregate, as that would result in lower maximal utilization of the bottom two links, but the basic MinMax formulation does not allow them to do that.

Similarly, there are common cases where the propagation delay of the MinMax solution can be unnecessarily large. Consider the single-aggregate example from Figure 2.6. Any solution will have the same maximum utilization, and from the viewpoint of avoiding congestion they will all load equally the $C \rightarrow D$ link, and are therefore all equally good. But in this topology the two middle links have significantly different propagation delays—clearly the network operator would like to send as much of the traffic on the lower-delay path.

Another obvious drawback of the traditional MinMax linear programming formulation is that it can be hard to compute a solution—complexity is proportional to the number of variables, and there is one variable per aggregate, per link, resulting in hundreds of thousands of variables even for moderately-sized networks. One approach to reduce complexity taken by related work [47, 29] is to use a path-based formulation instead of a link-based one, where each aggregate is pre-populated with all possible paths and there is a single variable per path. This new, simpler, formulation is shown in Figure 2.7. In this formulation the volume of traffic that crosses a link is expressed as the sum of the traffic of all paths that cross the link, and there is no need for flow preservation constants because paths are explicitly added as part of the problem definition.

The path-based formulation has $O(kN)$ variables where $N$ is the number of aggregates and $k$ the number of paths added to each aggregate. Usually the first $k$ shortest paths are added to each

aggregate, for a relatively small value of $k$—*e.g.,* 10 [47]. This renders the problem significantly easier to compute, but also sacrifices optimality—the two formulations are only guaranteed to always produce the same output when *all* paths are added for each aggregate, which is only feasible for very small topologies. As a result, in more complex topologies the path-based formulation can produce a solution which results in significantly higher utilization, or even a solution that fails to fit some aggregates' demands and causes persistent congestion.

All solutions that split traffic over multiple paths presented so far have been concerned with reducing utilization in order to avoid congestion. A radically different approach is taken by recent enterprise-oriented traffic engineering work such as B4 [45] and SWAN [42]. In the enterprise scenario all endpoints of all connections are controlled by the same organization, which makes it possible to enforce the network bandwidth taken by each connection using systems like BwE [52]. This high degree of control gives the routing system an important advantage—it can make sure that no aggregates in the network are congested within the core of the network, avoiding case 3 from Figure 2.1. The routing system can then *maximize* the utilization of the network while at the same time requiring very small queues.

One straightforward heuristic, used by B4 [45], to do so is to start by allocating as much demand as possible on each aggregate's lowest propagation delay path. If there are one or more aggregates whose demand is not satisfied after saturating their respective shortest paths, the algorithm allocates demand to those aggregates' second shortest paths and so on until all aggregates are satisfied or there are no more paths. In B4 not all aggregates are treated equally—the allocation process described above can be influenced by operator-supplied per-aggregate weights so that aggregates that are more sensitive to delay are allocated faster to shorter paths.

Interestingly, B4's behavior is reminiscent of that of MPLS-TE—it senses each aggregate's demands and attempts to route aggregates on their respective shortest possible paths; aggregates that do not fit are routed on non-shortest paths. The main differences are that B4 is centralized and can automatically use uneven splitting among each aggregate's paths, whereas MPLS-TE either sends each aggregate's traffic down a single path or, if used in conjunction with ECMP, splits the aggregate's traffic evenly among a number of paths.

## 2.3 Solution Space

The previous sections described the most relevant related work in the field of routing and traffic engineering. In this section we will provide a rough mapping of the solution space, which will hopefully help readers see where our work fits in the research area.

There are two main sources of delay a packet can experience in a WAN environment—it can be delayed while traversing a long-haul fiber link, or it can be delayed because of other packets at the queue of a device's interface. Those two sources of delay are at odds—because of limited network capacity it is often the case that lowering queuing delay comes at the expense of increasing

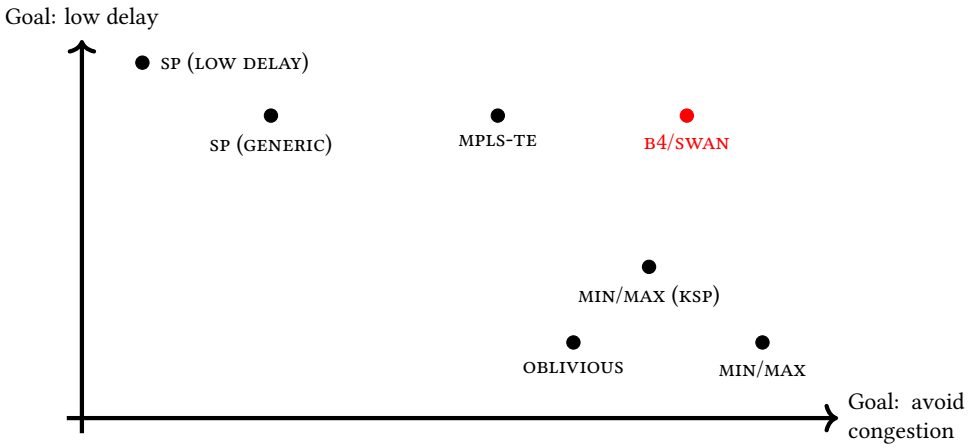



**Figure 2.8:** A map of the solution space of routing based on the routing system's objective; systems that directly control traffic sources in enterprise environments are colored red. The positions of the points are notional and do not correspond to specific quantitative values.

propagation delay. Therefore, one way to map the solution space is to examine how each solution deals with delay. Does it minimize propagation delay, or does it aim to achieve low queuing delay?

Figure 2.8 places some of the previously discussed solutions on a two-dimensional plot where on the $y$ axis we display how important achieving low propagation delay is for a system, and on the $x$ axis how concerned a system is with low queuing delay.

Shortest path routing with link weights set proportional to propagation delay, labeled SP (LOW DELAY), sits at one extreme of the design space. SP (LOW DELAY) always routes packets on the path with the lowest possible propagation delay, with no concern about congestion, and therefore queuing. By tweaking link weights shortest path routing can be influenced to take longer paths, in the interest of avoiding congestion or complying with policy. We label all such solutions SP (GENERIC). This category covers manual ad-hoc operator changes to link weights as well as automated solutions (*e.g.,* [31]). More recent traffic engineering solutions are capable of routing over non-shortest paths—MPLS-TE can handle cases where shortest path routing fails, regardless of weight assignment (*e.g.,* Figure 2.1), while still achieving comparatively low propagation delay.

Theoretically-optimal MinMax sits at the other extreme of the design space. MinMax, always spreads traffic as much as possible—an optimal solution has paths with high propagation delay, but the solution is as congestion-averse as possible. Path-based MinMax where each aggregate gets the $k$ shortest paths (labeled MIN/MAX KSP) achieves lower propagation delay than theoretically optimal MinMax as each aggregate is limited to a relatively small number of short paths, but due to the restricted set of paths MIN/MAX KSP can fail to fit the traffic matrix's demand in some cases in which MIN/MAX does. Oblivious routing (labeled OBLIVIOUS) has the same objective as MinMax, but is traffic matrix agnostic, and thus unable to reduce congestion to the level of MIN/MAX.

Recent, deployed routing systems such as SWAN or B4 (labeled B4) are based on centralized path optimization. These designs are tailored to enterprise WANs, where they can directly control

traffic sources. By knowing per-flow bandwidth a priori those systems can “pack” traffic safely on low-delay paths without worrying that variability may cause congestion.

Chapter 3

# The Challenges of Routing for Low Latency

As we stated in Chapter 1, topologies with high potential for low-latency routing are more connected and mesh-like. In this chapter we expound upon the nature of the delay-minimizing routing problem for mesh-like backbones.

Just how mesh-like *are* today's backbones—i.e., to what extent do they incorporate direct, latency-minimizing links? To shed some light on this question, we examine a set of date-stamped real-world POP-level backbone topologies from the Topology Zoo [50], spanning 1998 to 2012. We limit our study to backbones with more than 10 POPs, as it is at medium-to-large scale where cost pressures constrain a backbone's density of connectivity. For each backbone we compute $f = (L - N + 1)/L$—the fraction of its links whose removal would render the backbone a spanning tree—where $N$ is the number of nodes and $L$ the number of links. As a spanning tree contains the minimum number of links that render a set of nodes connected, this value intuitively represents the extent to which a topology incorporates links inessential for "bare" connectivity. The value of $f$ is 0 for a tree topology; a rectilinear 2D grid topology's value will be close to 0.5. Figure 3.1 shows a CDF of $f$ across backbones. Most of the backbones in this dataset are not mesh-like: half of them are rendered trees by removing fewer than 20% of their links. Moreover, it is likely that a significant fraction of the topologies in the dataset include virtual links, which exaggerate a topology's "meshiness." Few topologies approach a grid-like density of connectivity, and closer inspection reveals that there is no noticeable increasing trend in "meshiness" over the 14-year period spanned by this dataset.

## 3.1 The Bandwidth-Propagation Delay Tradeoff

Why aren't backbones becoming more mesh-like? One reason may be that routing over mesh-like backbones is hard. Any routing system that tries to minimize latency over a mesh-like topology must place as much traffic as possible on low-delay paths, and route the rest on higher-delay paths. Doing so requires maintaining a precarious balance between propagation and queuing delay—if the

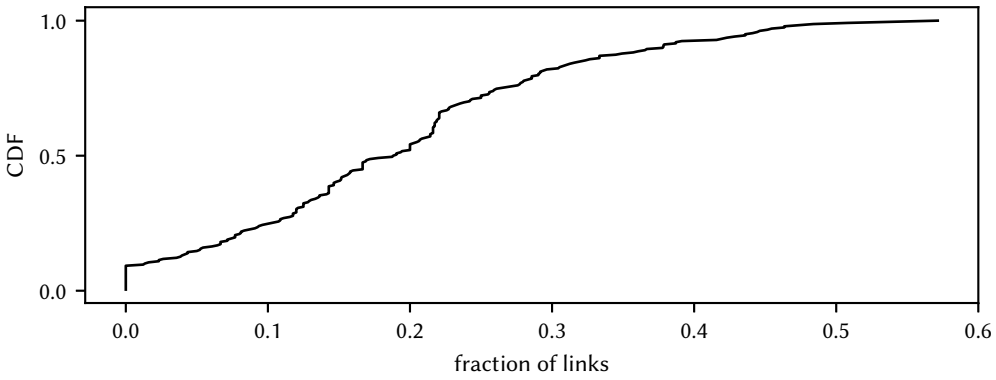


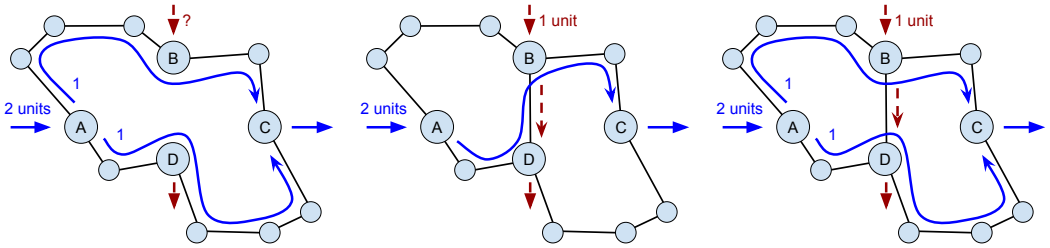

**Figure 3.1:** CDF of the fraction of links that, when removed from each topology, converts it into a spanning tree; 237 real-world topologies from the Topology Zoo [50] dataset.

system places more traffic on a path than any of that path's links can handle, congestion will occur, leading to queuing delays. Alternatively, if the system leaves a lot of spare capacity on low-delay paths, it will again increase latency, by causing traffic that could have been routed on a shorter-delay path to incur longer propagation delay.

This fundamental bandwidth-propagation delay trade-off manifests in surprising ways, even in simple scenarios. Consider SP/ECMP routing, MPLS-TE as described in Chapter 2, and state-of-the-art routing systems like B4. SP/ECMP routing ignores traffic demands, and places all traffic bound for a destination on the shortest path or paths. MPLS-TE takes account of demand: it places entire *aggregates*—each between one ingress and egress in the backbone—one by one. Once some links become full, further aggregates will be placed on the shortest path where there is still enough capacity. B4 will split aggregates where necessary, and greedily places traffic from aggregates on progressively longer paths. What these schemes share is that they *greedily place each aggregate's traffic on its shortest path first.* It is this common feature that leads to undesirable behavior for all these schemes in some scenarios. We refer to these systems as *greedy SP routing.*

Greedy SP routing systems differ in mechanism, but largely share the same objective. B4 is centralized; its central controller periodically assigns as much of each aggregate's traffic as possible to its respective shortest path, and then sends the rest on the next shortest path that still has free capacity, and so on. MPLS-TE is distributed, and each ingress router is responsible for aggregates that enter the network via that ingress. Periodically each aggregate is assigned by its ingress to the shortest path with enough free capacity to meet the aggregate's demand. The demand is then subtracted from the available capacity of the hops along the path, and new free capacities are propagated to other participating devices via the IGP.

Consider the ISP in Figure 3.2a. All links have unit capacity; arrows denote aggregates. Let us assume that the operator has set link weights either to all be equal to the same value or to be proportional to propagation delay in an effort to minimize latency. Regardless of which of SP




**(a)** Initial scenario.

**(b)** Minimal propagation delay; congestion at A.

**(c)** Longer propagation delay; no congestion.

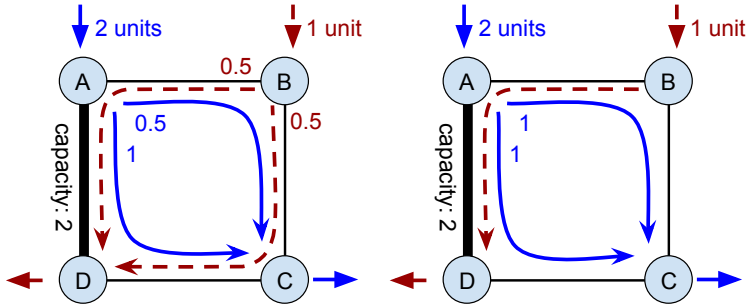
**Figure 3.2:** Simple scenario drawn to scale; all links have capacity of one unit. SP routing with ECMP, MPLS-TE, and B4 exhibit congestion when a new link is added whether cost is hop count or propagation delay; these routing schemes fail to fit offered demand because they *greedily* place each aggregate's flows onto their shortest paths first.

routing, MPLS-TE, or B4 one runs on the topology in Figure 3.2, one obtains exactly the same result.

Initially, as shown in Figure 3.2a, suppose there are flows from $A \to C$ totaling 2 units of demand, which the routing spreads among the two equal-cost paths $A \to B \to C$ and $A \to D \to C$. Now suppose further that the ISP adds a new customer at $B$ and, as a result, must carry 1 unit of traffic between $B$ and $D$. The operator must upgrade capacity in the backbone to carry this new demand, as the single aggregate from $A$ already fills links $A \to D$ and $B \to C$. Virtuously aiming to provide the lowest possible latency, the operator installs a direct link between $B$ and $D$ to carry the new traffic. To their surprise, as shown in Figure 3.2b, the new link causes congestion in seemingly unrelated part of the network—at $A$.

Why should *adding* capacity cause congestion? Because the new link's delay is low, its provisioning reduces the delay of the shortest path for the $A \to C$ aggregate. Any greedy SP-based routing scheme will thus dutifully place all of the $A \to C$ flows on the $A \to D \to B \to C$ path. Doing so saturates both links $A \to D$ and $B \to C$. The reverse path—$D \to B$—which carries the new customer's traffic is *also* saturated. In sum, under greedy SP routing, adding the new link *reduces* the capacity available to $A \to C$, which no longer fits and incurs drops at $A$. A different solution, though not within reach of greedy SP routing with delay-based link metrics, appears in Figure 3.2c. This placement of traffic avoids congestion by keeping only traffic for $B \to D$ on the direct link, and spreads $A \to C$ traffic evenly over the upper and lower paths, essentially ignoring the direct link.

Given the choice between routing flows over links with insufficient capacity, and hence increasing queuing delay (as in Figure 3.2b) and choosing longer-delay paths, and hence increasing propagation delay (as in Figure 3.2c), we posit that the routing system should avoid queuing if the topology as a whole allows doing so. While in prior work we outlined an attempt at trading off propagation delay and queuing delay [37], queuing delay is much less predictable than propagation delay: its magnitude depends on how deep the queues are at network devices. More importantly, it may worsen end-to-end delay for multiple aggregates that share a congested link.




**(a)** Greedy routing. **(b)** Congestion-free solution.

**Figure 3.3:** Greedy routing gets stuck in local minimum, fails to avoid congestion; all links have the same delay and the same unit capacity, except for $A \to D$, whose capacity is 2 units.

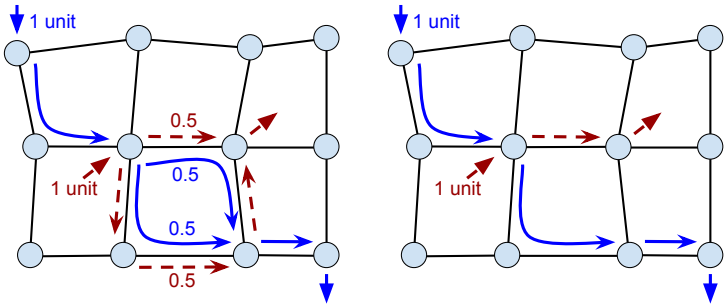
Obviously, if the routing system is to prioritize the avoidance of congestion when it places traffic, it must measure aggregates' demands—otherwise the routing system cannot proactively determine how much of an aggregate can safely be placed on a low-delay path before incurring queuing delays. In some enterprise WAN scenarios all end hosts are controlled by the same principal; in such cases that principal may simply cap aggregates' demands and report the caps to the routing system [45, 42]. In the more general ISP-like scenario, however, the routing system must carry traffic from end hosts whose traffic demands the ISP does not control. These demands further may exhibit high short-term variability. In Section 3.3.2 we discuss what levels of long and short term variability we expect to see in WAN traffic.

### 3.1.1 Greedy Routing and Varied Link Capacities

Another shortcoming of greedy SP routing schemes is that they can fail to avoid congestion even in very simple scenarios in the presence of varied link capacities. In the topology in Figure 3.3, all links have the same delay and the same unit capacity, except for $A \to D$, whose capacity is 2 units. In this case greedy SP routing starts by filling each aggregate's shortest paths evenly: $B \to C \to D$ and $B \to A \to D$ for the $B \to D$ aggregate and $A \to B \to C$ and $A \to D \to C$ for the $A \to C$ aggregate. It will assign 0.5 units of capacity to each one of these paths, at which point the $B \to C$ link will saturate. At this point the $B \to D$ aggregate's demand has fully been met, but there is no way for greedy routing to meet $A \to C$'s one more remaining unit of demand, as there are only 0.5 units of capacity left on the $D \to C$ link. This solution will result in persistent congestion at $A$. Figure 3.3b shows a solution that avoids congestion, though centralized greedy solutions such as B4 cannot find it. Distributed greedy solutions like MPLS-TE may or may not find it depending on when different devices perform path recomputation and reserve bandwidth.

### 3.1.2 Greedy Routing and Local Aggregates

An astute observer will notice that the main reason greedy routing fails to achieve the preferred, congestion-free outcomes in Figures 3.2 and 3.3 is that there is not enough path diversity for it to find alternative paths. Will making the network more mesh-like cure that problem?




**(a)** Greedy routing. **(b)** Lower-delay solution.

**Figure 3.4:** Greedy routing yields high delay on mesh-like networks; all links have a capacity of one unit and the two aggregates have a demand of one unit. The long-haul aggregate (shown in blue) has a second best path that is of slightly longer latency than its shortest path, while the second best path of the local aggregate (show in red) is significantly worse.

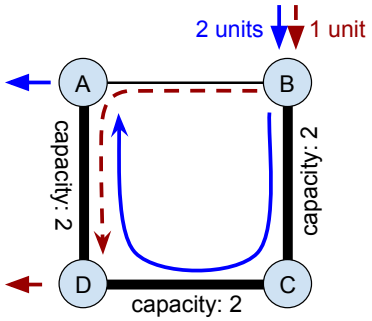



**Figure 3.5:** Congestion-free solution that is unattainable by SP routing regardless of assignment of weights; thick lines have a capacity of 2 units.

Alas, even in networks with great path diversity, greedy SP routing exhibits undesirable behavior. In Figure 3.4 we show a highly connected topology—a complete rectilinear grid. There are two aggregates—an aggregate that carries long-haul traffic from one end of the network to the other and an aggregate that is purely local whose shortest path is a single hop. As the local aggregate's shortest path falls along the shortest path of the long aggregate, a greedy SP routing solution (shown in Figure 3.4a) would saturate the link on the local aggregate's shortest path with traffic from both aggregates and then allocate the rest of both aggregates on their respective second-best paths.

Notice that while the second-best path of the long aggregate is only fractionally longer than its best path, half of the local aggregate's traffic will suffer a needlessly circuitous path. It is much better to route all of the long aggregate over its second-best path and route all of the local aggregate on its shortest path (as in Figure 3.4b). In essence, greedy SP routing tends to "punish" local aggregates in propagation delay.

### 3.1.3 The Need for Non-Greedy Routing

In Figure 3.2 it is possible for an experienced network operator to artificially increase the cost of one or more links, in the spirit of [31], in order to nudge even basic single-path SP routing into

finding the congestion-free solution from Figure 3.2c. By adopting weights that do not correspond to link delays, the operator effectively repurposes routing to achieve a different objective—to avoid congestion rather than minimize delay. This process requires intimate knowledge of the network's demands at any point in time, as even in simple scenarios it can be difficult to pick a weight assignment. For example there exists *no assignment* of link weights that will cause single-path SP routing to produce the desired outcome in Figure 3.5. To see why, note that this example is similar to the one from Figure 2.1 from Chapter 2. In this case it is impossible to force the single-unit aggregate over the $B \rightarrow A \rightarrow D$ path without also routing the two-unit aggregate over the $B \rightarrow A$ link and congesting it. B4 will be able to handle Figure 3.5; depending on the ordering of events MPLS-TE may also be able to handle it. However, those more advanced schemes fail to alleviate congestion and deliver low propagation delay in Figures 3.3 and 3.4 respectively.

## 3.2 Assessing Topologies' Potential for Low Latency

The examples presented in the previous section seem to indicate that today's routing and TE systems cannot place traffic onto a mesh-like backbone's multiple alternative paths so as to both satisfy user demands *and* minimize delay.

However, even though such examples are very helpful in understanding the behavior of current routing systems, one can argue that they are too small-scale and synthetic to bear relationship to what occurs in real-life networks today. Before we set off to design a new routing system that addresses the problems brought to light in the last chapter, it is worthwhile to further explore current routing systems using realistic traffic matrices on real-worlds topologies.

Since the performance of any routing system is intimately tied to the network topology, we will begin by understanding if, fundamentally, today's topologies are well suited for low-latency routing. Is it really the case that today's routing systems are not well suited to provide low latency, or is it the case that today's topologies are not diverse enough for the routing system to provide low latency?

If an operator wishes to build a network well-suited to providing robust low-delay communication, how would they measure the extent to which they had succeeded? One could say a topology offers low latency if the shortest paths between points of presence (POP) lie close to the corresponding great circle routes, but this falls short as a metric for two reasons:

- Geographic, geopolitical, and economic constraints limit where links can reasonably be provisioned.
- Shortest paths may end up congested if demand diverges from that envisaged during provisioning, leading to queuing delays and loss. Avoiding congestion without massive overprovisioning requires using alternate, longer paths.

We do not claim any deep insight into geopolitical or economic constraints that limit link deployment. For now, let us consider only network links that exist in real ISPs.

The "meshiness" metric used at the beginning of the previous chapter is very crude—it only measures how connected a network is and while it is clear that more connected networks are more prone to interesting routing behavior, the fact that a network is connected offers little insight into how useful this connectivity is in providing low-latency service. What we would really like is a network topology metric, agnostic to both routing and traffic, that characterizes how well suited the topology is to providing *robust low-latency* communications.

### 3.2.1 Low-Latency Path Diversity

Although the shortest paths in a network may not be ideal, they are the best paths we are sure are viable to provision. How well suited is a network topology to providing low-latency service under traffic loads that are not trivial to route—i.e., ones that do not fit on the shortest paths alone?

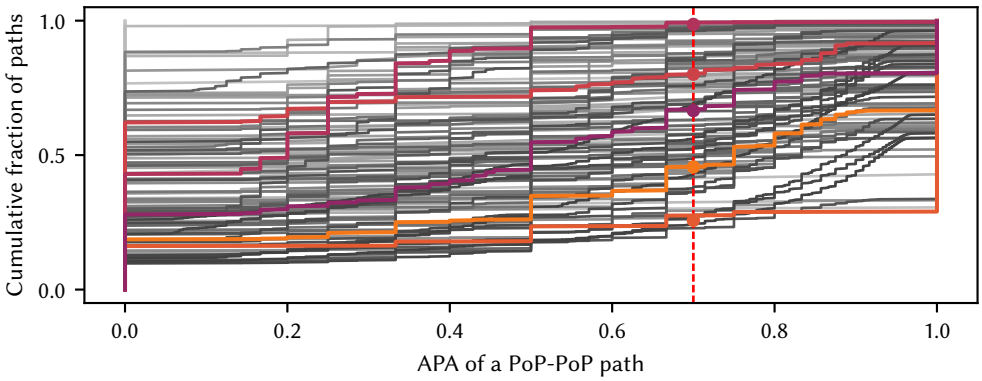



**Figure 3.6:** CDF curves of APA for all networks, given path stretch limit of 40%. Five random curves are highlighted. The vertical line at 0.7 indicates PoPs 70% of whose shortest-path links can be routed around without excessive delay.

To derive such a metric we start from a network map that includes all PoPs and link latencies. For each PoP pair, we compute the lowest latency path. Then for each link on the path, we consider the latency cost to route around that link if it were congested. If the map contains link capacities, we must also take these into account. For example, it is unreasonable to consider a 1 Gbps link as providing a viable alternate to a congested 100 Gbps path. We consider an alternate path as a *viable alternate* if its bottleneck has at least the capacity of the bottleneck on the shortest path. If there are multiple alternate paths, we progressively add the $n$ lowest latency alternate paths until their min-cut is sufficient to form a viable alternate. When this is necessary, we consider the propagation delay of the alternate to be that of the $n^{\text{th}}$ lowest latency alternate.

We define *path stretch* to be the fraction $\frac{d_a}{d_s}$, where the viable alternate path's propagation delay is $d_a$ and the delay of the shortest path between the two PoPs is $d_s$. We set a threshold for path stretch—for example, we may consider a path stretch of 1.4 to be acceptable—and measure alternate path availability (APA), defined as the fraction of links on the shortest path that can be routed around without exceeding this stretch limit. Each PoP pair gives an APA data point in the range from zero (no links can be routed around without excessive delay) to one (all links can be routed around). A CDF of those data points characterizes the network. The resulting curve gives insight into the availability of low-latency alternate paths, and is scale-invariant, so can be used to compare networks of different size and geographic scale. This curve captures the feasibility of routing around hotspots caused by congestion without dramatically inflating delay.

Figure 3.6 shows CDF curves of APA values with path stretch threshold of 1.4 for each of the 116 networks with diameter greater than 10 ms in the Topology Zoo (augmented with computed link latencies [35]). [1] The Topology Zoo is not without limitations; some topologies are rather old, and PoP locations are often unverified. Nevertheless, it gives a useful view of diverse WAN

[1]Since we are interested in low-latency routing it makes no sense to look at very local network where the latency is likely to always be low regardless of routing.

topologies over time; even older topologies elucidate then-current backbones' delay characteristics. In the dataset networks vary considerably in how well they provide low-latency alternate paths. Consider the $x$-axis value of 0.7; this indicates paths where 70% of links can be routed around without excessive delay. A corresponding $y$-axis value of 0.25 indicates that 75% of paths have low-latency alternates that route around at least 70% of the hops. Thus topologies whose curves are to the lower right on this graph provide usable path diversity.

A few curves are horizontal lines; these are clique topologies. We understand these to be overlay networks; for example, one is an older network provisioned using ATM virtual circuits. Overlays are not really interesting from our point of view: the ISP likely uses the overlay technology to provision on demand, rather than rely on intra-domain routing.

To reduce each curve to a single metric for each network, we compute *low latency path diversity* (LLPD) as follows.

$$LLPD = \frac{\text{number of POP pairs with APA} \geq 0.7}{\text{total number of POP pairs}}$$

The choice of 0.7 here is not crucial; as Figure 3.6 shows, the rank ordering does not change greatly for this set of topologies if we choose a different threshold in the upper half of the distribution.

An LLPD of close to one indicates that for most POP pairs, we can route around most of the links on their shortest path without incurring excessive delay. Conversely an LLPD of close to zero usually indicates a more tree-like network. Networks with LLPD in the middle of the range often consist of wide rings: while they have path diversity, for two nodes close together on a ring, the latency cost of going the "other way" around the ring is considerable.

Networks with high LLPD typically fall in two categories. Some are well interconnected, resembling a two-dimensional grid. An example of this class is GTS's network in central Europe, shown in Figure 3.7. Others, such as Cogent, span more than one continent, with good path diversity between continents. The long latency baseline between continents makes it easier for them to score well on latency stretch, but they also need to have good connectivity within continents.

### 3.2.2 Path Diversity is Hard to Use

In Section 3.1 we note using small synthetic examples that two-dimensional grid networks can be hard to route, as they inadvertently concentrate traffic. We use LLPD to understand if and to what extent this is a problem in real networks.

We analyze the topologies from the Topology Zoo. For each topology we synthesize 100 traffic matrices, each representing a moderate load for network's available capacity. To do so, we use a variant of the gravity model [68]. This model generates traffic aggregates between POP pairs according to a Zipf distribution, as real-world traffic has been characterized. The original model, however, is agnostic to traffic locality. To see how traffic locality affects routing, we add a locality

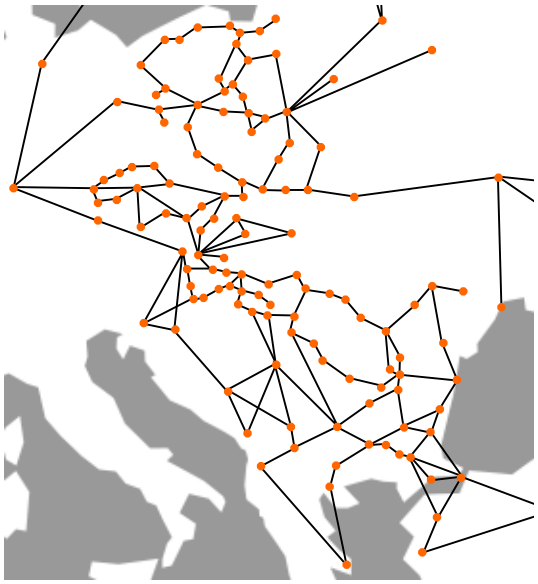


**Figure 3.7:** GTS's Central Europe topology

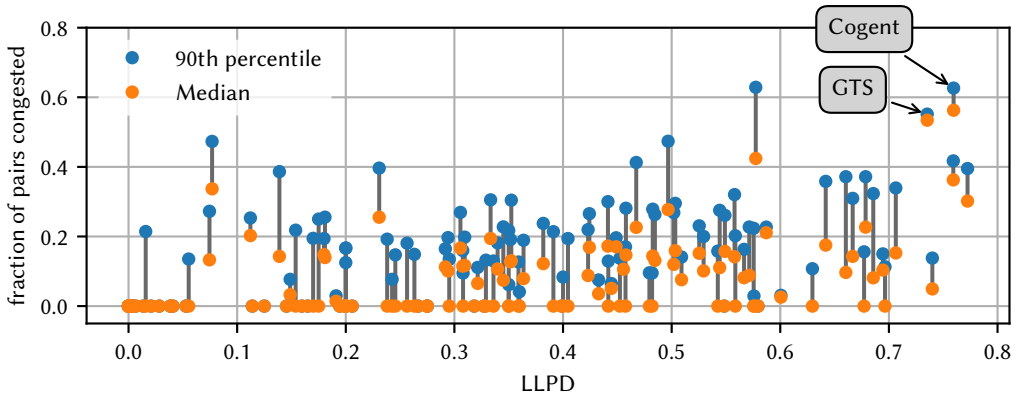



**Figure 3.8:** Networks with high LLPD tend to concentrate traffic when using SP routing. The $x$-axis shows the topologies sorted according to their LLPD, and the $y$-axis reports the fraction of source-destination pairs that experience congestion.

parameter to traffic matrix generation. For locality values greater than zero we redistribute some traffic from longer-distance flows to more local flows, increasing locality while maintaining the total traffic sourced and sunk by each POP. Specifically, a locality parameter of $l$ allows short-distance flows to increase by $l$ times their original demand. We find that a locality of 1 is enough to add significant locality, while larger values tend to under-load long-distance links too much to justify their presence in the topology. Unless stated otherwise, we use a locality value of 1 in our analyses. Section 7.2 contains a detailed explanation of how the traffic matrices used in this experiment are generated.

To ensure the network is moderately loaded but not close to being overloaded, we scale each traffic matrix so that the min-cut of the network has 24% headroom, if we minimize maximum

link utilization. In other words, all aggregates in the traffic matrix could be increased by 30%[2] and still be routed without saturating any link. In most topologies this corresponds to a median link utilization of 20-30%.

### Shortest path routing

We first wish to see how each network performs using delay-proportional shortest-path routing, in which the operators tunes link costs in a simple intra-domain routing protocol [59, 62] to minimize delay. Figure 3.8 shows the median and $90^{th}$ percentile of congested source-destination pairs across all topologies and traffic matrices, when networks are sorted by their LLPD value. The $x$-axis shows the topologies sorted according to their LLPD, and the $y$-axis reports the fraction of source-destination pairs that experience congestion. The figure shows that under moderate load shortest-path routing tends to concentrate traffic in networks with multiple low-latency paths (high LLPD).

One conclusion is that networks with good LLPD are not designed to be used with shortest-path routing. Such networks have many low-latency alternative paths, and it seems likely that they have evolved to be run with a traffic engineering scheme capable to use these alternative paths. To understand the interplay between topology and routing, we need to examine them using active load-dependent routing systems.

### Latency optimality

In Figure 3.9 we show the performance of active routing schemes. The top half of each graph is the same as in Figure 3.8: it shows the fraction of paths that are congested after the analyzed routing scheme has done its best to avoid congestion. The bottom half of each graph is inverted, and shows shows latency stretch, calculated as $\sum_f d_f / \sum_f d_{f,sp}$, where $d_f$ is the delay seen by flow $f$ when routed by the scheme, and $d_{f,sp}$ is the shortest path latency between that source and destination. A value of 1 indicates that all flows are on their shortest path.

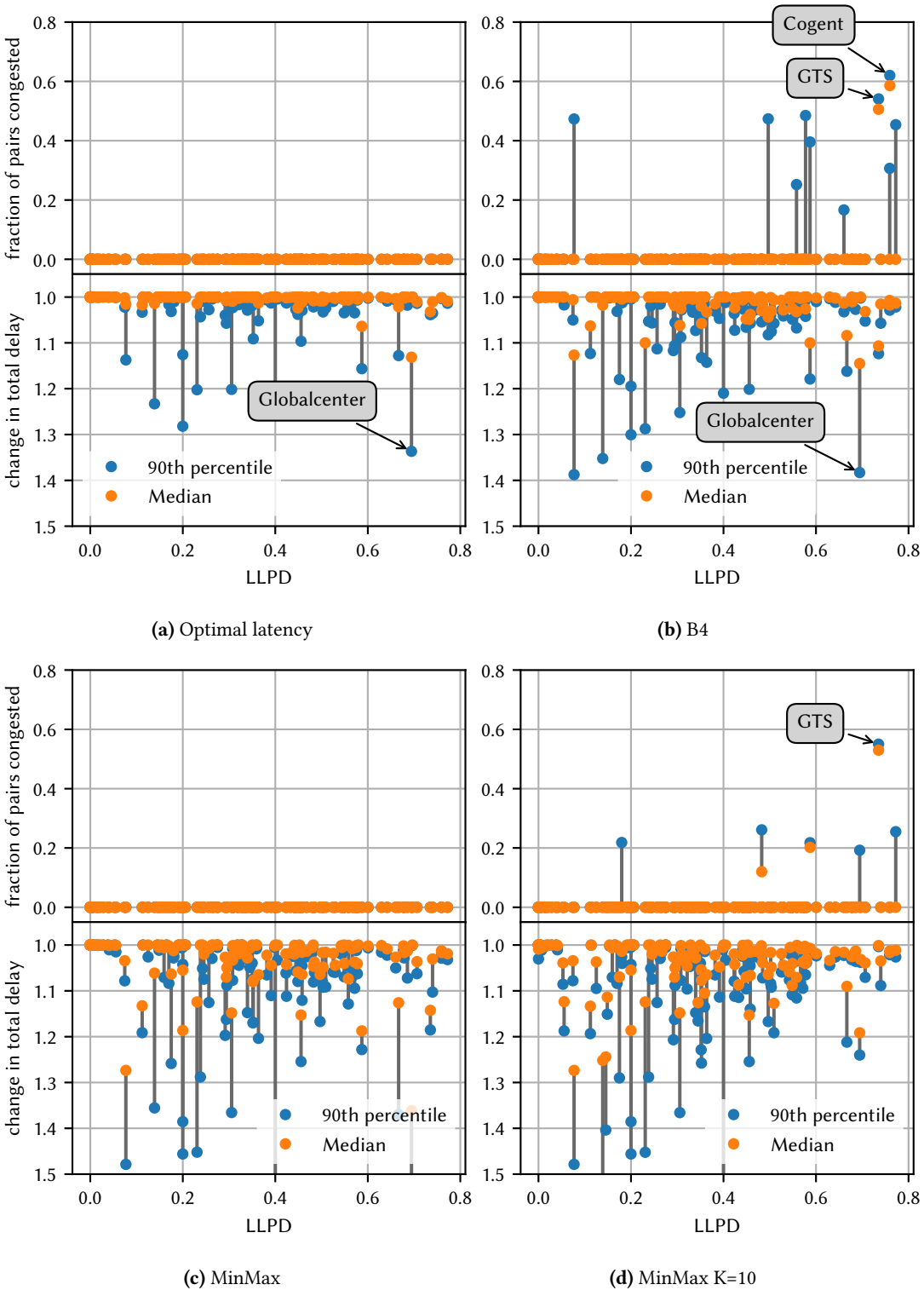
Figure 3.9a reports the results of an optimal routing scheme, where optimality is expressed as minimizing the sum of the propagation delays seen by all flows. Specifically, the optimal scheme minimizes

$$\sum_a n_a \sum_{p \in P_a} x_{ap} d_p \tag{3.1}$$

subject to the constraints that no link is overloaded and that all flows are routed. Here, $n_a$ is the number of flows in traffic aggregate $a$, $P_a$ are the paths $a$ might take, $d_p$ is the propagation delay of path $p$, and $x_{ap}$ is the fraction of traffic from $a$ placed on path $p$.

Figure 3.9a shows that it is possible to route all traffic, and to do so without causing excessive delay stretch. An exception, Globalcenter, is labeled; it is a full-mesh topology, so likely is an overlay network where it makes little sense performing dynamic routing at the IP level. Grid-like networks such as GTS and diverse intercontinental networks like Cogent that were prominent in

[2] Min cut load is 76%, so the traffic can increase by a factor of $\frac{1}{0.76} \approx 1.3$.




**(a)** Optimal latency

**(b)** B4

**(c)** MinMax

**(d)** MinMax K=10

**Figure 3.9:** Effects of active routing on congestion and delay. The top part of each graph shows the fraction of all non-zero demands in the traffic matrix that end up crossing at least one congested path, the bottom part shows latency stretch. For each $x$ value (different topology) we plot the median and 90th percentile from runs across a range of traffic matrices. The gray line indicates the span of the distribution.

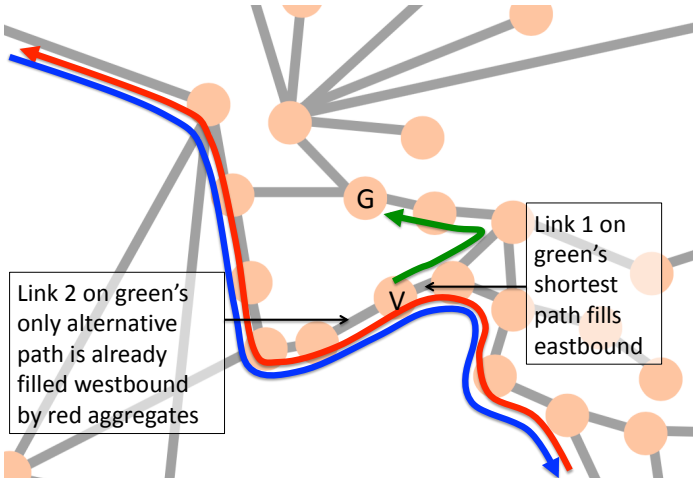



**Figure 3.10:** Inadvertent congestion on GTS using B4. The greedy nature of B4's path allocation causes both directions of the first link on the $V \rightarrow G$ path to quickly become saturated, at which point there are no alternative paths for the $V \rightarrow G$ traffic.

Figure 3.8 give low delay with this sort of optimal routing, which can make very effective use of their low-delay path diversity.

### Greedy low latency routing

How do deployed traffic engineering schemes perform? Automatic bandwidth allocation for MPLS-TE [78, 76] considers one aggregate at the time, and places each aggregate on its shortest non-congested path. As explained in Chapter 2, B4 uses a central controller to assign traffic from aggregates with a similar, slightly improved algorithm. [3] It starts by incrementally placing traffic from each aggregate onto its shortest path. This is done in parallel for all aggregates. When an aggregate's shortest path fills up, B4 starts allocating that aggregate onto the next shortest path, and so forth. Hence, while it considers low-latency paths first, B4 still uses a greedy algorithm. B4 [45] includes prioritization for subsets of traffic. In an ISP setting, it is less clear how to assign priorities than it is in Google's network. We give all traffic equal priority by default.

Figure 3.9b shows the performance of B4 on the topologies from the Topology Zoo, with the same parameters as in Figure 3.9a. B4 matches the optimal performance on many of the simpler networks. However, for most of the networks with mid-range LLPD, B4 gives slightly sub-optimal latency. Even more interestingly, it induces congestion on some of the networks with greatest path diversity: for GTS and Cogent, in particular, more than half of B4's paths cross a saturated link in the median case. Clearly, B4's greedy strategy frequently becomes locked into local minima in these topologies.

In Section 3.1 we showed similar effects on a synthetic topology susceptible to Braess's paradox [14]. We initially suspected that this was what was happening here, but in fact there are other more likely local minima that can trap B4. Consider the part of GTS's network shown in Fig-

[3] We focus on B4 but the same observations also hold for MPLS-TE.

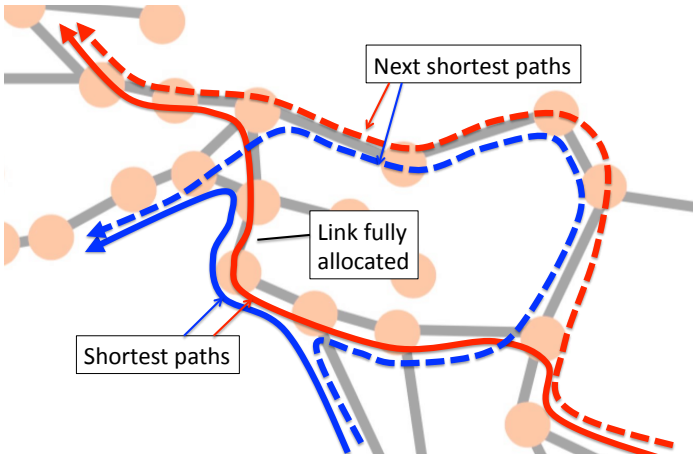



**Figure 3.11:** Excessive latency in the GTS topology using B4. As much as possible of each aggregate is routed on the shortest paths (the two solid lines) causing fully allocated links (like the one labeled) to be shared between the two aggregates; traffic from both aggregates is then sent on second-best paths. Note that even though the second best path of the red aggregate has comparable delay to its best path, the second best path of the blue aggregate is significantly longer—it would have been better to route more of the red aggregate on its second-best path. This real-world example is reminiscent of the synthetic one presented in Figure 3.4.

ure 3.10. This is a central part of this network, and a large number of aggregates flow through this region. Consider the aggregate from Veszprem (V) to Gyor (G). As B4 allocates traffic, link 1 fills up in the eastbound direction, occupied by the green and many blue aggregates. B4 would normally then start to allocate capacity on the second-best paths. For the blue aggregate of traffic flows, this is possible. However, if there are more red aggregates than blue ones, B4's algorithm will have already filled link 2 in the westbound direction with red traffic. There is no spare capacity for the green traffic as both link 1 eastbound and link 2 westbound are full, and these are the only links out of V. Of course, this example is a simplification of the real traffic allocation. In reality, the red and blue aggregates are hundreds of different aggregates, and other flows, not shown, are also present. However, the figure captures the basic cause and effect of B4's greedy choices.

The example shows that B4 cannot avoid congestion in this well-connected part of the network. In contrast, a closer-to-optimal placement would move red traffic aggregates onto the fractionally longer path through G, allowing room for the green traffic on link 2, and so avoiding congestion.

Even when B4 can fit the traffic, latency can be excessive. Consider Figure 3.11, where two aggregates share a link on their shortest paths. B4 will allocate the bottleneck link equally between the two aggregates until it is full, and then start filling the next-shortest paths for both aggregates. However, the next-shortest paths for the two shortest paths have different latency costs, with the blue aggregate needing to take a long detour. It would have been better to allow the blue aggregate to remain on its shortest path, and move more of the red aggregate to its second-best path, as there is minimal latency cost to the red aggregate from doing so.

### MinMax based routing

Other traffic engineering schemes such as TeXCP [47] and MATE [29] take the MinMax approach. A true MinMax approach would optimize traffic placement so as to minimize the maximum link utilization. As shown in Chapter 2, only minimizing maximum utilization is insufficient for a real system, as it would not generate unique solutions—many possible placements may have the same maximum link utilization, including ones with very suboptimal high-latency paths. One way to obtain a practical routing system is to minimize the sum of path latencies as tie-break between traffic placements with equal maximum link utilization.

Figure 3.9c shows the effectiveness of such a scheme. By definition, MinMax will fit the traffic if it is possible to do so: as expected, the figure shows that no aggregate experiences congestion. However, by focusing on utilization first and only using latency as a tie-break, many aggregates suffer significantly higher latency than they would with optimal routing (see Figure 3.9a). The reason is not complicated: to reduce maximum link utilization, some aggregates are forced over circuitous paths.

To prevent long paths from being selected unnecessarily, routing schemes such as TeXCP limit path choice to the $k$ shortest paths. The intuition is that if long paths are never given to the MinMax algorithm, a good balance will be struck between reducing latency and minimizing peak utilization.

In Figure 3.9d we show the results of running the MinMax algorithm using latency to tie-break, but supplying only the ten shortest paths, as suggested by TeXCP. For most networks with lower LLPD, there is little difference between full MinMax and MinMax with $k = 10$. These networks have little low-latency path diversity, hence some of the ten shortest paths are long. For networks with high LLPD, things are more interesting. Limiting path choice clearly does improve latency, though it is still worse than under B4. However, now the MinMax algorithm can no longer always avoid congestion. The main issue here is that networks with high LLPD have a very large number of possible, often non-disjoint, paths, so simply limiting choice to the $k$ best for a constant $k$ is insufficient to avoid congestion.

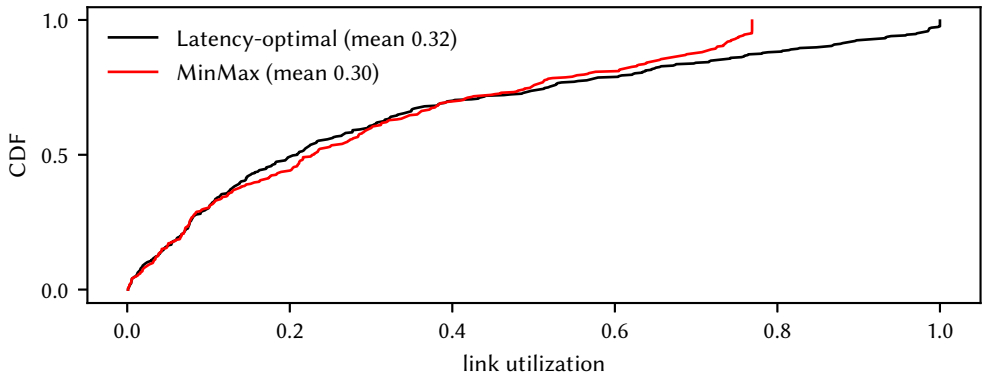



**Figure 3.12:** Link utilization in GTS.

## 3.3 The Headroom Dial

So far we have considered traffic as a fixed quantity that can be packed into a network. Real network traffic is neither constant rate not entirely predictable. A plausible option for a practical routing system is to reserve some minimum fraction of each link's capacity to accommodate foreseen but rare demand increases. We refer to this fraction as headroom.

Let us first examine how minimizing delay uses links' capacity. We consider again the GTS network, which has high LLPD.

Figure 3.12 shows CDFs of link utilization using the latency-optimal placement and our MinMax formulation, for one of GTS's traffic matrices from Figure 3.9. The median latency stretch on this topology in Figure 3.9 is 15% for MinMax and 4% for latency-optimal routing, but Figure 3.12 shows that most links are lightly loaded, and the difference in utilization of most links between the two schemes is not great. What matters is how loaded the most desirable links are.

Figure 3.12 also highlights that the few busiest links are loaded very close to 100% in the optimal routing scheme. No real network, however, would be deliberately operated with such extreme link utilizations, since traffic variability would cause (short-term) queuing which in turn would add delay. In practice some degree of headroom must be left on links.

We can regard this headroom as a dial that can be controlled by the routing system. We can calculate the latency-optimal path for a given value of headroom by simply scaling down link capacities according to the chosen headroom and running the optimal routing scheme on the modified topology. With headroom set to zero, we get the latency-optimal curve, but short-term queuing will adversely affect traffic. If we turn the headroom dial to the value calculated by MinMax as the maximal headroom possible on the busiest links (about 24% in Figure 3.12), then the latency-optimal algorithm converges with the MinMax algorithm, giving identical traffic placements. In between the two lies the viable range of traffic placements that all fit the traffic, but which trade off latency against headroom to accommodate traffic variability.[4]

[4]Figure 3.9 shows that B4 and MinMaxK10 sometimes lie outside this range.

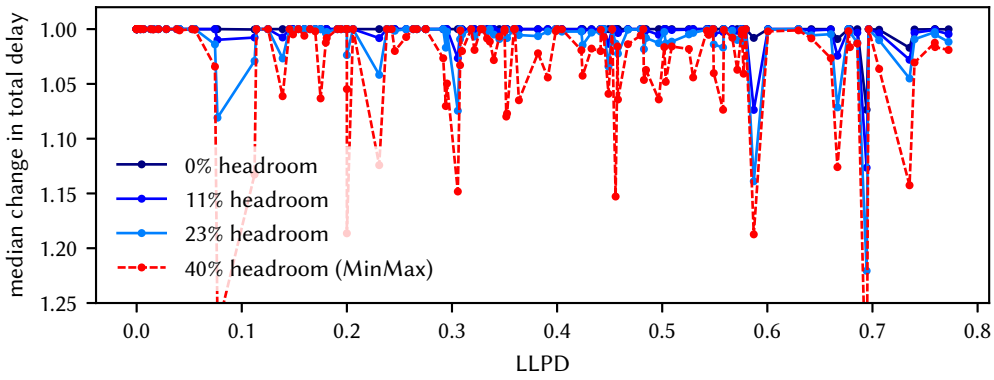



**Figure 3.13:** Latency stretch as headroom is increased.

Two key questions emerge from this view of headroom:

- How much headroom can be left before it starts to greatly impact latency?
- How much headroom is actually needed to allow statistical multiplexing on busy links without causing excessive short-term queuing?

### 3.3.1 Headroom vs. Latency

To see the effect of increased headroom on latency, consider Figure 3.13. This plot shows the median latency stretch as headroom is increased, when performing latency-optimal routing. To see the trend more clearly, we start with a slightly less loaded network - one in which the traffic matrix could be scaled by a factor of 1.65 before it is no longer possible to fit the traffic (i.e., the min-cut of the network is loaded at 60%). We then progressively increase the reserved headroom in steps from 0% reserved headroom to 40%. For this traffic level, with 40% headroom the latency-optimal placement converges with the MinMax placement.

The most prominent spikes with high LLPD are again from the clique networks; as noted before, these are less interesting because they are overlay networks, so have alternative ways to mitigate congestion. With the exception of these cliques, the other networks show relatively little delay stretch as headroom increases. This is the case even for networks with high LLPD. Only as headroom finally reaches the extreme of MinMax does delay stretch really increase greatly.

The implication is that it is probably unnecessary to live right on the ragged edge of triggering congestion to get paths with reasonably low latency. At the same time, minimizing headroom will normally decrease latency, so it is likely to be worthwhile actively estimating how much headroom is really needed to avoid significant transient queues building.

### 3.3.2 How Much Headroom is Needed?

Any load-dependent routing system must use estimates of traffic volumes to make its routing decisions. These estimates are inevitably imperfect. Suppose, for example, that the routing system

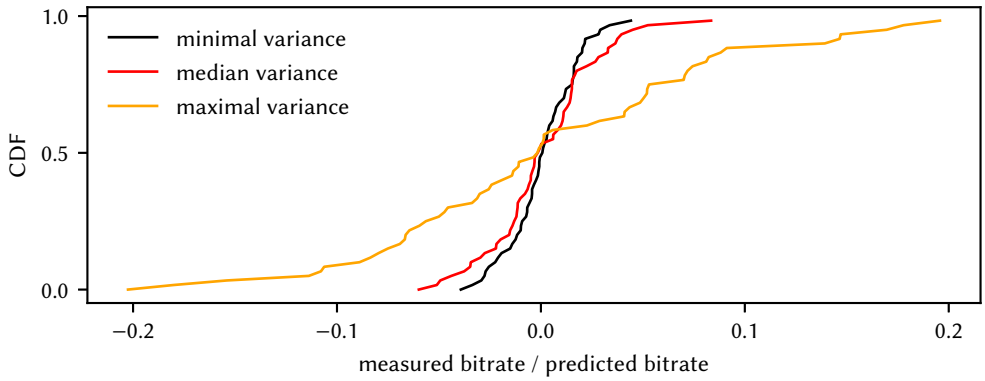



**Figure 3.14:** Minute to minute change of mean traffic level in the CAIDA dataset

recalculates routes every minute. Two factors need to be taken into account. First, how predictable is the mean traffic rate from minute to minute? Second, how well does short-term variability of traffic aggregates sharing each link statistically multiplex? If we can answer these questions, we can decide how much headroom needs to be allocated when calculating paths, so as to minimize latency due to propagation delay while also avoiding latency due to queuing.

## Predictable Mean Demand

As we noted in Chapter 1, commonplace overprovisioning by an ISP may leave aggregates' traffic demands bandwidth-constrained outside the ISP's backbone. The originating end-host application itself may in some cases limit a flow's bandwidth. Fixed-rate VoIP flows, for example, do not exceed a codec rate on the order of kilobits per second. Even for applications not so constrained, access links will often constitute a bottleneck, given that access links often are of far lower capacity than backbone links. Where applications burst, such an access-link bottleneck will smooth traffic peaks. Such smoothing by external bottlenecks should constrain the variability in demand over time, and thus render demand more predictable, *e.g.,* at a minute-to-minute granularity.

While measurements of traffic on a Tier-1 ISP's backbone links do not directly identify bottlenecks outside the ISP's backbone, they do allow an exploration of the variability (and thus predictability) of traffic demands on wide-area links, where predictability is consistent with the presence of such external bottlenecks.

We analyzed CAIDA packet traces spanning 2013-2016 from four 10 Gbps links within a U.S. Tier-1 ISP's backbone [15]. For each link we have 40 1-hour traces. Within each trace, we compared the mean traffic level each minute ($M_i$) with that from the previous minute ($M_{i-1}$). In line with our comments thus far on overprovisioning within ISPs' backbones, we note in passing that *no backbone link traced by CAIDA was congested.*

In Figure 3.14 we plot CDFs for the relative differences in mean traffic level during two consecutive minutes (*i.e.,* $\frac{M_i - M_{i-1}}{M_i}$) for three of these links. The line labeled *minimal variance* corresponds

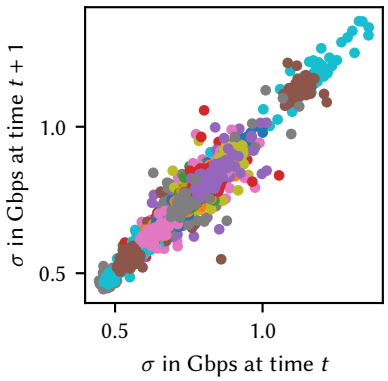



**Figure 3.15:** Minute to minute change of standard deviation.

to the one hour (out of all 40) where the set of relative differences exhibits the least variance—this trace is the most "predictable." Accordingly, the *maximal variance* line is the least predictable trace from minute to minute, and the *median variance* one represents what we expect traffic's minute-to-minute variation to be like most of the time.

Can a demand-sensitive routing system use measurements of past demand to determine the placement of future traffic, yet avoid placing future traffic in a way that causes congestion? This question ultimately is one of predictability of demand at the time granularity at which the routing system operates. These Tier-1 backbone link measurements suggest that on a minute-by-minute basis, traffic demands are usually fairly predictable. That is, they suggest that in most cases if one uses demand measurements from one minute to determine traffic placement for the next, it would not be unreasonable to reserve, say, 110% of observed demand from the prior minute, in the hope that 10% headroom will suffice to cope with any increase in demand in the next minute. One prior study reaches a similar conclusion: that demand is predictable over a minute-long bin size in the WAN, and is more predictable than demand on a LAN [66]. Furthermore, a more recent study of Google's WAN [41] measures a typical backbone link's utilization, which varies less than 10% from minute to minute.

While Figure 3.14 indicates 10% headroom is reasonable in the median case, the *maximal variance* curve suggests that for some periods a more sophisticated prediction algorithm may be needed, with up to 20% headroom required on some links. When several such aggregates are placed on the same 10 Gbps or 100 Gbps core link, it is very unlikely they will all exceed their predicted values simultaneously, so in many cases less headroom may be needed. There is a limit to what we can conclude from such traces though: although they do measure Tier-1 backbone traffic, we simply do not know if they are typical of other ISPs.

### Predictable Variability

On sub-second timescales we see greater variability. Is the variability of the traffic on short timescales sufficiently predictable? We measure the bitrate from the CAIDA traces each millisec-

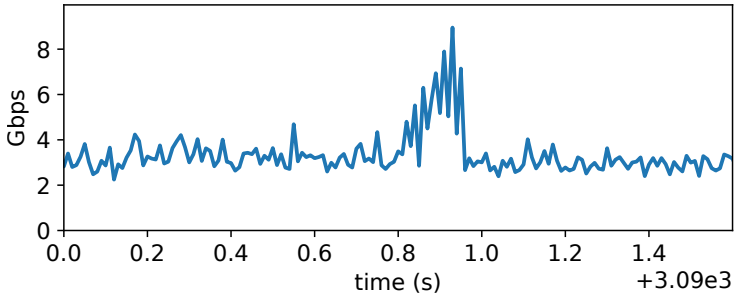



**Figure 3.16:** A short-term spike in traffic level; from 2016 CAIDA packet trace of uncongested U.S. Tier-1 ISP link.

ond, and calculate the standard deviation of these values for each minute. Figure 3.15 is a scatter plot of the standard deviation in minute $t$ plotted against the standard deviation in minute $t+1$. Different colors come from different traces, though some colors are reused. Although there is significant variability in the absolute value of standard deviation, the points are tightly clustered around the $x = y$ line, indicating that the variability of the traffic does not greatly change from one minute to the next.

### Sudden Changes

We note however, that we do see some exceptions to this predictability. As an example, Figure 3.16 shows traffic from a 2016 CAIDA trace averaged in 10-millisecond bins (instead of per minute). In this instance, background traffic, consisting of thousands of flows, consumes about 3 Gbps, but a single TCP flow that lasts for only 150 ms peaks at 5 Gbps. We conjecture these are datacenter-to-datacenter traffic, given the throughput. Second, we occasionally see a step change in both load *and* number of flows. Such cases almost certainly correspond to routing changes, though whether these are intra-domain changes or inter-domain changes we cannot tell. A dynamic routing system that meets the objectives above will not create unpredictable intra-domain changes, but it does still need to cope with unexpected changes in traffic level caused by BGP route changes.

### Long-Range Dependence

Previous studies (e.g., [54]) have discovered that LAN Ethernet traffic exhibits a degree of long range dependence (LRD). A formal introduction to LRD processes can be found at [20]. In brief, given a (weakly) stationary time series LRD refers to the property of the series to spend long stretches of time above or below the mean. This implies that the series has a long-term "memory" where a high (or low) level exhibits a strong effect on later levels. Later work [36] concludes that TCP's congestion control mechanism generates sustained correlations on its own and Internet traffic is inevitably LRD since it is mostly TCP.

LRD implies that simple Poisson models cannot be accurately used to model aggregated traffic. While we are not interested in modelling traffic, it may be that this unpredictability of LRD traffic negatively impacts the ability of the routing system to predict traffic levels. If this is the case, we

want to be able to measure the performance degradation.

Most of the experiments that we presented so far in this section (and some experiments later in this work) are conducted using a large set of packet traces from CAIDA. The traces are taken from a number of real-life WAN links in a production network, but we have no way of knowing how representative they are of what traffic looks like on other links or networks. It may be that the traces do not exhibit enough LRD to make the evaluation of our system representative of other Internet traffic.

Moreover, the bulk of the flows in the CAIDA traces are not bottlenecked close to the measured links—the speed of the flows' bottleneck links is a lot slower than the speed of the link where measurements are performed. This WAN scenario is at odds with the traditional measurement setup from [54] where traffic measurements are taken on the same LAN. Perhaps WAN traffic exhibits a different range of LRD behavior. Before we continue, we therefore need to answer the following questions:

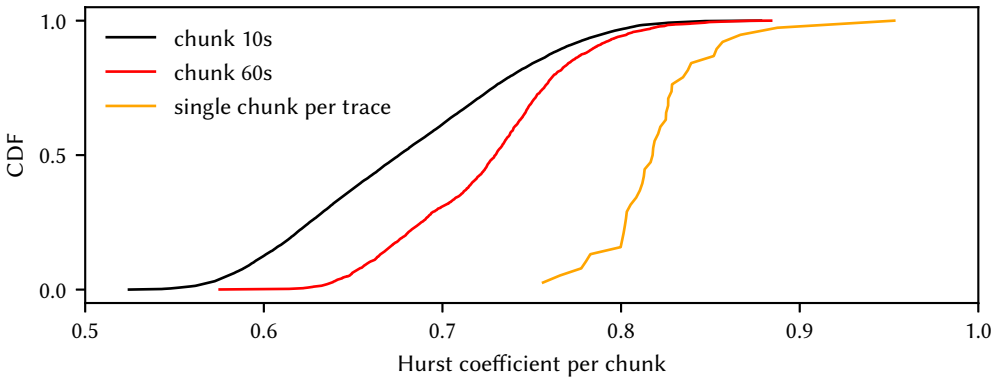
- Are the traces that we use representative of Internet traffic's LRD?
- Does the WAN traffic in the traces exhibit the same LRD effects as LAN traffic?

To measure LRD we use the Hurst parameter [44], which is between 0.5 and 1 for a process that exhibits LRD. To measure the Hurst parameter we use rescaled range (R/S) analysis, which is one of the methods employed by [54]. In that work they recorded the number of bytes crossing their monitor every 10ms.

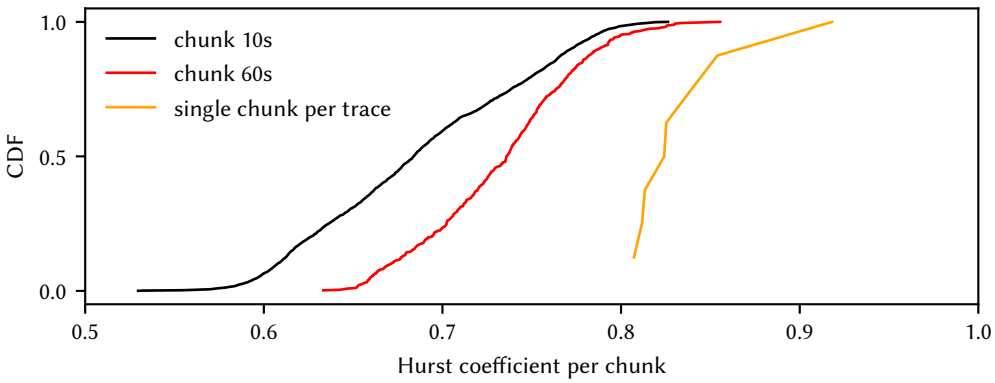
To keep as close as possible to their setup we bin each of the hour-long traces into 10ms bins, and combine multiple bins into chunks of given size. We treat the traffic level (number of bytes transmitted) of bins within each chunk as an independent time series and we evaluate its Hurst parameter. We repeat this experiment at various chunk sizes.

The results are shown in Figure 3.17a. We plot CDFs of the per-chunk Hurst parameters across all of the traces. The results suggest that the CAIDA traces exhibit LRD consistent with that of previously measured LAN traffic. Virtually all chunks across all traces exhibit some degree of long-range dependence (the Hurst parameter is larger than 0.5). The sigmoidal shape of each curve is due to the fact that the overall distribution of Hurst coefficients is normal, which is consistent with previous observations of LRD phenomena (not only network traffic).

Interestingly, as the chunk size gets larger we observe more long range dependence, as noticed by [36]. As the authors warn, this dependence should be taken with a grain of salt. At large timescales the process is no longer stationary and LRD is not distinguishable from long-range trends that change the mean of the data. How large a timescale needs to be for this to occur depends on the nature of the traffic in the trace itself. As previously discussed in this section, for the CAIDA traces we see mean traffic level remaining more or less (within 10%) stationary minute-to-minute, and we conjecture that the "true" level of LRD for traffic in the CAIDA traces is close to




**(a)** 10ms bin size




**(b)** 10ms bin size; multiple traces combined

**Figure 3.17:** Hurst parameters at different chunk sizes

the curve labelled "chunk 60s" on Figure 3.17a.

Each of the traces is taken from a 10 Gbps link, and has a mean rate of 3-4 Gbps. Nowadays a lot of busy core WAN links are 40 Gbps or even 100 Gbps [80]. How does aggregation affect the LRD that we observe? To answer this question we repeated the experiment from Figure 3.17a, but this time we did not perform our analysis on single traces, but we aggregated ten traces at a time. This results in 6 "super" one-hour traces out of our original smaller 60 one-hour traces. Each of our super traces has a mean rate of about 30 Gbps, representative of what traffic would be on a busy core link. The results are in Figure 3.17b.

The results are similar to the ones from Figure 3.17a which suggests that aggregating traces retains their LRD. The fact that LRD is present at different levels of aggregation implies that traffic in the traces exhibits self-similarity. If this is indeed the case we would be able to see roughly the same level of LRD when we change the bin size. To verify this we repeated the experiment from Figure 3.17, but with a bin size of 100ms. The results are in Figure 3.18. When each bin is 100ms we would only get 100 points over a 10 second interval, which is not enough to reliably estimate the Hurst coefficient using R/S analysis. This is why we omit the "10s chunk" curve from that figure.

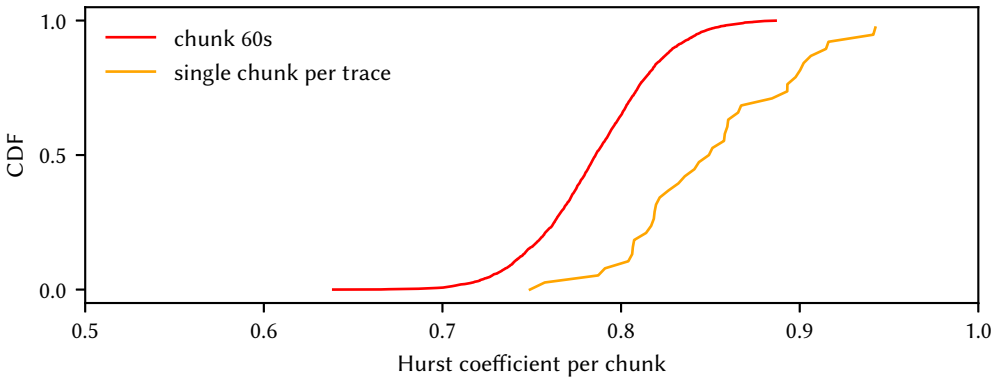



**Figure 3.18:** Hurst parameters at 100ms bin size

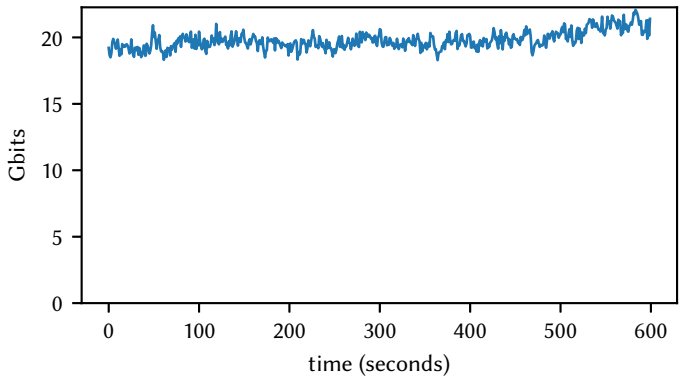



**Figure 3.19:** A minute from a combined hour-long trace with H = 0.8

Even though we scale the bin size by an order of magnitude the experiment shows that large levels of LRD are still present—traffic from the traces is indeed self-similar.

Note that self-similarity does not necessarily mean that traffic levels are unpredictable—the Hurst parameter is independent of the mean. If we added a very high value to all bins it would not affect the Hurst parameter values, but it would definitely render the traffic a lot more predictable, as we would be able to say with certainty that each bin's level is close to the very high value that we scaled the mean with.

To illustrate this point we examine on Figure 3.19 a minute from one of the combined traces. On the y axis we plot the number of gigabits transmitted each second, on the x axis we display seconds. The trace plotted has a Hurst coefficient of 0.8. This implies that it exhibits strong LRD. Despite that the range (difference between maximum and minimum level of bins) is very small compared to the high mean. Because of self-similarity and LRD the actual small scale variations at the top of the plot are probably hard to model, but we can certainly say that overall it is very unlikely that the traffic level of a bin exceeds 22 Gbps.

## 3.4 Summary of Findings

The findings presented in this chapter lead us to conclude that today's routing and TE systems indeed often cannot place traffic onto a mesh-like backbone's multiple alternative paths so as to both satisfy user demands *and* minimize delay. There is no dearth of low-level mechanisms for flexible *forwarding*: e.g., SP routing can be combined with virtual routing tables (VRFs) [69] to correctly handle the scenario in Figure 3.5; and SDN-based forwarding [55] can unevenly split traffic belonging to the same aggregate over any arbitrary path through the backbone. What is lacking is a dynamic routing system that *chooses paths and how aggregates should be split among them.* One challenge in building such a system is that it needs to both be able to react at a sub-second timescale, and produce an optimal or close-to-optimal traffic placement for thousands of aggregates. If a heuristic is used, it needs to be non-greedy and more sophisticated than the ones used by current state-of-the-art routing schemes.

Perhaps even more importantly, such a system would have to be able to measure and deal with each aggregate's inherent variability. In spite of the likely presence of LRD and the sudden changes shown in Figure 3.16, both long-term (Figure 3.14) and short-term (Figure 3.15) variability are readily predictable in the CAIDA traces, which we believe are representative of the type of traffic commonly seen on WAN links. Given these network conditions, it should be possible to construct a system that both achieves low latency by minimizing propagation delay, while at the same time avoiding queuing delay. Such a system would need between 10-15% headroom on most links, to cope with long-term variability in aggregate level; such headroom should be enough to mostly allow short-term variability of traffic aggregates sharing each link to statistically multiplex, but there may need to be an additional mechanism to detect and react to unexpected spikes in load as in Figure 3.16.

## Chapter 4

# Routing Goals and Design Overview

Given the inability of existing routing schemes to leverage topologies' potential for low latency traffic delivery, the obvious question is whether it is possible to design a practical routing system that both computes low-delay paths and automatically fine-tunes the headroom dial.

As we note in Chapter 1, ISPs eschew congestion in their backbones, typically by overprovisioning. This state of affairs often leaves aggregates' bandwidth demands constrained outside an ISP's backbone. Moreover, aggregates' traffic demands within an ISP's backbone are predominantly predictable from minute to minute. We offered evidence for this claim in Section 3.3.2, where we explored the variability (and thus predictability) of traffic demands on wide-area links in a real-world Tier-1 ISP's backbone.

Where demands are predictable and an ISP's backbone offers sufficient capacity to carry them, it should be possible for the ISP's routing system to minimize *both* queuing and propagation delays experienced by flows traversing that backbone. An ideal delay-minimizing routing system would therefore achieve the following high-level goals:

1. Route flows over paths that minimize the sum of propagation delays.

2. Minimize queuing delay: do not congest any link within the network when aggregates are stat-muxed.

3. When 1 and 2 cannot both be satisfied, prioritize avoiding queuing delay. Do so by rerouting as few flows as possible on paths of longer propagation delays.

4. When multiple solutions exist that minimize the sum of propagation delays, prefer the one that keeps local traffic local; keep the added delay experienced by an aggregate commensurate with its shortest-path delay.

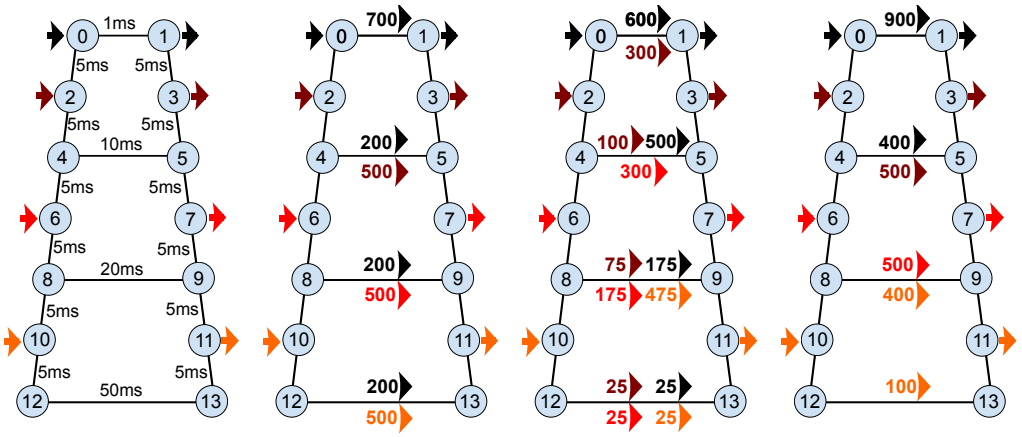
5. Avoid congestion when traffic demands change.

6. Adapt at a coarse enough timescale so as not to interfere with congestion control.

The routing system should also work in the (less common) case where an aggregate's bottleneck lies within the ISP's backbone. In this regime, the computed paths cannot minimize delay, as queues will inevitably build up in routers. They should, however, spread the traffic across diverse paths to the extent possible in order to maximize aggregate throughput. In other words in the case where congestion cannot be avoided the routing system should behave like the ideal MinMax scheme from Chapter 2.

In this work, we focus on how to compute and install paths for aggregates between each pair of ingress and egress routers (i.e., intra-domain routing). This is orthogonal and complementary to fine tuning the selection of ingress and egress routers through which aggregates should enter and leave (e.g., by tweaking the BGP configuration or overriding BGP decisions [81, 70]). We further discuss what it would take to deliver end-to-end latency across multiple domains in Chapter 8.

## 4.1 Requirements




**(a)** Ladder topology **(b)** MinMax **(c)** B4 **(d)** Ideal

**Figure 4.1:** MinMax and B4 do not explicitly target low delay, and so do not achieve minimal-delay paths. All links are of 1 Gbps capacity.

The *ladder* topology and aggregates shown in Figure 4.1a will serve as a simple example which will help us derive the requirements for an ideal delay-minimizing routing system by contrasting its desired behavior with the behavior of existing schemes.

As described in Chapter 2, TeXCP [47] and MATE [29] follow a MinMax approach: in an effort to avoid congestion, they minimize the maximum link utilization of any link in the network. B4 [45], by contrast, aims to maximize utilization by greedily placing load on shortest paths first, adding load to progressively longer paths as needed to fit all traffic.[1]

In Figure 4.1a, the arrows represent four aggregates, each with the ingresses and egresses shown. All aggregates' bottlenecks are outside the ISP's network. The uppermost aggregate's demand is 1.3 Gbps; the three lowermost aggregates each demand 500 Mbps. All links are of 1 Gbps

[1] In this work, we consider only B4's traffic placement algorithm, and not other aspects such as traffic prioritization, *etc.*.

capacity. The propagation delays on the vertical links at the left and right edges of the topology are all 5 ms, whereas the propagation delays on the horizontal links crossing the center of the topology increase progressively from top to bottom. Thus, intuitively, the path that offers each aggregate the lowest propagation delay is that which includes the horizontal link immediately above its ingress.

### 4.1.1 Requirement: Explicitly Target Low Delay

Since it focuses on congestion avoidance, MinMax is quite willing to assign traffic to paths that suffer from long delays. The delay-minimal traffic assignment that MinMax can compute on the ladder topology is shown in Figure 4.1b: a significant fraction of $0 \rightarrow 1$'s flows are routed over the bottom link, even though there is free capacity on the top link, which offers far lower delay. Note that by default MinMax treats any traffic-to-path assignment that minimizes the maximum link utilization as equally desirable. Thus single aggregates may be diverted over even longer-delay paths than just described: MinMax could for example route none of $0 \rightarrow 1$'s flow on the $0 \rightarrow 1$ link, as long as 700 Mbps are forwarded on every horizontal link. As explained in Section 3.2.2, when two traffic placements have equal maximum link utilization, we break ties in favor of the placement with lower sum of path latencies—the version of MinMax we compare against will always yield the solution shown in Figure 4.1b given the setup in Figure 4.1a.

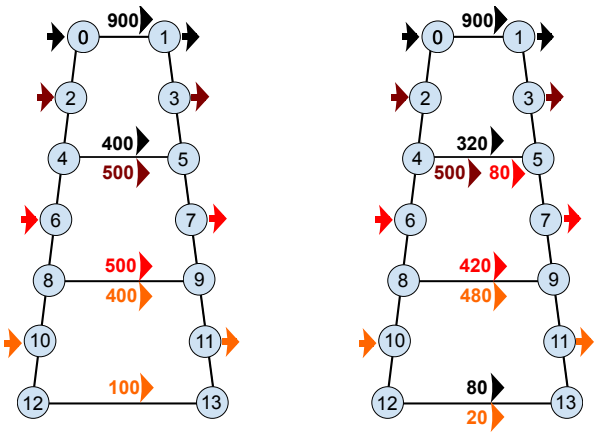
B4's greedy lowest-latency-first strategy offers users lower-latency paths than MinMax. However, B4 does not always arrive at a latency-minimizing traffic placement. Consider B4's result for the ladder topology, shown in Figure 4.1c. After filling each of the aggregates' best paths to, say, 90% utilization, B4 is forced to route along circuitous paths (e.g., some of the top aggregate is routed over the lowermost path). Flows in circuitously routed aggregates unnecessarily suffer the latency of the 5 ms side links—this is the same effect as observed on GTS's topology in Figure 3.11. Perhaps worse, the latency a flow in the $0 \rightarrow 1$ aggregate experiences is very unpredictable, depending on which paths flows' 5-tuples hash to.

### 4.1.2 Requirement: Adapt to Variable Demand

As we showed in Section 3.3, a delay-optimizing routing system will often run links on low-delay paths at very high utilization. In Figure 4.1c, for example, B4 fills 90% of the $0 \rightarrow 1$ link (as opposed to 70% in the case of MinMax). To avoid congestion on links so close to capacity, an ISP cannot re-optimize paths every 5-10 minutes, as systems that limit source traffic rates (like B4 and SWAN) can. Note that MinMax-based systems like TeXCP do not tend to drive links to high utilization, and so are inherently robust to small fluctuations in traffic volumes. When targeting low delay, however, even tiny traffic fluctuations risk congesting highly-utilized links. At the same time, a delay-minimizing routing system must leave enough headroom on links to avoid frequent routing changes upon the slightest changes in demand, which would violate goal 6.

### 4.1.3 Target Behavior

So how would a routing system that complies with our goals behave? Clearly we desire a behavior close to the ideal one presented in Section 3.2.2. How would that look on the ladder topology? Figure 4.1d shows the ideal assignment of traffic to paths that minimizes the sum of per-flow propagation delays without creating congestion (satisfying goals 1 and 2). While easy to express, this objective does not always yield a unique solution.




**(a)** Ideal **(b)** Excessive stretch

**Figure 4.2:** The solution to the left is the same as the ideal one in Figure 4.1d. The solution to the right has the same total propagation delay as the one on the left, but is less desirable due to excessive stretch.

To comply with goal 4, when multiple solutions exist that minimize total delay, an ideal routing system should prefer the one that further minimizes the sum of per-flow delay stretch—i.e., the ratio between the delays of the paths on which a flow is placed and the delay of the shortest path for that flow. Consider the two solutions in Figure 4.2. Both offer the same total sum delay, but Figure 4.2b's is not as desirable. Some of the top aggregate's flows have been rerouted over the very long bottom path—those flows have been "sacrificed" to make room to allow aggregate $6 \rightarrow 7$'s flows to take their shortest path. This choice does not affect total delay, but placing flows from the same aggregate on paths with very different delays is not desirable, as it worsens packet reordering upon routing changes.

Finally, to comply with goals 5 and 6, we envision that an ideal routing system periodically would re-optimize paths on a one-minute timescale. A minute is long enough to propagate a new routing solution to all devices in the network [45]. It is also long enough for measurements of throughput to be stable, but short enough that the underlying traffic demand has not had time to change greatly, as we showed in Figure 3.14. Every minute, such a system would collect data about demands over the past minute, predict traffic levels during the next minute, compute a new routing solution optimal with respect to this prediction, and install new paths (if any). To minimize the delay before optimal paths are used, all these steps must complete quickly, ideally within several seconds.

### 4.1.4 Greedy Heuristic or Closer-to-Optimal Solution

Given the stringent sub-second time requirement of a delay-minimizing routing system, it is natural to ask whether we truly need an optimal routing solution or we simply require a good one. While B4 falls short of achieving the best latency-minimizing traffic placement, its heuristic runs quickly. Moreover, looking back at the schemes in Figure 4.1 it is clear that B4's solution negatively impacts only a handful of flows. Indeed, in Section 7.1 we will use packet-level simulation to demonstrate that in the scenario from Figure 4.1 B4 performs significantly better than MinMax, and is close to an optimal solution.

While we will clearly need to address variability, as B4 is meant to be used in a controlled environment, do we also need a different traffic placement algorithm? Instead of a designing a slower, more complicated algorithm can we not simply use a greedy B4-like heuristic to quickly compute path placement, and extend it to cope with variability?

In Section 3.2.2 we demonstrated that this is not the case. Not only do greedy heuristics tend to produce traffic assignments which are at odds with low propagation delay, but such schemes are likely to get trapped in local minima in mesh-like topologies, inherently limiting the operator's ability to upgrade the network. Even when generating the optimal solution is infeasible, we should aim to use a non-greedy heuristic that avoids getting caught in local minima.

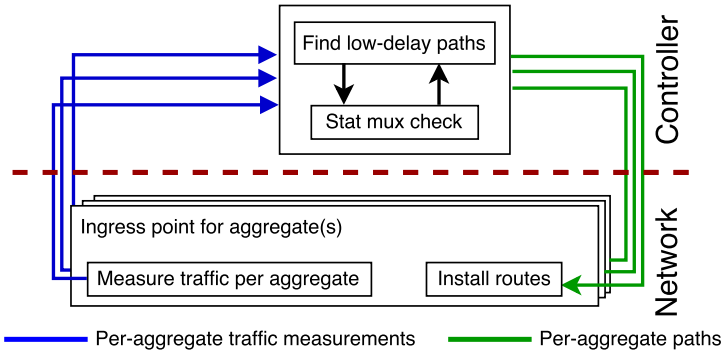



**Figure 4.3:** High-level operation: controller and ingress points.

## 4.2 LDR's Design

In this section we will overview the design of a system, called Low Delay Routing (LDR) that accomplishes the goals set forth at the beginning of this chapter.

LDR's architecture (Figure 4.3) incorporates functionality at each ingress router and in a central controller. The controller participates in a link-state routing protocol (as in [73]), through which it knows the network topology. The primary role of an ingress router is to feed measurements of traffic volumes (i.e., *demands*) to the controller. Armed with global knowledge of aggregates' demands and of the network topology, the controller centrally solves an optimization problem whose solution specifies one or more paths for each aggregate. After optimization, ingress routers direct traffic along label-switched paths (e.g., MPLS [23] ones) dictated by the controller.

The LDR controller casts the problem of choosing non-congesting, low-latency paths as a multi-commodity flow problem in which each aggregate is a distinct commodity of quantity equal to that aggregate's notionally fixed demand. While others have previously explored casting various routing objectives as optimization problems [13], doing so naively for our particular delay-minimization objective in a network of hundreds of links carrying thousands of aggregates yields too big an optimization problem to solve on the several-seconds timescale that meets our overall responsiveness objective. In Chapter 5, we contribute an efficient *iterated* formulation of this optimization problem that meets our delay-minimization objective on a several-seconds timescale.

In reality, however, an aggregate's demand will not be fixed: its mean minute-to-minute volume may be predictable, but on shorter timescales its volume may vary randomly. Routing systems that either directly control or closely profile sources' traffic, such as B4 and SWAN, don't face this problem: they are omniscient with respect to the exact traffic volume they expect from each traffic aggregate. In contrast we wish to target the uncontrolled environment of an ISP's or enterprise's backbone, where sources' precise demands are unknown. To cope with this uncertainty in demand, we describe in Chapter 6 a method for taking time series of aggregates' variability and combining them to predict how aggregates will share a link. When LDR considers placing aggregates on a link it uses this method to determine if they will multiplex without overloading the link. As shown in

Figure 4.3, LDR's controller alternates between placing aggregates and considering how they will statistically multiplex, revisiting placement with increased per-aggregate headroom to accommodate short-term variability as necessary.

The controller is also responsible for detecting when the traffic has changed rapidly in a way that may cause congestion. In that situation the controller reacts by immediately allocating more capacity to the offending aggregate(s). Whenever the link-state routing protocol reports a link or router failure, the controller runs a similar re-optimization, re-routing aggregates impacted by the failure. We further explore failures in Section 8.3.

We begin our exploration of LDR's detailed design by describing how the controller distributes network state.

## 4.3 Installing Network State

As mentioned above, all network state is centrally-controlled and the controller needs to instantiate a new label-switched tunnel for each new path. There are two main approaches to doing so—given a path of $N$ hops the controller can either install this path's state synchronously or asynchronously.

In the first case the complete path is sent to the egress and then state for the path is installed hop by hop on the reverse of the data path. This path instantiation process is very similar to the way reservation messages are sent in the second phase of the operation of RSVP. When the ingress receives the path instantiation message it can immediately start forwarding packets down the new path, as all nodes downstream will have already installed the appropriate routing state. The delay in installing the new path is proportional to the propagation delay of the path being installed plus the time for the initial message to reach the egress from the controller. There are $N$ messages involved—one generated by the controller and one for each $N-1$ hops downstream of the ingress.

The alternative is for the controller to directly communicate the routing state with each node along the path. It can asynchronously send updates to all $N-1$ nodes downstream of the ingress, and later when all those have been acknowledged, send a final update to the ingress, which will "enable" the newly installed path. The delay in installing the network state is equal to the maximum among the round trip times between the $N-1$ downstream devices and the controller plus the propagation delay from the controller to the ingress. Similarly to the synchronous approach, the asynchronous one involves $N$ messages, but they all need to be sent from (and acknowledged to) the controller. This concentration of controller traffic may render the asynchronous approach to installing network state unfeasible in large deployments where lots of paths may need to be installed at the same time. On the positive side, the asynchronous approach is more deployable, as it does not require a new RSVP-like protocol to instantiate network state.

In our simulation-based implementation of LDR we take the asynchronous approach—in all packet-level simulation results presented in this work when paths need to be installed the controller issues $N-1$ messages in parallel, followed by a single message to the ingress once the controller

has seen acknowledgments for all $N-1$ initial messages. We believe that for the POP-level network sizes that this work is focusing on (up to a couple of hundreds of nodes) the message overhead on the controller is not a concern.

## Chapter 5

# Optimization

Given that we gather data from ingress routers about the rates of traffic aggregates and their egress routers, and that a controller knows the current status of links, and link propagation delays, the job of the controller is then to calculate the path that each aggregate takes though the network, including splitting an aggregate between multiple paths if necessary. This can, in principle, be viewed as a simple optimization problem.

Many flows, including DNS and TCP short flows that terminate in slow-start, are latency-bounded. Their completion time is proportional to RTT. For longer TCP flows, if they compete with other TCP flows at a bottleneck, their throughput is inversely proportional to RTT [63]. If however, large flows are constrained by a narrow customer tail circuit, or are application-limited, as is often the case with video streaming, then throughput is largely independent of RTT.

Backbone congestion impacts both long flows and, due to queuing delay, latency-bounded short flows. Thus, a key constraint is to avoid all backbone congestion if feasible. Usually this is possible, as ISP backbones are generally provisioned so that they have enough capacity to cope with aggregate demand, with individual bandwidth-limited flows being constrained on customer tail circuits.

However, it is usually uneconomical to provision so that every link in the backbone is over-provisioned at all times if shortest path (as measured by propagation delay) routing is used. Instead it makes sense to statistically multiplex traffic peaks, running some links near peak capacity while offloading excess traffic to longer paths.

Finally, not all traffic is created equal. Short flows tend to be more latency-sensitive than long ones, so it makes sense to prioritize the routing of traffic with a lower mean flow throughput. This can be calculated from the measurement of aggregate traffic bandwidth and from an estimate of the number of flows in the aggregate. We will return to this issue later.

## 5.1 Objective

Given these demands, the controller's role is to optimize placement of aggregates so that *the sum of the propagation delays of all flows is minimized*, subject to the constraint that *all traffic aggre-*

*gates are placed on uncongested paths*, or split between multiple uncongested paths if necessary. Summing the propagation delays of all flows ensures aggregates containing many small flows are preferentially routed on low delay paths.

This formulation is readily expressed as a linear optimization, solvable by well-known general purpose LP solvers [21]. There are several ways to do this. One way is to cast the problem as a multi-commodity flow problem, with one commodity per aggregate of flows, in the spirit of Bertsekas et al. [13]. Such an optimization scales with the product of number of aggregates and number of links, and quickly becomes intractable for larger networks. We use an alternative formulation that explicitly considers paths; it is not only more efficient, but is also amenable to adding policy constraints (not discussed in this work).

We will discuss variability of aggregates in detail shortly but, for now, let us assume that each aggregate can be characterized by a single bandwidth value. If $A$ is the set of all aggregates and $L$ the set of all links, then the controller can optimize for delay by minimizing:

$$\sum_a n_a \sum_{p \in P_a} x_{ap} d_p$$

subject to the constraints that no link is overloaded:

$$\sum_a \sum_{p \in P_a} x_{ap} B_a < C_l \quad \forall l \in L$$

and that all flows are routed:

$$\sum_{p \in P_a} x_{ap} = 1 \quad \forall a \in A$$

Here, $n_a$ is the number of flows in aggregate $a$, $P_a$ are the paths $a$ might take, $d_p$ is the propagation delay of path $p$, $x_{ap}$ is the fraction of traffic from $a$ placed on path $p$, $B_a$ is the total bandwidth of $a$, and $C_l$ is link $l$'s capacity. Notice that this is the exact same objective as the one used by the optimal routing scheme from Section 3.2.2.

There are several problems with this formulation. First, it only cares about total delay. This becomes a problem when deciding which of two otherwise equal aggregates to evict from a full link. If we have two aggregates competing for a link, and we are going to need to move one to a long path, it makes sense to move the one whose RTT is already larger (example from Figure 4.2 in the previous chapter). This gives more predictable latency based on geography, for example, fitting better with how CDNs work. Thus a better function to minimize is:

$$\sum_a n_a \sum_{p \in P_a} x_{ap} (d_p + \frac{d_p M_1}{S_a})$$

where $S_a$ is the delay on $a$'s shortest possible path, and $M_1$ is a very small constant.

This formulation is also flawed if congestion could not be avoided. If the link-overload con-

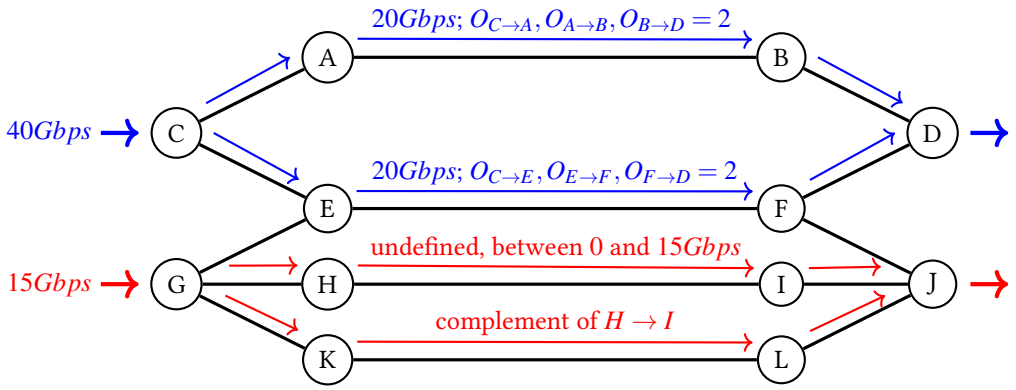



**Figure 5.1:** All links are 10 Gbps; $O_{max} = 2$, attained at the top links. If the optimizer uses the objective in 5.1 the bottom aggregate's traffic may end up congesting one of the bottom paths even though there is capacity for it to fit.

straint cannot be satisfied the solution is undefined. In such cases we wish to spread traffic so as to equalize links' overload, minimizing its effect. This leads us to a further refinement:

$$\left[\sum_a n_a \sum_{p\in P_a} x_{ap}(d_p + \frac{d_p M_1}{S_a})\right] + M_2 O_{max} \tag{5.1}$$

with the modified constraint:

$$\sum_a \sum_{p\in P_a} x_{ap} B_a < C_l O_l \quad \forall l \in L \tag{5.2}$$

and the additional constraint:

$$1 \leq O_l < O_{max} \quad \forall l \in L$$

Here, $O_l$ is the degree that link $l$ is overloaded, and $M_2$ is a very large constant. This formulation minimizes with the highest priority, the maximum level of overload, $O_{max}$, seen by any link. Minimizing $O_{max}$ in this way spreads congestion to the maximum extent, minimizing its effect.

The objective function in 5.1 will force the optimizer to set $O_{max}$ to its minimal value of 1 when there is enough capacity in the network to accommodate all aggregates' demands. Notice, however, that when traffic does not fit the objective function does not always perform as expected.

Consider Figure 5.1. In this simple network there are two aggregates—one with demand of $40Gbps$ and one with demand of $15Gbps$. Suppose that the optimizer uses the objective in 5.1 to pick a path assignment. The top aggregate will be split among the two top paths as shown, as any other split will cause a link in the network to be loaded above $O_{max} = 2$. Because there exists no assignment in which $O_{max}$ is below 2, the optimizer will happily accept *any* solution in which links are loaded up to 2 times their original capacity as equally good—note that the $M_2$ penalty in Equation 5.1 only applies to $O_{max}$. This means that in Figure 5.1 the optimizer might decide to put all of the $G \to J$ aggregate on the $G \to H \to I \to J$ path, as that would load links along that path to $O_l = 1.5 < O_{max} = 2$. This is clearly the wrong path assignment, as there is enough capacity to fit

the bottom aggregate among the two bottom paths without causing congestion (*e.g.,* by splitting it evenly among them).

To address this shortcoming we append the sum of oversubscription ($\sum_l O_l$) to the objective function:

$$\sum_a n_a \sum_{p \in P_a} x_{ap}(d_p + \frac{d_p M_1}{S_a}) + M_2 O_{max} + \sum_l O_l \tag{5.3}$$

In cases where there is a single part of the network in which congestion is unavoidable (like in Figure 5.1) minimizing both total and maximum oversubscription guarantees that congestion will be avoided in the rest of the network, if at all possible. If traffic fits and $O_{max} = 1$, all $O_l$ will also be 1, which essentially renders $M_2 O_{max} + \sum_l O_l$ constant.

In summary, the objective function derived above causes the solver to minimize three different objectives at different priority (ranked from high to low):

1. The solver will avoid causing congestion. Because the $M_2$ multiplier of $O_{max}$ is defined as a very large number, any value of $O_{max}$ above 1 will drive the value of the objective function significantly up, giving the solver an incentive to avoid solutions that cause even a single link's load to exceed its capacity.

2. Given all solutions that either avoid congestion, or minimize congestion if avoiding it is not possible, the solver will pick the one that minimizes propagation delay.

3. Given solutions that minimize delay equally, the solver will prefer ones that put more flows on paths that are close to each aggregate's respective shortest path. This property is due to the small $M_1$ constant multiplier of $d_p/S_a$—the ratio of the delay of the path divided by the delay of the shortest path in the aggregate. This fraction will always be equal to 1 or more, as all paths in an aggregate are at least as long as the shortest one. By trying to minimize the total value of the objective, and therefore the sum which those fractions are part of, the optimizer will avoid excessively long paths.

It is imperative that the solver observe this ranking of objectives regardless of the rest of the constants in the problem (*i.e.,* flow counts, aggregate volumes and link capacities) otherwise it may, for example, pick a solution that exhibits congestion even if a congestion-free one exists. Preserving the pecking order among objectives depends on correctly picking $M_1$ and $M_2$, which act to numerically separate the three objectives.

If our solver of choice uses arbitrary precision arithmetic, we can choose $M_2$ to be an arbitrarily big number, and $M_1$ an arbitrarily small number. Regrettably, even for tiny problem sizes, arbitrary precision arithmetic is prohibitively expensive, so we have to choose limited-precision numbers for $M_1$ and $M_2$. Even worse, the minimum and maximum values that we can pick for $M_1$ and $M_2$ depend on the rest of the constants in the problem and solver-specific numerical problems may

arise if the *range* of the objective or one of the constraints is too large [77]. We now examine exactly how to pick $M_1$ and $M_2$.

### Sizing Constants in the LP

Let us start with $M_2$. $M_2$ should be set to be the smallest number so that a small increase in its factor (the maximum oversubscription $O_{max}$) is guaranteed to drive up the value in the objective function in a way that trumps any increase due to the two lower-level objectives from the list above. In other words:

$$O_{max} = \frac{\text{propagation delay cost of worst possible move}}{p}$$

where $p$ is a small *problem-independent* constant, representing how much we are willing to relax the oversubscription constraint. We have discovered that a value of $p = 0.01$ is a reasonable default that is small enough to avoid oversubscription yet large enough to avoid causing numerical issues with the linear solver that we use. This is also the value that we use in all experiments that we run in this work.

Since the objective function minimizes delay, the worst possible move the optimizer can make is the one that will increase delay the most—moving all aggregates' flows from their aggregate's shortest path to its longest one. Even though such a move is unrealistic in most cases, as all flows are unlikely to fit on the shortest or longest paths, this approach gives a very liberal upper bound to use in the formula above:

$$O_{max} = \frac{\sum_a n_a \max_{p \in P_a}(S_a - d_p)}{p}$$

where $n_a$, $P_a$, $S_a$ and $d_p$ are the same as in Equation 5.3.

Similar logic can be applied to sizing $M_1$, however instead of the worst possible increase in the objective function, we should consider the smallest (best) possible increase due to propagation delay alone.

## 5.2 Minimizing $O_{max}$ Across all Links

A further complication is added because even after minimizing the worst overload, other disjoint parts of the network may also be overloaded, but to a lesser degree. We can only optimize $O_{max}$ for the worst congested region, so may not optimize congestion elsewhere. For example, consider Figure 5.2 which is very similar to Figure 5.1, but in this case the network does not have enough capacity to fit the bottom aggregate as well as the top one. When Equation 5.3 is used the top aggregate will be, just like before, split evenly among the two top paths. This time, however, the extra $\sum_l O_l$ term will not help: any split that causes the $O_l$ values of the paths of the bottom aggregate to sum up to 2.5 is equally good as far as the optimizer is concerned. Ideally we would

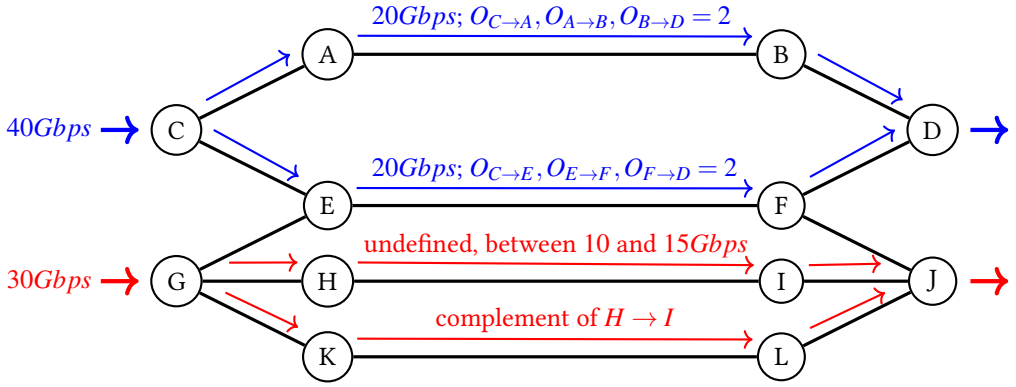



**Figure 5.2:** Same scenario as in Figure 5.1, but the bottom aggregate's demand is 30*Gbps* and not satisfiable by the two bottom paths. This creates two regions of different oversubscription in the network. A single application of Equation 5.3 will minimize oversubscription in the top part of the network, but not in the bottom one.

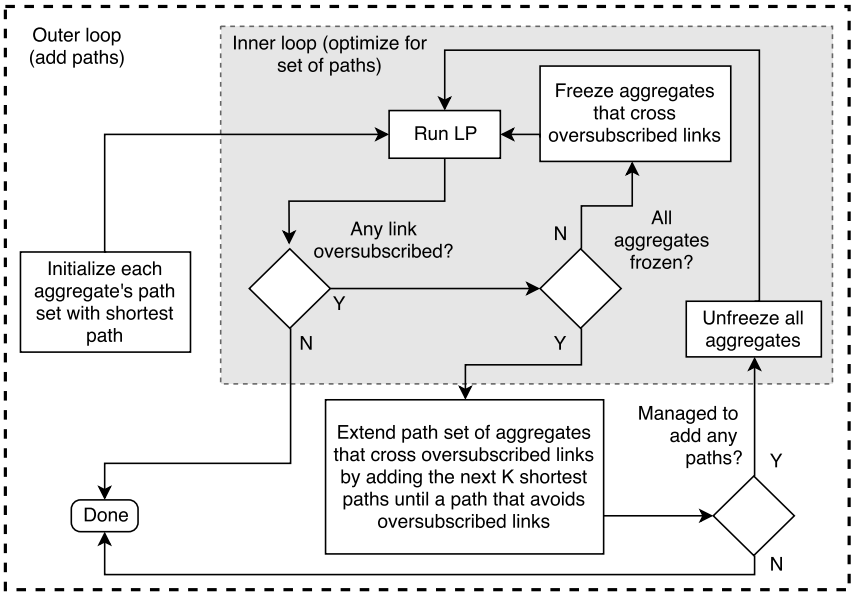



**Figure 5.3:** Obtaining paths and per-path aggregate fractions, assuming each aggregate's demand is known. This is a two-stage process: the inner loop uses Equation 5.3 to minimize both delay and oversubscription for a given set of paths (Section 5.2) and the outer loop adds new paths (Section 5.3).

like to spread congestion in the bottom part of the network, sending 15*Gbps* to each of the two bottom paths, but a single application of Equation 5.3 cannot do so.

The inner loop from the iterative solution shown in Figure 5.3 addresses this problem. Given all aggregates' paths, we optimize for the worst $O_{max}$, then freeze aggregates all of whose paths traverse links overloaded to degree $O_{max}$, removing them and the capacity they consume from further consideration. We then repeat this process for the remaining aggregates whose placements have not yet been frozen, find the new, lower value of $O_{max}$, and freeze those congested aggregates' placements, too. This process repeats further, successively freezing aggregates with lower and lower levels of oversubscription. It terminates with final path allocations when there are no more

"unfrozen" aggregates, or when constraint 5.2 can be satisfied with $O_{max} = 1$, at which point the remaining flows have found optimal lowest latency uncongested paths.

In Figure 5.2 the iterative process will first freeze the top aggregate as both of its paths traverse links with $O_l = O_{max} = 2$. After the first iteration the top aggregate and top links are frozen, and another run of the optimization will only consider the bottom aggregate, evening out the load on the two bottom paths as well.

### Informal Proof of Inner Loop

We now sketch an informal proof that this iterative process of linear optimization conducted under overload yields the lowest possible level of congestion on all links in the whole network.

By definition a single application of Equation 5.3 will minimize the maximum oversubscription of any link in the network. Let's assume the value of the maximum oversubscription is $O_l$, and the set of links that have this oversubscription is $L_o$.

We will prove by contradiction that if one of the paths of an aggregate crosses a link in $L_o$, then all of its paths must also cross at least one link in $L_o$. Let us assume the opposite—that there is at least one aggregate with one or more paths $P_l$ that cross a link in $L_o$, and one or more paths $P_i$ that do not cross a link in $L_o$. This implies that by moving some positive amount traffic from all of the $P_l$ paths to any of the $P_i$ paths we can lower the oversubscription of the links in $L_o$. But by definition, because of the LP, their oversubscription cannot be further minimized. Therefore if one of the paths of an aggregate crosses a link in $L_o$, then all of its paths must also cross at least one link in $L_o$.

Notice that this implies that links in $L_o$ cannot be occupied by aggregates that have no paths that cross links in $L_o$. If we "freeze" all aggregates that cross at least one link in $L_o$—i.e., remove them from the network and remove their traffic from the capacity of all links along their paths, we will be removing all capacity on all links in $L_o$. We can repeat the application of Equation 5.3 to produce a new solution on the network with the modified link capacities, which will then minimize the oversubscription of a different set of links. The repeated application of this procedure is guaranteed to minimize the oversubscription of all links in the entire network.

This procedure is guaranteed to terminate because at each iteration we are freezing at least one aggregate. As we will later demonstrate, in practice this iterative approach terminates quickly since multiple aggregates will be frozen at each step.

## 5.3 Adding Paths Iteratively

Finally, we must face the issue that performing this optimization for the set of all possible paths is computationally infeasible for large networks. Fortunately, we observe that this is not necessary, given the objective function in Equation 5.3. Consider a single aggregate that does not fit on its shortest path. An optimal solution will place as much traffic as possible on the shortest path, then

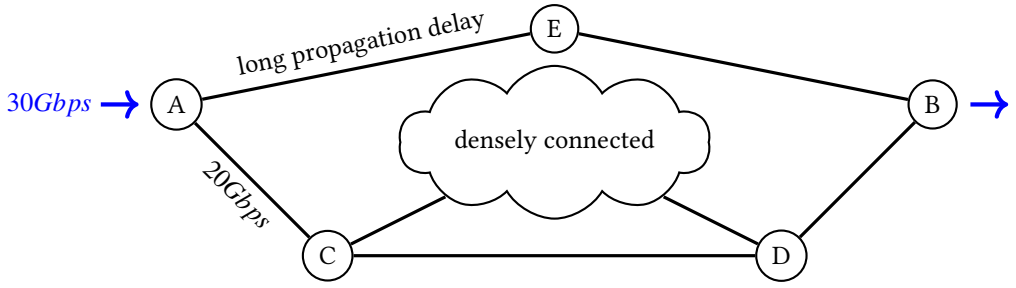



**Figure 5.4:** A simple pathological example. All links have the same $20Gbps$ capacity and there is a single aggregate whose shortest path passes through $C$; the top path via $E$ has longer propagation delay than any other path from $A$ to $B$. LDR's iterative path addition process will have to add all paths that go through the densely connected region before the only viable path via $E$ is discovered.

allocate the rest to the next-best path. If the next-best path is not itself congested, then adding further paths for this aggregate serves no purpose, as they will never be used. Essentially, there is a "delay threshold" then for each aggregate, beyond which paths of longer delay will never be used. We do not know this delay threshold a-priori, but we can learn it from successive runs of the LP optimizer.

As shown in Figure 5.3, we associate each aggregate with a list of its $k$ shortest paths, where initially $k = 1$. All paths from the aggregates' lists are added to the set of paths given to the LP. We then use the iterative optimization from Section 5.2 (inner loop in Figure 5.3) to minimize both propagation delay and oversubscription. The inner loop will do its best to find a solution with $O_{max} = 1$. If it fails, we look for links with $O_{max} > 1$. For all aggregates that cross those links we extend the list of paths by generating shortest paths for an increasing $k$ until we find a path with no $O_{max} > 1$ links. We then run the inner loop again.

While in essence greedy this approach of iteratively adding the $k$ shortest paths is guaranteed to provide the same solution as providing all paths to the optimizer—at the final iteration of the algorithm all paths below each aggregate's "delay threshold" are present in the set of paths available to the optimizer, and the optimizer would never use any paths above the threshold.

Even though this approach involves multiple runs of the LP optimization, it actually runs very quickly because the number of variables (paths) in each run is small. The bottleneck is not the linear optimizer, but the $k$ shortest paths algorithm [82], the results of which can be readily cached yielding sub-second runtime even with tens of thousands of aggregates—97.3% of all algorithm runs in Chapter 7 completed under a second.

## 5.4 Path Addition Heuristic for Large Networks

A drawback of the iterative path addition process described above is that if the network is large, densely connected and very loaded the optimizer may need to add shortest paths for a very large value of $k$ before it reaches a path with no oversubscribed links.

A simplified example of this behavior is shown on Figure 5.4, where the only aggregate in

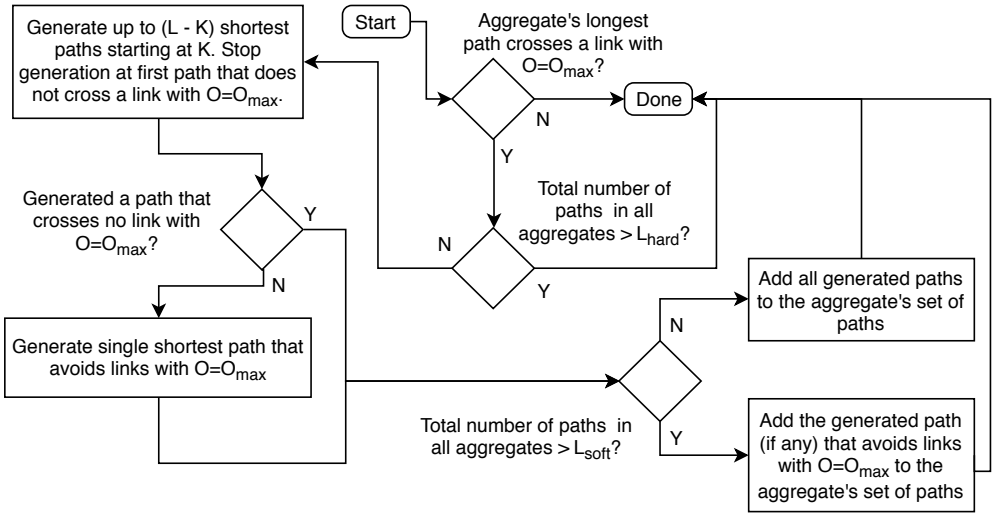



**Figure 5.5:** Adding paths to a single aggregate, assuming the aggregate's set of paths is pre-populated to contain its shortest path. When the longest path in the set is oversubscribed, $k$ shortest paths are added until either a non-oversubscribed path is found, the aggregate's path set has grown to the per-aggregate limit ($L$) or the global soft path limit is hit ($L_{soft}$). After either of those limits is reached paths are skipped to avoid exploring densely connected regions as in Figure 5.4. No paths are added if the hard limit ($L_{hard}$) is reached.

the network goes between $A$ and $B$. The iterative optimization and path addition process described above will start by placing as much of the aggregate as possible on its shortest path $A \to C \to D \to B$. As the aggregate does not fit entirely on its shortest path the first optimization pass will conclude with links $A \to C$, $C \to D$ and $D \to B$ being oversubscribed at $O_{max} = 1.5$. The optimizer will now attempt to run the $k$ shortest paths algorithm for progressively larger values of $k$ until it finds a path between $A$ and $B$ that avoids any oversubscribed links. The problem is that in this case the only path that satisfies this condition has a comparatively high delay and will not be explored until *all* paths between $A$ and $B$ that go through the densely connected part of the network are explored and added to the problem. If that part of the network is sufficiently connected there will likely be an exponential number of such paths. Clearly, adding this many paths to the problem is not feasible.

An obvious way to deal with scenarios like the one in Figure 5.4 is "skip over" to the top path without adding all paths after $k = 1$. This way the problem that the LP solver needs to solve will only have two paths and in this case the solution will still be optimal, even though we did not feed a large number of paths to the optimizer. In general, however, it is not possible to guarantee optimality if not all paths up to each aggregate's "delay threshold" are added to the problem.

LDR follows the above approach—in the example from Figure 5.4 LDR will give up after adding a certain number of paths. After this happens, the iterative addition process will artificially increase the cost of all oversubscribed links which will cause the $k$ shortest paths algorithm to skip over oversubscribed parts of the network.

In Figure 5.5 we describe the behavior of this heuristic by focusing on how paths are added

**Table 5.1:** Limits used when adding paths

| limit (designation on Figure 5.5) | value |
|---|---|
| hard ($L_{hard}$) | 200000 |
| soft ($L_{soft}$) | 100000 |
| per-aggregate ($L$) | 1000 |

to a single aggregate. The figure assumes that the aggregate's shortest path is already in its set of paths—this will always be true for all aggregates, as regardless of the outcome of the optimization each aggregate needs at least one path. The first check that happens is whether the aggregate's longest path crosses any oversubscribed links (links with $O = O_{max}$). As previously discussed, if this is not the case we have reached the aggregate's "delay threshold" and we know that there is no need to add any additional paths to this aggregate.

The size of the optimization problem, and therefore the time to solve it, is directly proportional to the number of paths. As LDR is meant to run online, it is paramount that we provide some hard limit on the runtime of its solver. To this end, LDR imposes a global limit for the total number of paths added to the problem across all aggregates. If we have reached this limit, we will not add any more paths to any aggregate.

If the hard limit has not been reached, we will attempt to generate the next $L - K$ shortest paths, starting at the number of paths already added to this aggregate—$K$. In LDR there is a per-aggregate path count limit $L$ which ensures that in case there are multiple aggregates in the situation from Figure 5.4 no single aggregate starves other aggregates from paths.

If either we cannot generate any additional paths or none of the paths that we generate avoid links with $O = O_{max}$, we will bump up the cost of all links with $O = O_{max}$ and generate the single shortest path between the aggregate's source and destination—this will solve the problem from Figure 5.4.

Finally, LDR employs a soft limit above which only paths that avoid oversubscribed links are added, even if the number of paths in the aggregate has not reached $L$. The soft limit allows more paths that avoid oversubscribed links to be added before hitting the hard limit—it is useful in cases where there is a very large number of aggregates compared to the size of the hard limit (*e.g.,* there are 10000 aggregates and the hard limit is set to 20000 paths).

Table 5.1 shows the values of the three limits that we use throughout the evaluation of LDR. We note that those values are high enough so that in moderately loaded networks, like the ones in Figure 3.9, the limits are almost never used. We explore in Section 7.3.7 conditions under the limits will be used.

## 5.5 Prioritizing Traffic in LDR

The description of the optimization process so far has assumed that all flows in the network are equally delay sensitive. The objective function from Equation 5.3 prioritizes each aggregate based

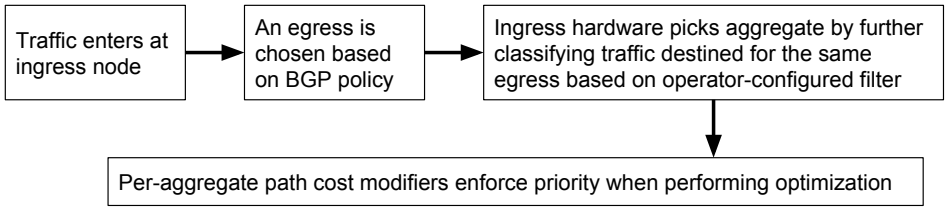



**Figure 5.6:** Optimization process with extended aggregates; each aggregate is defined as a combination of $\langle ingress, egress, filter \rangle$.

on the number of flows within the aggregate. Assuming that flow counts are accurate, in an optimal solution all flows in the network will benefit from low delay paths to the same extent. This is a fair outcome if the ingress points cannot differentiate between different types of traffic, and the operator does not possess any insight into the type of traffic carried by each aggregate.

In reality, however, all types of flows benefit from lower delay, but some types much more so than others. For example, within the confines of the same aggregate, long-lived NetFlix TCP sessions, which carry non-interactive video traffic, should not be treated the same as short-lived web TCP transfers. If the ingress of the network can differentiate between the Netflix traffic and the web traffic (*e.g.*, by examining IP addresses in packet headers), it should be possible for the routing system to give lower-delay paths to the web traffic. How can LDR support such finer-grained prioritization of traffic?

To handle the example above, we need to extend our definition of aggregate so that the routing system supports multiple aggregates between the *same* pair of ingress and egress nodes. We then need to extend LDR's optimization process to handle the new aggregates and apply prioritization to them as needed by the network administrator. Figure 5.6 outlines how we extend LDR to support prioritization.

### Multiple Aggregates Between the Same Pair of Nodes

As usual, when traffic enters the network, an egress is chosen for it based on BGP policy. An aggregate is then picked by the ingress by further applying an operator-defined filter to all traffic destined for the same egress. This way an aggregate is identified by a 3-tuple of $\langle ingress, egress, filter \rangle$ instead of the usual 2-tuple of $\langle ingress, egress \rangle$. Obviously, if the hardware does not support classification, or the network operator has not configured a filter, effectively the definition of an aggregate is reverted back to the original one.

The optimization process, as previously described, does not depend on the actual definition of an aggregate, so it can run unmodified with our extended aggregate definition.

### Prioritizing Aggregates

Armed with our finer-grained aggregates, we now arrive at the heart of the problem—how can we enable the operator to prioritize different aggregates?

The most straightforward approach is to artificially inflate the number of flows in the aggre-

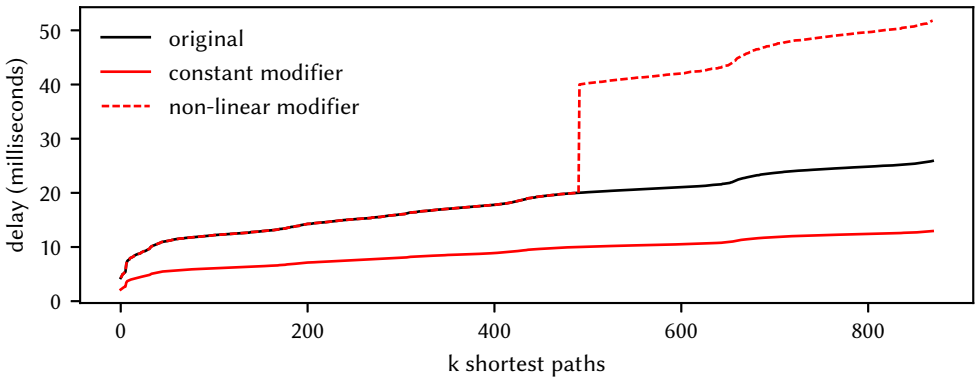



**Figure 5.7:** Different priority modifiers.

gate that we want to prioritize—the larger the flow count, the more costly it is for the optimizer to put this aggregate on higher-delay paths and the more unlikely it would be to do so. This simple scheme can be accomplished by adding a per-aggregate priority multiplier to the aggregate's flow count $n_a$ from Equation 5.3.

On one hand, the new priority multiplier gives the operator the ability to apply lower-delay treatment to traffic that they know to be delay sensitive, and that they can identify as it enters the network (*e.g.,* short web transfers). It also provides them with the ability to de-prioritize traffic that they know is not as delay sensitive (*e.g,* Netflix TCP flows), with the desirable property that in the absence of higher-priority traffic, the optimizer will put even lower-priority traffic onto desirable low-delay paths.

On the other hand, a simple multiplier may be too coarse-grained for a large class of policies that the operator may want to apply, such as "ensure that the delay of the aggregate does not go above 20*ms* if possible"—a policy the operator may want to support as part of an SLA with a customer. Figure 5.7 shows an example of the cost (propagation delay in milliseconds) of the $k = 860$ shortest paths paths of an aggregate in Cogent's network (curve labeled ORIGINAL); the total cost of the aggregate in the final solution will be based on how many flows the optimizer will assign to each of these paths. Notice that the curve forms a non-decreasing additive cost function of the path delay $f(p)$. In the case of the ORIGINAL curve $f(p) =$ *delay of path p.*

Applying a constant multiplier is akin to multiplying the cost of each path by the same number. For example the curve CONSTANT MODIFIER is obtained by multiplying all paths' cost by .5.

The cost curve that will let us support the example policy from above is the one labeled NON-LINEAR MODIFIER. It preserves the cost of the paths up to the 20*ms* threshold, above which the cost of the path shoots, while still remaining non-decreasing.

While traditionally such a non-linear $f(p)$ would be costly to incorporate into the optimization process, our path-based formulation makes it trivial—as seen from Equation 5.3, from the viewpoint of the optimizer per-path costs are constants. As the $k$ shortest paths are progressively added to the

optimizer (see Section 5.4), any arbitrary non-decreasing $f(p)$ can be applied to them to generate per-path costs. This approach gives the operator a great degree of flexibility, as long as they are intimately familiar with their network's demands and it is possible to distinguish between different types of traffic. As we will demonstrate in Chapter 7, even if either of those is false, the default of treating all traffic as equally sensitive to delay results in a solution that yields very low delays.

## 5.6 Reordering, Jitter and Control Plane Overhead

In a network managed by a load-dependent routing system it is expected that changes to the traffic matrix will translate to changes to the routing state of network devices. However, we want to avoid cases where a small change in flows' demands results in significantly different routing state. Such large changes are undesirable because they:

- lead to a large volume of management traffic and processing overhead at network devices. Even in modern state-of-the-art OpenFlow switches processing an update can take many milliseconds and put strain on the device's control plane.

- cause packet reordering. Reordering can negatively affect the throughput of TCP sessions by causing them to unnecessarily reduce their transmission window.

- increase jitter. Real-time flows moving to paths with significantly different propagation delays will be negatively affected.

- render the network harder to manage. It is much harder for the network operator to debug a problem with the network when the routing state at switches constantly changes.

- can cause transient congestion. Imagine that the change results in swapping a large volume of traffic between two links where at least one is loaded close to saturation. If traffic is added to the already-loaded link before traffic is removed from it, the link's queue will overflow. Previous efforts alleviate those transient effects [46], but there are cases (for example if both the source and the destination links are very loaded) where no ordering exists which will avoid transient congestion when the change is large.

As discussed in Chapter 2, traditional MinMax-based load-dependent routing systems focus on minimizing link utilization. This creates a feedback loop: placement of flows on links influences the level of traffic on the link, which directly affects link utilization. In a distributed scenario different devices may adapt to link utilization measurements at different times, reacting to the output of a different device, leading to oscillations.

LDR's centralized optimization process does not suffer from such oscillations, but it provides no guarantees that small changes in the optimization's input will not cause a large fraction of the flows to take a different path. Even though the optimization function has a single unique solution for a traffic matrix it can be ill-conditioned—there are no guarantees that a tiny change in the traffic load of an aggregate will not lead to an entirely different routing solution. We will, therefore, define *well-conditioned* in the context of LDR (and other centralized load-dependent routing systems) to mean responding to small changes in traffic demand by proportionally small changes to routing state.

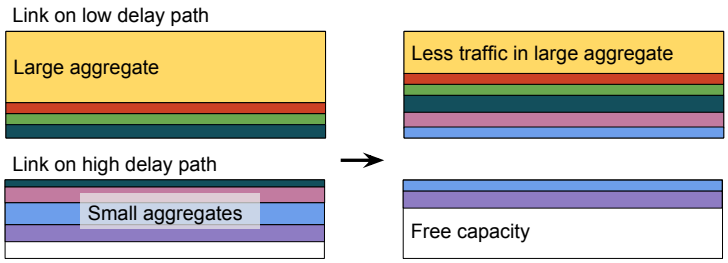



**Figure 5.8:** Effect of a single large aggregate's demand decrease

There are two main inputs to LDR's optimization process—per-aggregate demands and per-aggregate flow counts. We will now examine how a change to either of those can influence LDR's output.

### 5.6.1 Changes in Per-aggregate Demands

On the left side of Figure 5.8 we show a typical LDR output with respect to two links in the network and a handful of aggregates with varying sizes (demands). The top link is part of the shortest paths of the aggregates shown, while the second link is part of their higher-delay paths. In this example let's assume that each aggregate's flow count is proportional to its volume.

LDR drives the link that is on the lower-delay paths to a high utilization, as assigning more flows to it will result in lower total per-flow propagation delay. Not all aggregates fit on their shortest path, and to avoid congestion some aggregates are routed over the second link. Note that LDR's optimization does not "stripe" aggregates, but instead packs them onto links, only splitting a single aggregate in this example. Let's examine what happens when the measured level of the large aggregate decreases. The new solution will depend on what the actual per-path delays are, but one possible outcome is shown to the right. The shrinking aggregate frees up room for some of the smaller aggregates, and LDR places them onto a shorter path (and hence onto the top link).

The flows of the aggregates that move will experience reordering, as they have just been moved to a path that has lower propagation delay. A move in the opposite direction—from a low delay path to a higher delay one is less disruptive as it causes no reordering, but will, by definition, result in some flows experiencing longer delay.

In general every time an aggregate that is currently being routed over a high-utilization link changes capacity there will be a knock-on effect on other aggregates, perhaps even aggregates it shares no links with—imagine that the second link in Figure 5.8 is also full and the shrinking aggregate causes LDR to move flows from the second link onto a third one. Whether or not this is the desirable outcome depends on a lot of factors. It may be that a disruptive change like the one from Figure 5.8 yields a negligible overall decrease in total delay experienced by all flows. The optimization process will always produce the optimal solution, but we may be paying a hefty price,

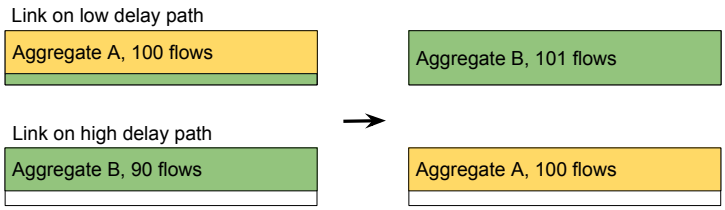



**Figure 5.9:** Effect of a single aggregate's flow count change

in terms of changing paths in the network, to only gain a little bit of optimality.

### 5.6.2 Changes in Per-aggregate Flow Counts

LDR's optimization function minimizes the total delay of all flows in the network. This implies that each aggregate's contribution to the overall value of the objective function is dependent on its flow count. How does the system react when there is a change in the relative ordering of aggregates' flow counts?

In Figure 5.9 we show two links with two aggregates. To the left we have one aggregate ($A$) with 100 flows and another one ($B$) with 90 flows. To the right is LDR's solution when $B$ grows to have 101 flows, but its total volume remains unchanged. Because the new flow count for $B$ is higher than that of $A$ the optimizer decides that the optimal solution is to put $B$, instead of $A$, on the shortest path. While this solution does indeed place more flows on the shortest path and results in a slightly better total flow delay, the overall gain is tiny compared to the cost of completely swapping the two aggregates.

### 5.6.3 Limited Optimization

How can we limit the negative effects when either an aggregate's demand or its flow count changes? From the examples above it should be obvious that instead of always going for the optimal solution we should aim for one that is good, but avoids excessive change. A natural way to produce such a solution is to restrict the optimizer's freedom when placing flows so that it remains close to the one from the previous, currently installed in the network, solution. At the same time we do not want to be too restrictive—when an aggregate's volume increases, changing how it splits traffic among its paths may be inevitable; in such cases we want to avoid moving aggregates to shorter paths (as that would cause both jitter and reordering) in favor of moving aggregates to longer paths (as that would only cause jitter).

Consider an aggregate and one of its paths in two LDR solutions obtained at time $t$ and $t+1$. If in the $t+1$ solution the path carries less traffic than it did in the $t$ solution, then clearly the optimizer has chosen to move some of the path's traffic (and hence its flows) onto a different path. As previously discussed, we would prefer it if those flows moved to a higher-delay path, as that would avoid reordering. How can we make the optimizer only consider such moves? If the path

in question is the aggregate's shortest path and it carried $X\%$ of the aggregate in the old solution, we can simply limit it to carry no more than $X\%$ in the new solution as well. The limit will ensure that no new flows are moved to the shortest paths, and any flows moved away from the shortest path will necessarily go on longer paths.

What if the path is not the shortest path? We do not want to always limit all paths in a similar way, because when an aggregate grows the optimizer needs to be able to find room for it. Remember that LDR's objective function will always "pack" traffic into the first $k$ shortest paths of the aggregate. This means that when an aggregate's demand increases the optimizer will insert the excess capacity on on either the longest path in the previous solution, or a new path that is longer. We, therefore, need to never apply the limit to the longest path in the aggregate, or any paths that are longer. This observation leads to a simple limiting rule. Consider an aggregate $a$ and let $D_a$ be the delay of its longest path in the solution at time $t$. When generating a new solution at time $t+1$, for each of the paths of $a$:

- if the path's delay is less than $D_a$, limit the fraction of the aggregate that can go on the path to never exceed the path's fraction from the previous solution, or 0 if the path was not part of the previous solution.
- if the path's delay is more or equal to $D_a$ apply no limit.

Applying those limits to LDR's LP definition has the effect of producing a solution which is sub-optimal with respect to an unconstrained solution when aggregates' demands decrease. This is easy to see—a decrease in an aggregate's demand means that there is now free capacity on its paths. But the limits described above will prevent the optimizer from placing a larger fraction of the aggregate on any of its previous paths (except the longest one), which results in sub-optimal total per-flow delay. In essence the per-path limits avoid reordering by ignoring opportunities to pack flows more tightly onto low-delay paths.

If the limits are applied on every update cycle the solution will drift further and further from the optimal one as the traffic matrix changes. To discover the magnitude of this drift LDR runs in parallel two versions of the optimization—one with the limits applied, an another one without them. The total per-flow delay values from the two solutions are compared and if the delta is above a small threshold (1% by default) the unlimited solution is applied. To avoid causing reordering to a large number of flows simultaneously, it is possible to relax the limits to a handful of aggregates at a time instead of to all at once.

When the network is very loaded, applying the limits may cause the optimization to fail to fit all aggregates' demands, even when they would fit without the limits. In this case LDR will always use the solution that fits the demand, preferring to cause reordering rather than potential congestion within the network that it manages.

## Chapter 6

# Characterizing Demand

To run the optimization from Equation 5.3, the controller needs to know the number of flows in each aggregate, $n_a$, and the aggregate's demand, $B_a$. How should it obtain these values?

## 6.1 Counting Flows

Any system designer aiming to achieve the goals of Chapter 4 is faced with a dilemma—while the aim is to minimize the completion times of flows that cross the backbone, performance constraints preclude identifying and routing single flows or directly measuring their bandwidths within the backbone's core. It is, however, feasible for ingress routers to estimate the number of flows comprising each aggregate destined for each egress router, and to send this information to the controller.

Counting flows has received ample attention in the literature. Rather than rehash known techniques, we merely note that a router may include hardware support of the sort described by Estan and Varghese [30] or, absent hardware support, (*i.e.,* on today's legacy routers), an external box may estimate aggregates' flow counts from sparsely sampled mirrored packet streams, such as those provided by sflow [72]. In our LDR implementation we use an adaptation of Duffield *et al.*'s technique [27] on a sampled stream of 1 in 100 packets.

At that sparsity, the number of flows observed will be an underestimate—many short flows will be missed. Nevertheless, our experience assessing the accuracy of this sampled method on packet traces drawn from a collection of U.S. Tier-1 backbone links is that the resulting estimate of flow count is within a factor of two of ground truth—good enough for our purposes.

Counting flows is not essential to LDR's operation. If a device does not support sampling or the cost of enabling it is too high, $n_a$ will be ignored by the optimizer. As a result LDR will not be able to weight aggregates proportionally to their flow counts, and may thus sometimes violate goal 3 from Chapter 4. Ultimately, the choice between these flow-count-aware and flow-count-agnostic versions of LDR is a matter of cost-benefit analysis for the operator. We will describe the full flow-count-aware design of LDR and evaluate both versions.

## 6.2 Aggregate Demand

In addition to the number of flows, the objective of our linear optimization requires a value for each traffic aggregate's expected demand, $B_a$. In real networks these demands are not constant; the mean demand changes over time in a largely predictable way, but on short timescales demand varies around the mean in an unpredictable manner. All aggregates will vary on short timescales to some degree, but some are more inherently variable than others, depending on the type of traffic and number of flows that comprise the aggregate. Our goals require us to statistically multiplex multiple aggregates onto links, and to run core network links at high utilization, but to do so in such a way that significant queues do not build.

### 6.2.1 Adding Headroom

A simple strategy to dealing with variability is to allocate a small fixed amount of headroom to all links in the network, expecting that it will be enough to buffer any short-term variability and prevent queues from forming. We can compute a prediction of the mean rate for each aggregate, based on its measured behavior from the last minute. Using this mean value as an estimate of $B_a$, we can run the optimization from Chapter 5, with the capacity of each link in the network reduced by the amount of headroom we want to allocate. How much headroom can we add?

Obviously, we need enough headroom to cope with variability, but there is another concern. We need to consider the bandwidth-delay tradeoff that we discussed in Section 3.3—adding headroom to *all links* makes it easier for the system to predict traffic levels and deal with transient congestion, but at the same time it will unnecessarily lengthen some flows' paths. The total delay experienced by flows in the network will always be an increasing function of the amount of headroom added, but how strong is that dependency?

In some cases, the combination of topology and traffic matrix may be such that we can comfortably add a very significant amount of headroom (*e.g.,* 10%) to all links, and only increase overall delay experienced by flows by a fraction of a percent—*e.g,* because for a lot of aggregates there exists an alternative path with propagation delay very close to the shortest path one.

In other cases, the alternative path may be significantly longer—*e.g.,* perhaps it crosses a backup link with very long propagation delay. In those cases we should drive the shortest path to as high a utilization as we can, while avoiding queuing.

Realistically loaded traffic matrices on real-world topologies will likely fall under both of the categories above. Even for the same topology, different traffic matrices can exhibit widely varying levels of sensitivity to headroom. In a large fraction of the topology/traffic matrix combinations that we examine in Chapter 7 the simple approach of adding headroom to all links is an overkill and results in low-delay capacity being wasted. Moreover, even if we were to choose a single amount of headroom to add to all links, how would we go about picking the right amount? Any number derived from our measurement data is likely to not be globally representative. Perhaps there is a

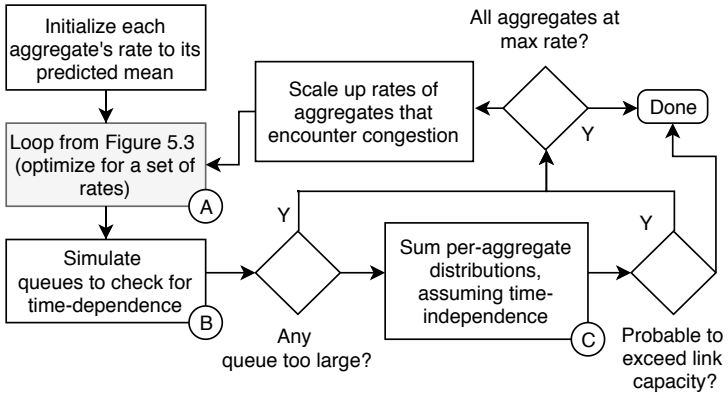



**Figure 6.1:** Picking rates to cope with short-term variability. Initially each aggregate's level is set to that of its predicted mean (see Section 6.2.3), the optimizer is then run (Ⓐ) and the solution is checked for temporally correlated events (Ⓑ) and whether multiplexed traffic is likely to overflow any queue in the network (Ⓒ). If either of those is true the levels of aggregates that are likely to experience congestion are moved closer to their maximum and the process repeats.

more general approach that we can take.

### 6.2.2 Statistical Multiplexing

Even if we knew the future variability of all aggregates perfectly, we would still be faced with a circular dependency. To run the optimization and place traffic onto links we first need to know $B_a$ for each aggregate. Unfortunately, whether or not traffic statistically multiplexes onto a link without causing excessive short-term queuing depends on which aggregates share the link, and therefore on the outcome of the optimization. In effect, we cannot choose a representative value of $B_a$ for each aggregate without knowing which other aggregates it will multiplex with, but we cannot decide paths without knowing $B_a$. How can we break this cycle?

A usable value of $B_a$ must lie somewhere between the aggregate's mean rate and its maximum rate. Using the mean would implicitly assume no short-term variability, whereas using the maximum rate would assume the worst possible statistical multiplexing, where the traffic peaks of all aggregates are perfectly synchronized.

Our strategy is shown in Figure 6.1. First we compute a prediction of the mean rate for each aggregate, based on its measured behavior from the last minute. Using this as an initial estimate of $B_a$, we perform a preliminary optimization of aggregate placement.

We then assess how well these aggregates will statistically multiplex. To do so, we make the observation that although the precise short-term demand of an aggregate is unpredictable, its degree of variability is predictable, as the type of traffic and mean number of flows in large aggregates does not usually change rapidly (see Section 3.3.2). Thus we measure each aggregate's variability during the previous minute, and use this to predict how variable it will be in the next minute (see below).

Knowing the variability allows us to evaluate whether the preliminary assignment of aggre-

gates to paths results in acceptable statistical multiplexing on each link. If this test passes for all links, then the traffic placement is good. If it fails for any link, we scale up $B_a$ for those aggregates traversing that link, and re-optimize (Ⓐ in Figure 6.1). This procedure is guaranteed to terminate because $B_a$ is bounded above by the maximum rate of that aggregate; if $B_a$ were to be scaled up to the aggregate's maximum rate, then the statistical multiplexing test is guaranteed to succeed. In practice it will usually succeed nearer the mean than the maximum.

### 6.2.3 Predicting Mean Traffic Level

In Figure 3.15 we presented data that suggests short-term variability in WAN traffic remains largely constant from one minute to the next. These findings lead us to believe that we can reliably predict how well aggregates multiplex, but of course the data does not imply that aggregates are stationary—as we demonstrated in Figure 3.14 aggregates' long-term variability exhibits more variation, with mean traffic levels likely to evolve in time. Below we outline the approach that LDR uses to deal with long-term variability.

At the beginning of each minute-long epoch, the controller examines all updates it has received during the previous minute from all ingress routers. Those updates contain the traffic levels of each aggregate. LDR's controller could simply estimate the mean traffic demand an aggregate is expected to offer a minute in the future by adding 10% to the mean per-second traffic levels from the previous minute. While this simple prediction algorithm handles most common cases, it is possible for an aggregate to exhibit greater minute-to-minute variability [41]. To be able to also handle those cases, LDR errs on the side of caution and slowly decays its prediction of the mean traffic level when the actual measured level decreases, but quickly increases its prediction when the measured level increases. In Algorithm 1 we show the conservative strategy that LDR uses to estimate mean throughput and hence the initial value of $B_a$ for each aggregate.

Note that this algorithm will always attempt to reserve 10% headroom on all links in the network. In Section 7.4 we further use packet-level simulation to demonstrate that this target headroom amount is appropriate for our evaluation scenarios. We acknowledge that other network conditions may require different headroom targets, and in Section 6.3.3 we outline a simple mechanism that would allow the network operator to sense the headroom target value appropriate for their network.

### 6.2.4 Assessing Link Multiplexing

Predicting the mean traffic level for each aggregate gives us an initial estimate for $B_a$—the traffic level that we need in order to minimize Equation 5.3. However, as we previously explained, the mean level is just a starting point. What we are after are per-aggregate values for $B_a$, somewhere between each aggregate's mean and its maximum, that will result in a solution with just enough headroom to minimize propagation delay while at the same time avoiding congestion. Obviously in order to pick those values we need to take into account each aggregate's short-term variability.

**Algorithm 1:** Predicting next minute's mean level from the previous minute's measured mean level. By default 10% headroom is reserved to allow aggregates' mean levels to evolve.

```
prev_value                          // Value measured last minute
prev_prediction                     // Value predicted last minute
decay_multiplier ← 0.98             // 2% decay when level drops
fixed_headroom ← 1.1                // Always scale level by 10%
scaled_est ← prev_value * fixed_headroom;
if scaled_est > prev_prediction then
    next_prediction ← scaled_est;
else
    decay_prediction ← prev_prediction * decay_multiplier;
    next_prediction ← max(decay_prediction, scaled_est);
```

For example, if an aggregate's demand is highly variable, but it is placed on the basis of its mean demand so that it occupies 95% of a link, it is highly likely that this aggregate will cause temporary queues to build. As the controller should not choose solutions that build queues, it should instead place the offending aggregate according to a rate closer to its maximal demand, rather than its mean.

To capture both the long-term variability of aggregates (addressed in Section 6.2.3) as well as their short-term variability (addressed in this section), LDR's ingress routers read hardware counters multiple times each second and send the controller a stream of raw counter values. These periodic readings form a discrete fine-grained distribution of the aggregate's traffic level. DevoFlow [22] demonstrated that commodity network devices can comfortably poll hardware counters in a switch with two thousand rules (*i.e.,* with one rule per aggregate) ten times per second. More recent proposals [58] promise to make reading counters even cheaper. LDR broadly follows the DevoFlow approach: over the course of every minute, each ingress switch sends to the controller 600 counter values (one for every 100 ms interval) for each of its aggregates.

While mean-rate prediction is possible on a minute-to-minute basis, accurately predicting the actual traffic level of the aggregate on sub-second (or even second) granularity is likely not possible in WAN environments. Luckily, we do not require such fine-grained prediction. Instead, given a provisional placement of aggregates to paths, we only need to decide whether short-term variability is likely to lead to excessive queuing. Aggregates may fail to statistically multiplex, either because traffic bursts in different aggregates are temporally correlated, or because these aggregates are just too variable so when multiplexed they are statistically likely to exceed link capacity.

Correlated bursts may, for example, occur if there are multiple aggregates going to a set of CDN nodes. If a periodic process updates those nodes, bursts in those aggregates may be synchronized. To test for this, the controller effectively simulates the previous minute's traffic (Ⓑ in Figure 6.1). For each aggregate, it has measurements of data transmitted in each 100 ms period. It simply sums the values from each aggregate for each 100 ms period to test if the link bandwidth would be exceeded. If it is, the excess traffic would be queued, so is carried over to the next 100 ms

period. The controller rejects any solution in which transient queuing delays exceed 10 ms, as this might impact real-time traffic.

To evaluate uncorrelated statistical multiplexing (Ⓒ in Figure 6.1), the controller treats aggregates as random processes, each with a different discrete distribution given by its 100 ms bandwidth measurements. When these aggregates multiplex on a link, we care about the resulting multiplexed bandwidth distribution.

We treat each aggregate's measurements as a probability mass function (PMF). For each link, we take the convolution of the PMFs of aggregates that cross that link, and examine the convolved PMF. If the probability that this PMF exceeds the link's capacity is below a threshold, LDR concludes that the aggregates will multiplex well enough to fit. The threshold comes from the maximum 10 ms queue the controller is willing to allow—the measurements span an interval of 60 seconds, during which we may allow the rate of the link to be exceeded for up to 10 ms; this yields a threshold of $\frac{10}{60000} = 0.00016$.

If a solution is rejected due to correlated or uncorrelated multiplexing failure, the values of $B_a$ for some aggregates were too low. The controller then examines the aggregates that failed to multiplex and linearly steps their value of $B_a$ closer to their maximum rates (outer loop in Figure 6.1). The optimization is repeated, and statistical multiplexing reassessed. This process continues until it finds a solution that yields sub-10 ms queuing.

It might seem that convolving discrete per-aggregate distributions is prohibitively expensive—we have tens of thousands of aggregates, each with a different distribution. The controller would have to perform as many convolutions as there are links which pass the simulation step from Figure 6.1. At the same time, these convolutions must only take a few milliseconds to perform, or they will slow down the optimization process. A couple of tricks let LDR perform these convolutions quickly.

First, we only need to convolve links close to saturation. We can guarantee that both multiplexing tests will pass if the sum of the maximum traffic levels of all aggregates that cross a link does not exceed that link's capacity. These links can shortcut the entire loop in Figure 6.1. In normal network operation only a small subset of the network's links will be in this high-utilization mode [41].

Second, convolution in the time domain is equivalent to multiplication in the frequency domain. LDR does not perform convolution directly. Rather, it transfers data to the frequency domain using a fast Fourier transform (FFT), multiplies the frequencies, then inverts the FFT to get the convolved distribution. This algorithm is of complexity $N \log N$, where $N$ depends on the quantization applied to the discrete time-domain data. We found that quantization into 1024 levels per distribution yields acceptable performance. If the convolution becomes a bottleneck, all FFT/IFFT computations can be readily offloaded to a GPU.

Is measuring bandwidth every 100 ms frequent enough? Our intuition was that for flows with

RTTs in the tens of milliseconds, measuring an aggregate's demand every 100 ms should capture enough short-term demand variability for us to reserve sufficient capacity to ensure short queues. Results supporting this are given in Section 7.4. In shorter-RTT settings such as data centers, senders' rates would change more quickly, and we would need to poll counters far more often. LDR is not intended for data-center use where other approaches [2] are more appropriate.

## 6.3 Dealing With Unexpected Variability

Once an ingress router receives a new mapping of aggregates to paths, it will immediately install the relevant routes in its routing table, together with any multipath split ratios.

The same polled counter data that an ingress router passes to the controller also allows the controller to continuously monitor whether aggregates are behaving as the controller expects. It is impractical to install new paths and radically change the traffic placement for every small change in load. Under normal circumstances we re-optimize the network once a minute, and we do not wish to re-optimize too frequently, as shifts in traffic placement may cause packet reordering. Yet a minute is a long time to endure overload after traffic conditions change significantly.

In LDR the controller uses the counter data not only to perform periodic re-optimizations of the network, but also to detect and react to unexpected changes in traffic load. We now describe two mechanisms that are used in tandem by LDR to limit the effect of unexpected events like the one in Figure 3.16.

### 6.3.1 Triggered Optimization

Ideally, we would like to only trigger a new optimization pass when we know that a queue is starting to build somewhere within the network, otherwise we would be needlessly wasting network resources and causing churn. However, LDR does not directly measure queues of network devices—LDR only sees traffic counter values from ingress devices.

Moreover, even if LDR were to directly measure queues at ingress points, a single ingress does not have sufficient knowledge about other aggregates' demands to be able to single-handedly make the decision to trigger an optimization. Often a sudden increase in the aggregate's level will cause a queue within the core of the network, where the aggregate combines with other aggregates, to overflow, instead of one closer to the edge, where the aggregate enters the network.

As described in Section 6.2.2 the controller receives a continuous stream of per-aggregate byte counters from each ingress at a rate of 10 values per second per aggregate. At a small time interval (5 seconds by default) the controller checks to see if the current routing solution is likely to overflow queues in the network using the convolution-based mechanism from Section 6.2.4 and each aggregate's 600 most recent counter values—a minute's worth of history. In Section 7.4 we demonstrate that an aggregate's short-term variability from the current minute is a sufficient predictor for the following minute's short-term variability.

When the controller detects that an aggregate exhibits variability high enough to overflow a queue under the current routing solution, it reacts by triggering an optimization pass like the one described in Chapter 5. When the optimization completes a new routing solution is sent to devices in the network. To avoid this process becoming pathological, if any requests to trigger a new optimization pass are received while another one is in progress, they are ignored and a single optimization pass is scheduled after the current one completes.

### 6.3.2 Low-priority Marking

The controller may require significant time to re-balance routing after an unexpected event. Even ignoring round-trip times between the controller and network devices of the order of tens of milliseconds, it can take up to 5 seconds for the controller to detect the event and trigger an optimization. After the optimization pass is complete, new tag-switched paths may need to be installed throughout the network before the new routing solution is enabled. As we demonstrate in Section 7.3, the optimization itself can be expected to complete in under a second. More troubling is the time to install new routing state which depends on a wide variety of factors and can vary greatly depending on the network device [22].

During this process the offending aggregate may cause drops in the network. To prevent these drops from negatively affecting other aggregates, LDR's controller installs rules that mark specific allocation-exceeding aggregates' packets as low priority using diffserv codepoints. As soon as the controller detects that it needs to trigger an optimization, before it starts running the optimization, the controller sends the diffserv rules to the ingress routers of aggregates whose traffic caused the triggered optimization.

If the unpredictable aggregates share high-utilization links with other traffic it is very likely that queues form while the controller is running the triggered optimization. In this case the low-priority diffserv rules will cause unpredictable aggregates' traffic to be preferentially dropped, protecting other better-behaved aggregates. This mechanism helps reduce disruption in cases where a single aggregate "spikes" like in Figure 3.16. The ingress routers remove the diffserv rules as soon as they receive a new set of paths from the controller, after the triggered optimization is complete.

### 6.3.3 Triggered Optimization and Headroom

The mean-level estimation algorithm from Section 6.2.3 by default aims to allocate 10% headroom across all links in the network to allow the mean levels of aggregates to evolve minute to minute. The performance of the triggered optimization mechanism that we just discussed is closely related to the amount of headroom allocated—the more headroom there is, the less likely is that an unexpected event triggers an optimization. On the other hand, increasing the headroom target for the mean level estimation algorithm will cause some flows to be routed on longer paths, thus hurting propagation delay.

While we use a set of real-world traces in Section 7.4 to experimentally verify that a 10% headroom target provides a reasonable tradeoff between propagation delay and queuing delay caused by unexpected events, we acknowledge that under different traffic conditions either more or less headroom may be appropriate.

Luckily, the triggered optimization algorithm itself can be used by the network administrator to adjust the overall amount of headroom needed by the network. If LDR reports the number of optimizations triggered over a long period of time (*e.g, one hour*), the network administrator can

decide if the number is too high and increase the headroom target for the mean-level estimator. Inversely, if the operator wishes to reduce the delay experienced by flows in the network they can reduce headroom until the number of triggered optimizations becomes unacceptable.

## Chapter 7

# Evaluation

In this chapter we provide a detailed evaluation of LDR's behavior. We first examine the scenario from Figure 4.1a, as this illustrates how the delay of one aggregate trades off against another. Such toy examples aid understanding, but are rather unrealistic, so we then evaluate LDR's "static" parts—its optimizer and its mean-level prediction algorithm—on a large number of real-world topologies and traffic matrices. We finally use large-scale simulation to evaluate the system as a whole, including its "dynamic" components—the convolution-based short term variability estimation algorithm and the triggered optimizations.

Alongside LDR we also evaluate a variant of our system that does not measure flow counts, thus weighing all aggregates equally, labeled LDR NFC.

One problem we face is in choosing a control experiment. It would be simple to compare against shortest-path routing, but for all scenarios where an active load-dependent routing system provides any benefit, shortest-path routing cannot fit the traffic. ISPs tune link-state metrics to avoid congestion, and when this fails, often manually tweak label-switched paths. As we do not know how they do this, we cannot effectively compare with current practice. Instead we will compare with state-of-the-art active systems from the research literature.

As explained in Chapter 2, TeXCP [47] and MATE [29] perform *MinMax optimization* over a restricted set of pre-selected paths. While the published work on TeXCP suggests pre-selecting the ten shortest paths, we find that doing so often results in a solution that cannot fit the offered load in many of our experiments (e.g., as in Figure 3.9d). MinMax solutions are also not unique in many cases.

To provide the best possible comparison, as in Section 3.2.2, we evaluate against two variants of MinMax. The first, labelled MinMaxK10, uses the ten shortest paths, as with TeXCP. The second variant, labelled simply MinMax, can use all paths so it is optimal in relieving congestion. We do this by replacing LDR's optimization function with the MinMax function in our linear optimizer. In both cases, to solve the non-unique problem, we tie-break in favor of minimizing total delay. We also evaluate against B4 [45], which as we previously discussed, performs greedy routing.

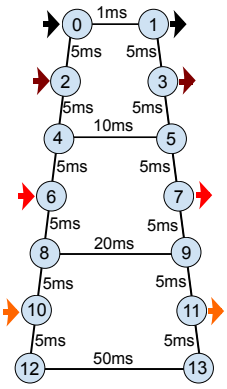



**Figure 7.1:** Ladder topology

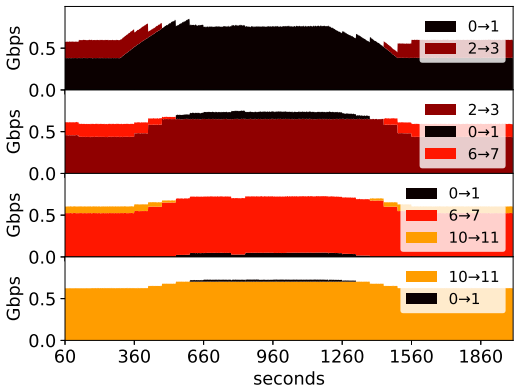



**Figure 7.2:** Ladder topology MinMax / MinMaxK10

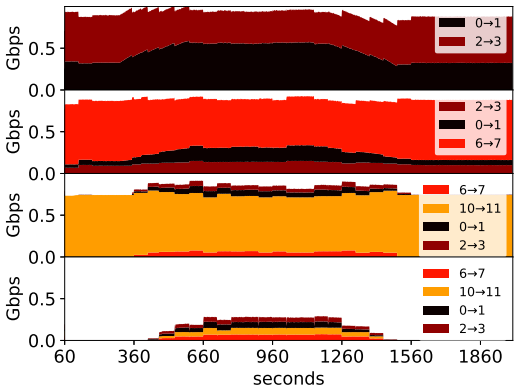



**Figure 7.3:** Ladder topology B4

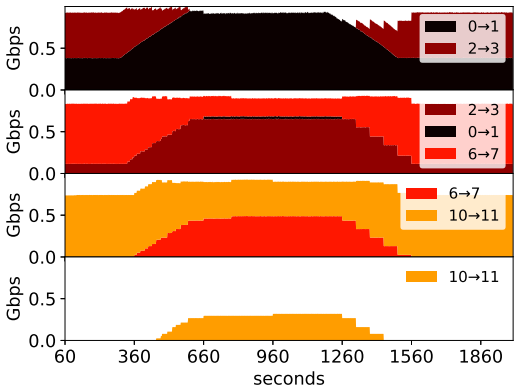



**Figure 7.4:** Ladder topology LDR

## 7.1 The Impact of Latency on Path Selection

We begin with a small example meant to give the reader an intuition for the high-level behavior of the routing algorithms under evaluation. We explore the behavior of the schemes on the topology in Figure 4.1a, which we reproduce in Figure 7.1 for ease of explanation. We place the four aggregates shown by the colored arrows. The four middle links have capacities of 1 Gbps; all others are 10 Gbps. Each aggregate carries ten sessions that make back-to-back 50 KByte TCP transfers emulating web transfers and 500 long-lived TCP flows as background traffic. Initially, all four aggregates are each limited by external access links (not shown in Figure 7.1) to 500 Mbps. We simulate the topology using the htsim packet-level simulator [40].

B4 uses a *greedy strategy* that places traffic on shortest paths first, so it should give low delay. However, it cannot be used in ISPs as it assumes a priori knowledge of flows' demands. For this experiment we extend B4 with LDR's mechanism for predicting aggregates' demands and variability (Section 6.2), and treat an aggregate's priority as proportional to its estimated size.

At time 300s we start to increase the rate of the top (black) aggregate by increasing the access link speed of its long-lived bulk transfers. We keep increasing until time 600s, when the overall rate of the aggregate reaches 1.2 Gbps. At 1200s we start to decrease its rate until it reaches 500 Mbps

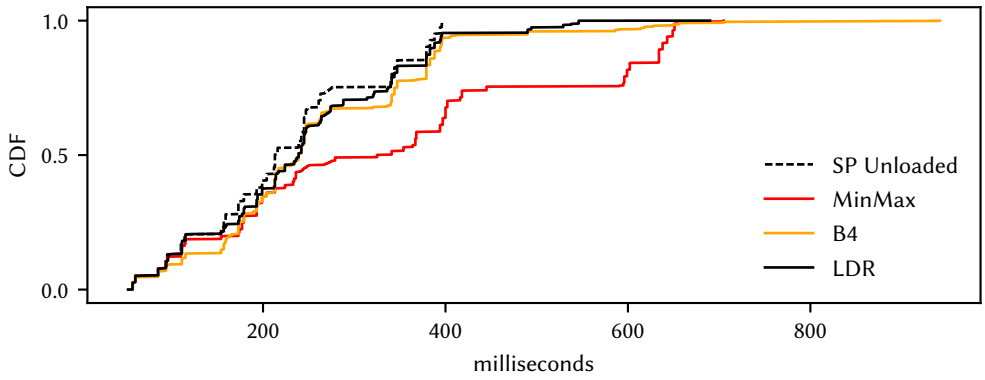



**Figure 7.5:** Ladder topology completion times

at 1500s. We examine how the each algorithm places the demand, and its impact on short-flow completion times.

Figures 7.2, 7.3, and 7.4 show how the different schemes split the aggregates' load. The four rows in each figure show the bandwidth on each of the four horizontal links in the network —the top row is link $0 \rightarrow 1$, the remaining rows show links $4 \rightarrow 5$, $8 \rightarrow 9$, and $12 \rightarrow 13$, respectively. Within each row, the colors show as a stacked graph how much traffic from each aggregate traverses that link.

MinMaxK10 and MinMax perform identically in this small test, as there is a very small number of paths and both schemes end up using the same set of paths; both spread traffic equally across the links, regardless of path latency, as this results in lower maximal link utilization. When the increase in $0 \rightarrow 1$'s demand occurs, the excess traffic spreads out with no attempt to keep traffic away from the long bottom path.

B4 initially puts as much traffic as it can on each aggregate's shortest path until utilization reaches 90%, then spreads out to longer paths. When the black aggregate reaches 1.2 Gbps, the top three core links are all filled by the greedy algorithm before any aggregate is fully allocated. This results in the longest link being used by traffic from all aggregates; some of the black aggregate sees 110 ms latency due to this circuitous routing.

LDR prefers to always utilize shorter paths, initially loading the top two links to 90%. As the rate of the black aggregate increases, LDR slowly shifts each aggregate one link down as needed to make space.

Figure 7.5 shows the completion times of the short web flows from this experiment. The SP Unloaded curve is a control experiment; it shows the short flow completion times when routed on their shortest path in the absence of other traffic. By both minimizing delay and avoiding congestion, LDR provides short completion times, close to those of the control experiment. MinMax does not primarily optimize for delay, so it gives the worst completion times. As expected, B4 does achieve short completion times, but some flows are given unnecessarily long paths (see tail of the

B4 curve in Figure 7.5). Note that a few flows do worse with LDR than with B4; there is no such thing as a free lunch—some flows have to do a little worse to avoid others getting very bad routing. All schemes achieve similar aggregate throughput, as can be seen from Figures 7.2–7.4.

## 7.2 Generating Traffic Matrices

Before we move on to more involved evaluation of the properties of LDR, we need to describe how the synthetic matrices used throughout this thesis were generated—much of the validity of the results presented in Section 3.2 and the ones in this chapter rests on their quality. As a starting point we used the gravity-based model from [68], which we briefly rehash here. In that model the volume of traffic ($T(n_i, n_j)$) for each of the $N(N-1)$ pairs in the network is obtained by:

$$T(n_i, n_j) = T \frac{T^{in}(n_i)}{\sum_k T^{in}(n_k)} \frac{T^{out}(n_j)}{\sum_k T^{out}(n_k)} \tag{7.1}$$

where $T$ is the total traffic in the network. The values $T^{in}(n_i)$ and $T^{out}(n_i)$ are drawn from an exponential distribution. This model has been shown to produce realistic traffic matrices, even though it uses only a single parameter—the mean value of the exponential distribution. While it represents a good starting point, we note that this simple approach exhibits a couple of significant drawbacks.

The first one is that there are no guarantees that the network will be able to fit the resulting traffic matrix. The level of saturation of the network depends on both the mean value of the exponential and the actual network topology. On one hand it may be that we pick a mean value that results in a traffic matrix that exceeds the maximum flow of the network—*i.e.,* one for which no routing scheme will ever be able to fit the demand. On the other hand, if we pick a mean value which is too low we will generate a traffic matrix that is trivial—*i.e.,* one for which every aggregate can be completely routed on its shortest path without causing congestion.

Ideally we would like to have better control over the network's load level. We take the approach of previous work [38], which suggests scaling the traffic matrix after generation in order to set the network's load to an arbitrary point between the two extremes described above. To do so we first generate a traffic matrix using a random exponential distribution with an arbitrary mean value. We then obtain the minimum maximal link utilization possible under any routing scheme by solving the theoretically optimal MinMax multi-commodity flow problem (described in Chapter 2). If this link utilization value is $u$, then we know that network is $1/u$ away from being saturated—*e.g.,* if $u = 0.3$ then we know that if we scale all demands by $1/0.3 = 3.3$ we will achieve a maximally loaded network. If instead we want a network which is *e.g.,* 70% saturated we need to scale aggregates by $\frac{1}{0.7u}$.

The second issue with the approach in Equation 7.1 is that it does not take into account geographic distance between ingress and egress pairs. In a lot of scenarios traffic matrices will exhibit a degree of geographic locality [71]—e.g., because big resource providers attempt to locate resources as close as possible to end users. As we would like to explore how LDR functions in those scenarios as well as the non-local ones, we optionally add a degree of locality to the matrix generated using Equation 7.1 while preserving its properties.

Crucially, when we add locality to an already generated traffic matrix we want to preserve the values of *both* incoming and outgoing traffic at each node from the original traffic matrix. We use the following LP:

$$\begin{aligned}
\text{minimize:} \quad & \sum_i \sum_j D_{i,j} B^{new}_{i,j} \\
\text{subject to:} \quad & B^{old}_{i,j} \max(\{0, 1-l\}) \leq B^{new}_{i,j} \leq B^{old}_{i,j}(1+l) && \forall i \in N, \forall j \in N \\
& \sum_i B^{old}_{i,j} = \sum_i B^{new}_{i,j} && \forall j \in N \qquad (7.2) \\
& \sum_j B^{old}_{i,j} = \sum_j B^{new}_{i,j} && \forall i \in N \qquad (7.3)
\end{aligned}$$

where $B^{new}_{i,i}$ is the traffic volume between nodes $i$ and $j$ in the new, localized, traffic matrix. The constant $B^{old}_{i,j}$ is the traffic volumes in the original matrix between nodes $i$ and $j$, the constant $D_{i,j}$ is the distance of the shortest path between nodes $i$ and $j$. The constant $l$ is a positive parameter which determines locality. The larger $l$ is the more freedom the optimizer has to change different aggregates' demands to minimize the total traffic volume per unit of geographic distance—*i.e.,* to make the traffic more local. If $l$ is 0 the optimizer will be forced to set all $B^{new}$ equal to $B^{old}$. If the parameter is 0.5 the optimizer will be free to move up to 50% of each aggregate's volume to another aggregate. It would seem that as soon as $l$ reaches 1 the resulting traffic matrix will only have a handful of large aggregates, as the optimizer will seemingly have the ability to move all of the volume of any aggregate to a more local alternative, but notice that the constraints 7.2 and 7.3 will force it to preserve the sums of incoming and outgoing traffic at each node, so the resulting matrix will never be too far off the original one.

In summary, the algorithm that we use in this thesis when generating a traffic matrix with a given load and locality is as follows:

1. Using some random seed, generate a traffic matrix using the gravity-based model from [68].
2. Add locality to the generated traffic matrix by solving the LP formulated above. If the locality value is 0, then this step is a no-op.
3. Compute the MinMax link utilization $u$ in the localized traffic matrix.
4. Scale the traffic matrix so that its load matches the desired one—*e.g.,* if the load we aim for is 70% of the maximal one, we will scale all aggregates' volumes by $\frac{1}{0.7U}$.

Notice that adding locality happens before scaling, as that ensures that the resulting traffic matrix has exactly the desired load factor. Except for the second step, the process is identical with the one recommended in [38].

To see how the addition of locality in the second step behaves in practice we examine three different traffic matrices generated with the same seed and the same load value, but with three

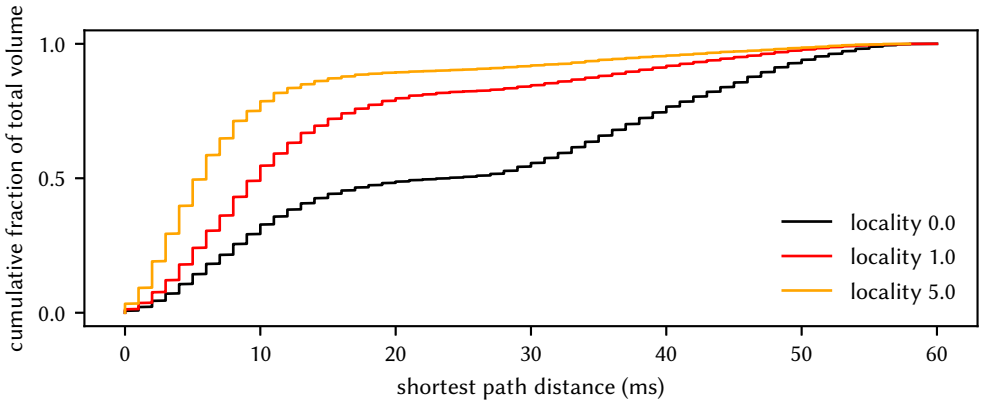



**Figure 7.6:** Cumulative fraction of total volume in Cogent's topology that travels a given shortest-path distance.

different values of locality—0, 1 and 5. The topology is that of Cogent—the largest one in the TopologyZoo dataset, and one with high LLPD (see Section 3.2). In Figure 7.6 we show the locality of traffic volume in each of the three traffic matrices. To generate the plot we sort all aggregates based on the length of their shortest path. Each point on the plot is a separate traffic aggregate; the $x$ value is the length of the aggregate's shortest path and the $y$ value is the cumulative fraction of the total traffic volume in the entire traffic matrix.

Cogent's topology contains large European and North American parts, connected by a handful of long-haul trans-oceanic links which account for the flattening of the LOCALITY 0 curve. Looking at the that curve, we can see that 50% of the traffic volume travels 20 ms or less—*i.e.,* about half of all traffic is between Europe and North America. Recent studies of Deutsche Telekom's network [71] suggest that in large ISPs this is not the case, but instead traffic is significantly more localized. As we increase locality we notice that less and less traffic is being moved between the two continents, loading the long-haul links less and less. At the extreme of LOCALITY 5 only about 10% of all traffic crosses between Europe and North America, with long-haul links being underutilized. We conjecture that this is also not a very realistic scenario. LOCALITY 1, which exhibits an 80/20 split between local and remote traffic, is probably closer to reality.

In Section 3.2 we focused on networks with locality of 1, but as we believe that networks are likely to exhibit a wide variety of locality values, in our evaluation of LDR in this chapter we do not limit ourselves to a single value but instead we explore a range of locality and load parameters.

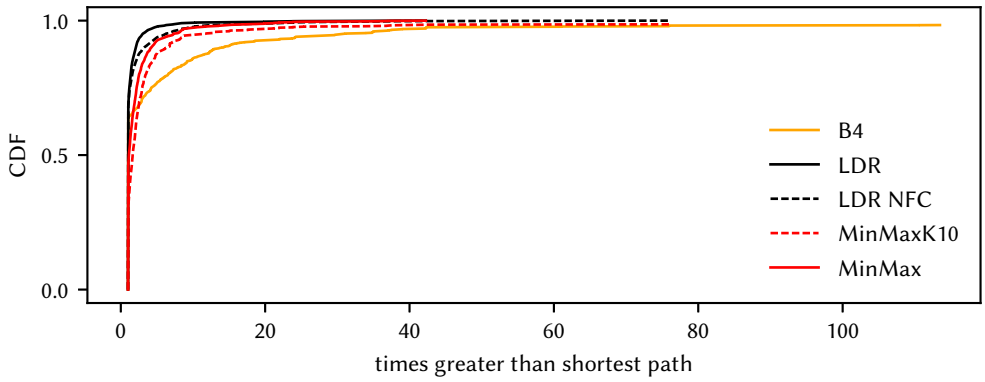



**Figure 7.7:** Maximum flow stretch; LLPD $< 0.5$; no headroom

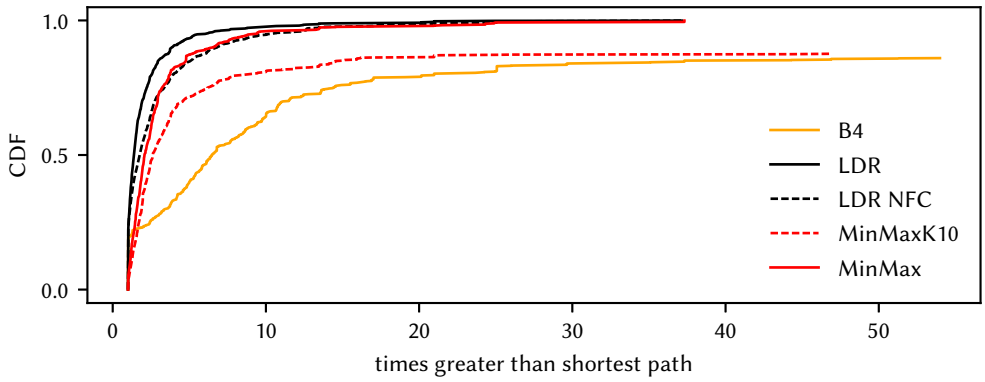
## 7.3 Static Components of LDR

In this section we revisit the landscape that we outlined in Section 3.2. We evaluate LDR's optimization mechanism and better understand the interactions between topology, traffic locality, load and latency. We seek to answer the following questions:

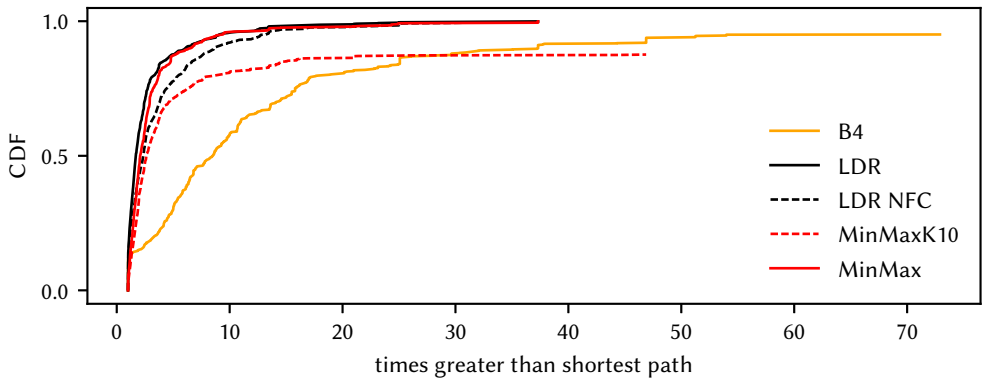
- Are networks with high LLPD truly harder to route? How does the path stretch achieved by LDR compare on those difficult networks with other schemes' path stretch?
- Does LDR consistently reduce latency compared with other routing schemes under traffic matrices across a range of load and locality parameters?
- How many paths on average does LDR need for each aggregate? How does this number compare to other schemes?
- Is the runtime of LDR's optimizer sufficiently low for real-time operation?
- How sub-optimal is LDR's optimization process? Does it matter in practice?
- Does vanilla LDR exhibit enough jitter and reordering to warrant the additional complexity of limiting the optimization by using the mechanism in Section 5.6? Does the mechanism work?
- Is LDR's prediction algorithm sufficient to predict minute-to-minute mean traffic levels in real-life backbone links?

### 7.3.1 Low vs. High LLPD

Generally speaking, the higher the LLPD, the harder it is to route a network using shortest path routing, but the more options a dynamic load-dependent routing system has to move load around. As we have seen in Section 3.2.2, with more options, heuristic algorithms such as B4 and MinMaxK10 are more likely to get stuck in a local minimum and fail to fit the traffic.




**(a)** no headroom




**(b)** 10% headroom

**Figure 7.8:** Maximum flow stretch; LLPD $> 0.5$; where the CDF fails to reach 1.0, this indicates that in the remaining scenarios the routing system could not find a placement that would fit all the traffic

Let us observe some measurements that illustrate the general observation above. First, let us examine the effects of LLPD and of headroom on latency stretch. Figures 7.7 and 7.8 show CDFs of the maximum path stretch for each traffic matrix under the same min-cut 77% load and locality 1 as Figure 3.9. The different curves illustrate the effects of LLPD and headroom. In Figure 7.7 we see the networks with LLPD less than 0.5. These networks have few low-latency alternate paths; for some topologies and traffic matrices the maximum stretch is very high—over 100x in the limit with B4. There is not much to choose between the different algorithms here, as the topologies do not provide many routing options. No headroom is reserved in Figure 7.7, but adding 10% headroom makes almost no difference to the CDF.

Figures 7.8a and 7.8b show the networks with high LLPD without headroom and with 10% headroom respectively. Where the CDF fails to reach 1.0, this indicates that in the remaining scenarios the routing system could not find a placement that would fit all the traffic. This happens with both B4 and MinMaxK10. LDR with headroom and MinMax give very similar maximum stretch; this is mostly due to our formulation of MinMax additionally optimizing for latency once

its utilization goal has been satisfied. Without this enhancement, MinMax can choose very long paths. As we saw in Figure 3.13, LDR with headroom and MinMax exhibit very different median latency stretch. B4 is the outlier—in most cases it routes some flows on paths 10x longer than their shortest path.

We must discuss headroom in the context of B4. B4 was designed for use on Google's network with controlled traffic sources. If we wish to use it for ISP networks, we will also need to add headroom. Our formulation for assessing traffic predictability and statistical multiplexing is quite general, so we can also apply it to B4 to calculate desired headroom. When we do so, we find that headroom interacts with B4 in an interesting manner. Consider again the GTS topology in Figure 3.10, where B4 became congested. If we allow, say, 10% headroom, B4 will stop short of saturating all the links on the first pass, and move on to placing some of the long distance traffic on the slightly longer path via G. If the traffic from G→V that B4 failed to place can fit within the reserved headroom, then it can still be routed after all other traffic has been placed. This is the reason when headroom is added in Figure 7.8b, B4 can fit traffic in a wider range of scenarios, though these graphs do not capture the degree to which B4 eats into the supposedly reserved headroom to do so. B4 also pays a latency price when it does so.

### 7.3.2 Performance Under Varied Load and Locality

We observe that both traffic locality and the extent to which the traffic matrix is loaded play a large role. As we adjust those two parameters of our synthetic traffic matrices, latency stretch and the ability to fit traffic both change, though different routing schemes are affected differently. To capture these effects, on Figure 7.9 we explore maximum path stretch under three different locality values for a lightly loaded scenario where the network is loaded up to 61% of its maximum flow and a more heavily loaded one where it is loaded up to 77% of its maximum flow. The plot for load 77% and locality 1 is the same as the one in Figure 7.8a, but on a log scale. In all traffic matrices flow counts are distributed proportional to traffic volume.

Let us first examine how the systems behave when it is possible to fit the *entire* traffic matrix by routing each aggregate on its single shortest path. We will call such traffic matrices *trivial* as in those cases there is no need for traffic engineering, and legacy shortest path routing is enough to yield an optimal solution. Notice that more of the matrices in the datasets are trivial, yielding points with max stretch of 1.0 on Figure 7.9, as the traffic distribution becomes more local and less loaded. This is expected since in those cases a larger fraction of the bytes in the network have to only travel only a couple of hops between the source and the destination.

When the traffic matrix is trivial all non-MinMax routing solutions will perform the way single shortest path would. MinMax-based solutions will spread traffic as much as possible over the entire network (in the case of MinMax) or over the first K shortest paths (in the case of MinMaxK10). When TMs are more local and less loaded, the difference between MinMax-based and non-MinMax

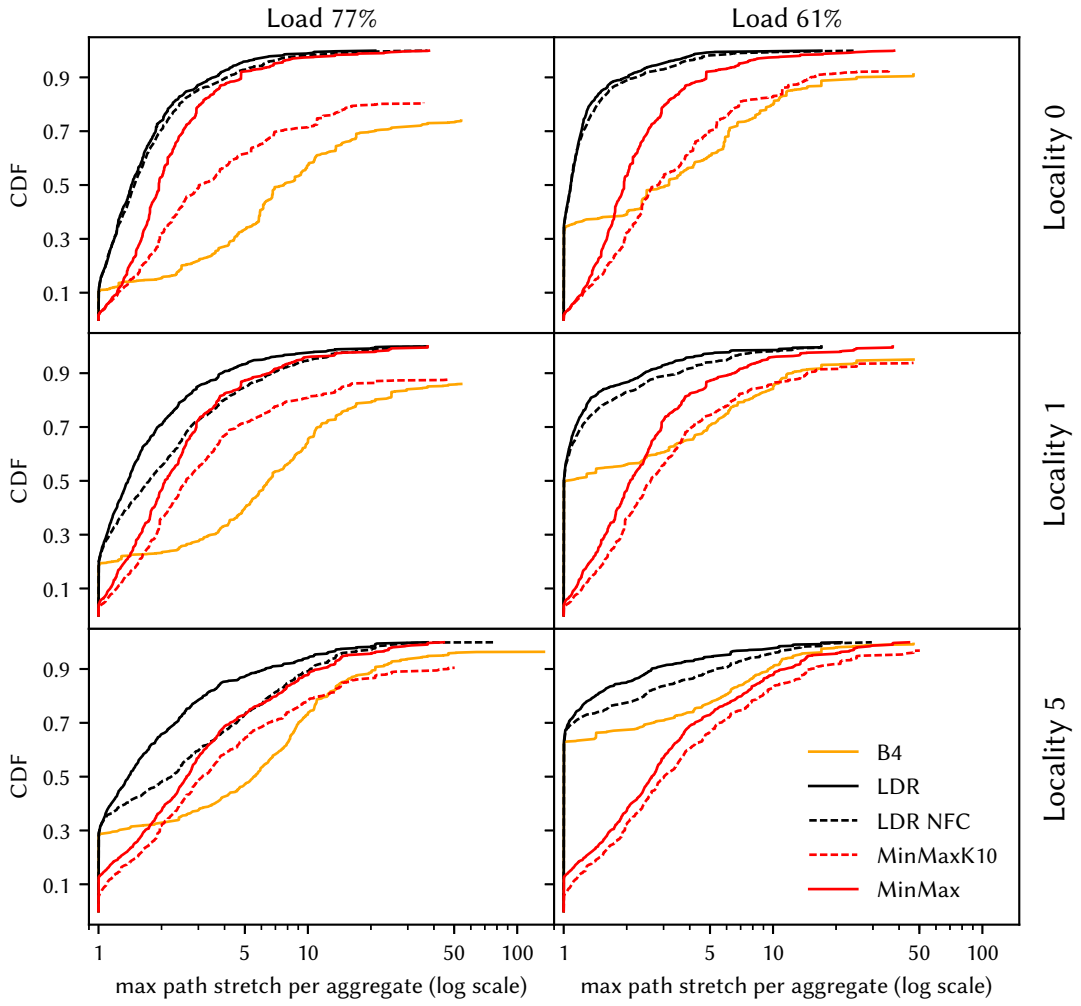



**Figure 7.9:** Maximum flow stretch under different load and locality values; LLPD > 0.5; no headroom; where the CDF fails to reach 1.0, this indicates that in the remaining scenarios the routing system could not find a placement that would fit all the traffic

based is more pronounced as a larger fraction of the traffic matrices are trivial.

While the optimizer-based solutions (LDR and MinMax) are guaranteed to fit traffic if possible, the other two may cause congestion even when a congestion-free solution exists. The less loaded and more local the traffic matrix is, the easier it is for a greedy heuristic to fit the traffic, because it is less likely that it gets stuck in the kind of local optima previously discussed in Section 3.2.2.

Figure 7.9 also lets us reason about the importance to LDR of taking each aggregate's flow count into account when running the optimization from Chapter 5. The gap between LDR and the version of LDR which does not take flow counts into account increases as locality grows, and is more pronounced if the network is more loaded. This effect occurs because in all traffic matrices flow counts are distributed proportional to traffic volume. As traffic gets more local a larger fraction of the TM's total volume is between geographically adjacent aggregates. In those cases the optimal solution would have to be skewed towards short-haul aggregates (as they carry a larger fraction of

bytes and have more flows). The version of LDR which weighs all aggregates the same cannot do that.

The more local and less loaded the traffic matrix, the more likely it is for the $k = 10$ version of MinMax to be a close approximation of the optimal MinMax solution. This is because when traffic is more local there are less long-distance aggregates and therefore less chance that a long-distance aggregate's shortest path shares links with multiple short-distance aggregates' shortest paths.

In general, the more local traffic is the higher the stretch of all solutions. This is especially true when the network is highly loaded, as then all solutions have to route on paths further from the shortest path. Less loaded and more local traffic matrices are easier to route for all schemes, but will tend to give higher path stretch when MinMax-based schemes are used. More loaded and less local traffic matrices, on the other hand, are harder to route with greedy heuristics and a lot of traffic matrices fail to fit the offered load under those schemes due to local optima. LDR provides consistently good performance, even when it does not take into account flow counts.

### 7.3.3 Fraction of Flows Routed on Shortest Path

Figures 7.8a and 7.9 are useful in understanding the behavior of different routing schemes, but they only focus on the flow with maximum stretch. What happens to the other flows in the network?

To answer this question we plot on Figure 7.10 CDFs of the fraction of flows that are routed on the shortest path for each traffic matrix. If this value is 1.0, then for that particular traffic matrix, the routing scheme chose to route all flows on their shortest path, behaving like single shortest path routing. Note that those trivial traffic matrices are the same traffic matrices that exhibit max stretch of 1.0 on Figure 7.9.

MinMax-based solutions perform surprisingly well on this metric, as they minimize propagation delay as a secondary goal. As those schemes spread traffic as much as possible, their performance is largely independent of the load of the network. As traffic gets more local all schemes route more and more flows on their shortest paths, with B4's behavior converging to that of LDR. Regardless of load or locality LDR performs very well, consistently routing a larger fraction of flows on their shortest path.

Interestingly, all median values are above 0.8, implying that regardless of the routing scheme, one can expect that 80% of the flows in the network are routed on their shortest path. If we focus on schemes that aim to minimize delay, this fraction grows to well above 90%. This is to be expected, as traffic matrices under realistic load are expected to be heavily influenced by the topology and vice versa—no operator purposefully designs their network so that most of flows will *not* go over the shortest path, regardless of the routing scheme they choose to use in their network.

Figures 7.9 and 7.10 lead us to conclude that in our experiments traffic engineering only makes a difference for 10-20% of the flows in the network. At the same time, as Figure 7.9 suggests, different traffic engineering schemes can significantly benefit or hurt the propagation delay of

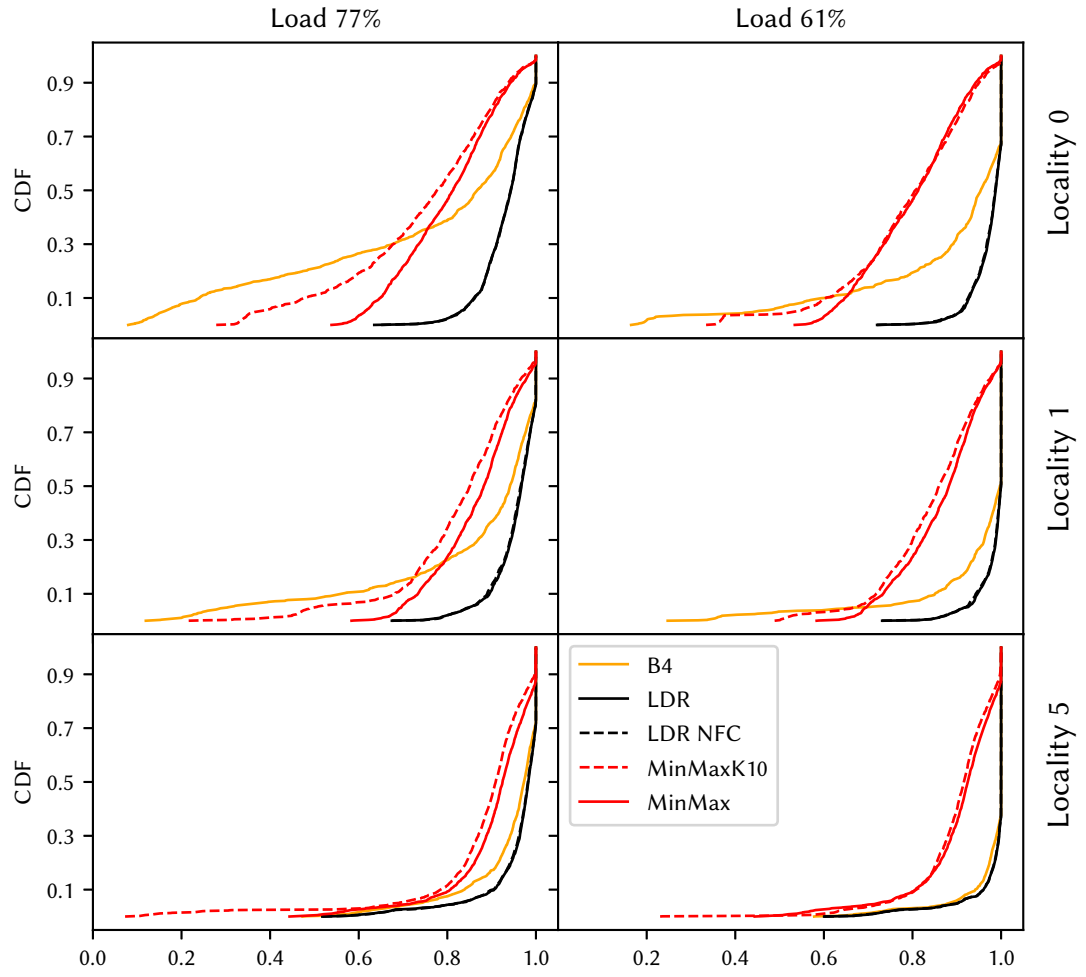



**Figure 7.10:** Fraction of flows that are routed on the shortest path under different load and locality values; LLPD $> 0.5$; no headroom

the flows for which they make a difference. Even more importantly, poorly performing traffic engineering schemes can significantly hurt the queuing delay in the entire network by causing congestion, as is evident from Figure 7.9.

### 7.3.4 Absolute delay

All evaluation presented so far deals with relative quantities, as they make it easier to compare a wide range of topologies and traffic matrices. To understand further what is happening, we need to examine absolute delays rather than relative ones, and to do this we need to look at individual topologies. We examine all topologies and traffic matrices with locality 1 and load of 77% of maximum flow—i.e., all the data points from from Figure 3.9. To focus on typical scenarios, from those data points we chose the two traffic matrices that give the median change in total delay when running B4 and MinMax.

We plot a CDF in Figure 7.11 of the absolute stretch $D_p - D_s$ across all flows, where $D_p$ is the

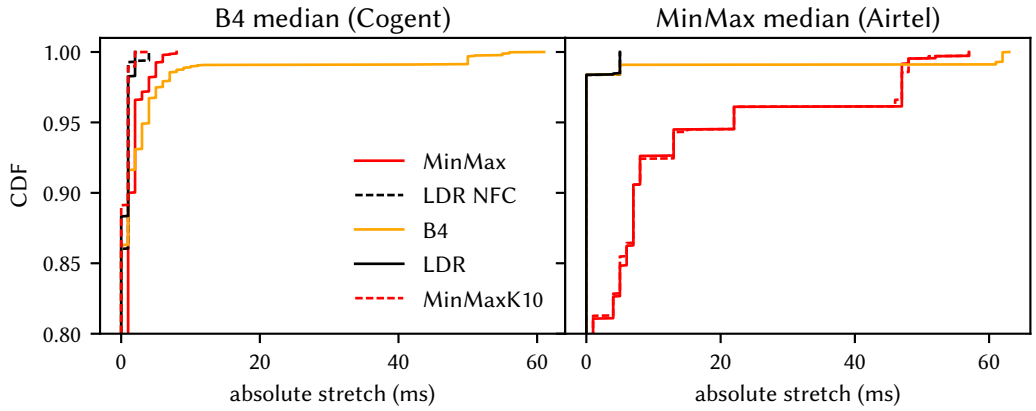



**Figure 7.11:** Absolute stretch in median topologies; only showing the top 20% of each distribution.

delay a flow experiences and $D_s$ is the delay of the shortest possible path the flow can be routed on. As expected from Figures 7.9 and 7.10 in both cases for about 80% of all flows it makes no difference what routing scheme is used, so we only plot the top 20% of each distribution.

The two topologies are very different. The topology giving median performance for B4 is Cogent. This topology is a good one for MinMax, giving similar absolute stretch to LDR. In contrast, the median topology for MinMax is Airtel. In this case MinMax yields significant absolute path stretch for almost 20% of flows. B4 performs similarly in both these scenarios: for 98% of flows it is a good heuristic approximation to LDR, but the remaining 2% of flows suffer the same sort of circuitous routing we see in the simplistic Ladder topology. In short, B4's greedy algorithm gets stuck in a local minimum and runs out of short paths to use; LDR avoids such problems.

### 7.3.5 Path Count

In this work we focus on routing systems that have the ability to split any given aggregate's traffic across multiple paths, if needed. Whereas a decade ago such systems were confined to academic research, recent advances in routing hardware and software-defied networking have made this functionality available in off-the-shelf network devices [32]. Nevertheless, there are still disadvantages to using multiple paths for a single aggregate.

Obviously, each additional path consumes network resources. In a tag-switched system like LDR each new network path will need up to as many rules as it has hops. Luckily, the sizes of devices' forwarding tables are continuously expanding and the added state is much less of a concern today than it was in the past, as large-scale deployments of other routing systems demonstrate [45].

A more fundamental disadvantage arises from the fact that in their quest to avoid reordering, as explained in Chapter 2, devices do not split traffic on a per-packet basis, but on a per-flow one. Contrary to this behavior, in our system design and evaluation we have so far assumed that each device has the ability to split an aggregate's traffic perfectly given any arbitrary split by the controller. Is this a reasonable assumption?

As described in Section 6.2.4, LDR's convolution-based short-term variability estimation mechanism uses samples from each aggregate's level as seen by its ingress. All flows that form an aggregate will, naturally, not have the same level of variability. If the ingress splits an aggregate across multiple paths and a larger fraction of the more-variable flows end up on one of the paths, there is no way for the system to detect and react to the potential transient congestion caused by those more-variable flows. Is this a design fault? Does LDR need a mechanism which looks at variability *per-path* instead of per-aggregate?

To answer these questions we examine the number of paths that LDR and each of the other schemes use when generating forwarding state for the traffic matrices from Figure 7.9. In Figure 7.12 we present the same data as in Figure 7.9, but instead of on maximum path stretch we focus on maximum number of paths. On the plot there is one point per traffic matrix. Each point represents the number of paths used by the aggregate that uses the most paths in the traffic matrix.

Clearly the optimizer-based schemes are very sparing in their use of network resources. Even MinMax, which spreads traffic in order to minimize utilization, uses up to 2 paths per aggregate in about 80% of all cases. This is because our implementation of MinMax, as well as others [47], minimizes latency as a secondary objective to link utilization. In practice this means that after achieving minimal link utilization in scenarios like the one in Figure 2.6 MinMax will prefer to concentrate traffic on low-latency paths, yielding lower path count. Unlike MinMax, LDR does not minimize link utilization, but always minimizes latency, which results in even less network state with about half of traffic matrices using only one path per aggregate (but not necessarily the shortest path) in lower-load and higher-locality scenarios.

B4, on the other hand, will greedily fill up each aggregate's shortest path and then spill traffic

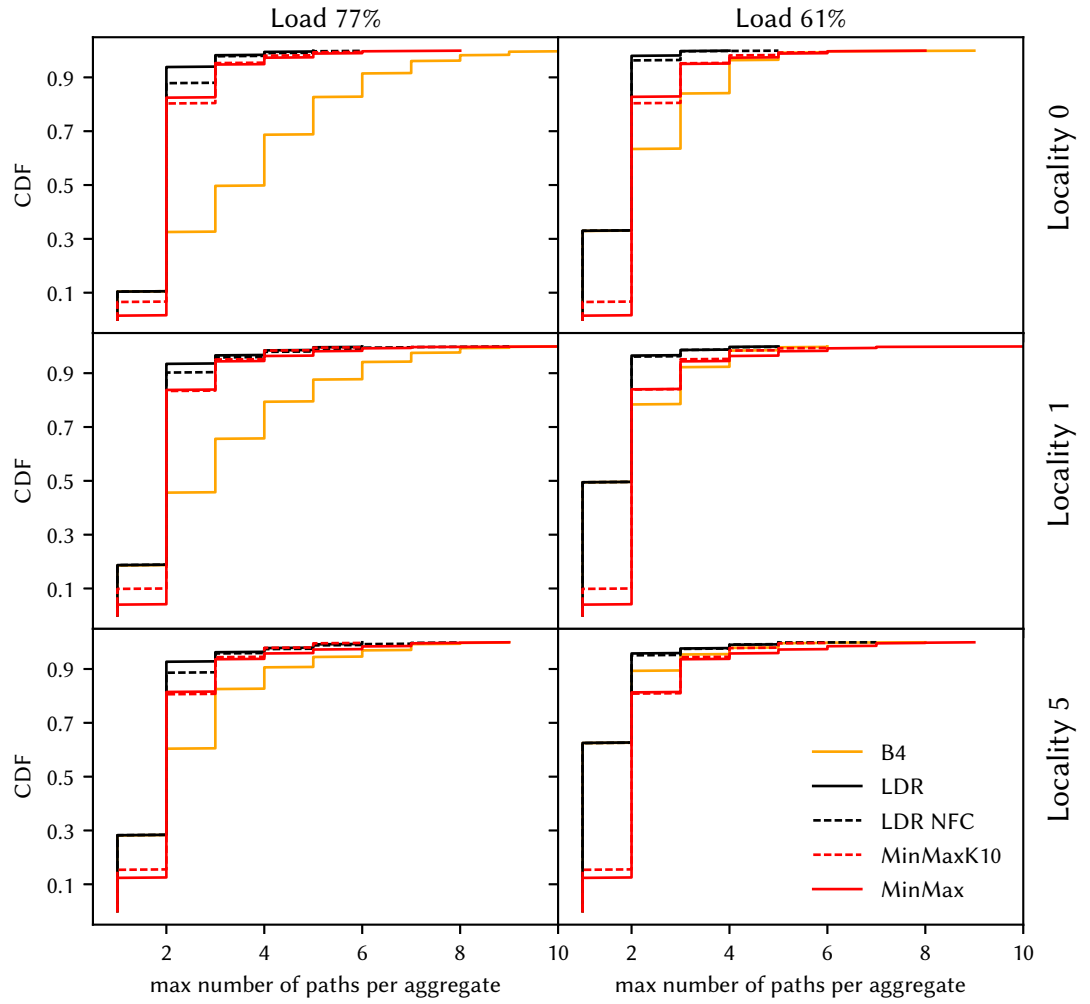



**Figure 7.12:** Maximum path count under different load and locality values; LLPD > 0.5; no headroom

onto multiple longer paths as needed. This results in a significant amount of network state. As topologies get more local and less loaded, they become easier to route for B4 and its behavior converges to that of LDR.

Figure 7.12 only shows the extreme case—the aggregate that uses the most paths. To examine how many paths other aggregates have, we plot on Figure 7.13 the fraction of aggregates that have only one path for each of the cases in Figure 7.12. Remarkably, in all cases both MinMax and LDR route a very large fraction of aggregates on a single path. These results, in combination with the ones from Figure 7.12, lead us to conclude that both schemes are very sparing in how they split traffic. In most cases, regardless of load or locality, more than 95% of aggregates experience no splitting at all, as they are routed on a single path. The remainder of the aggregates are likely to use only up to 2 paths, in the case of LDR, or up to 3 paths, in case of MinMax. As LDR routes its aggregates predominantly on a single path, we do not expect it to be affected by the negative effects of splitting. We further verify this claim using large-scale simulation results in Section 7.4.

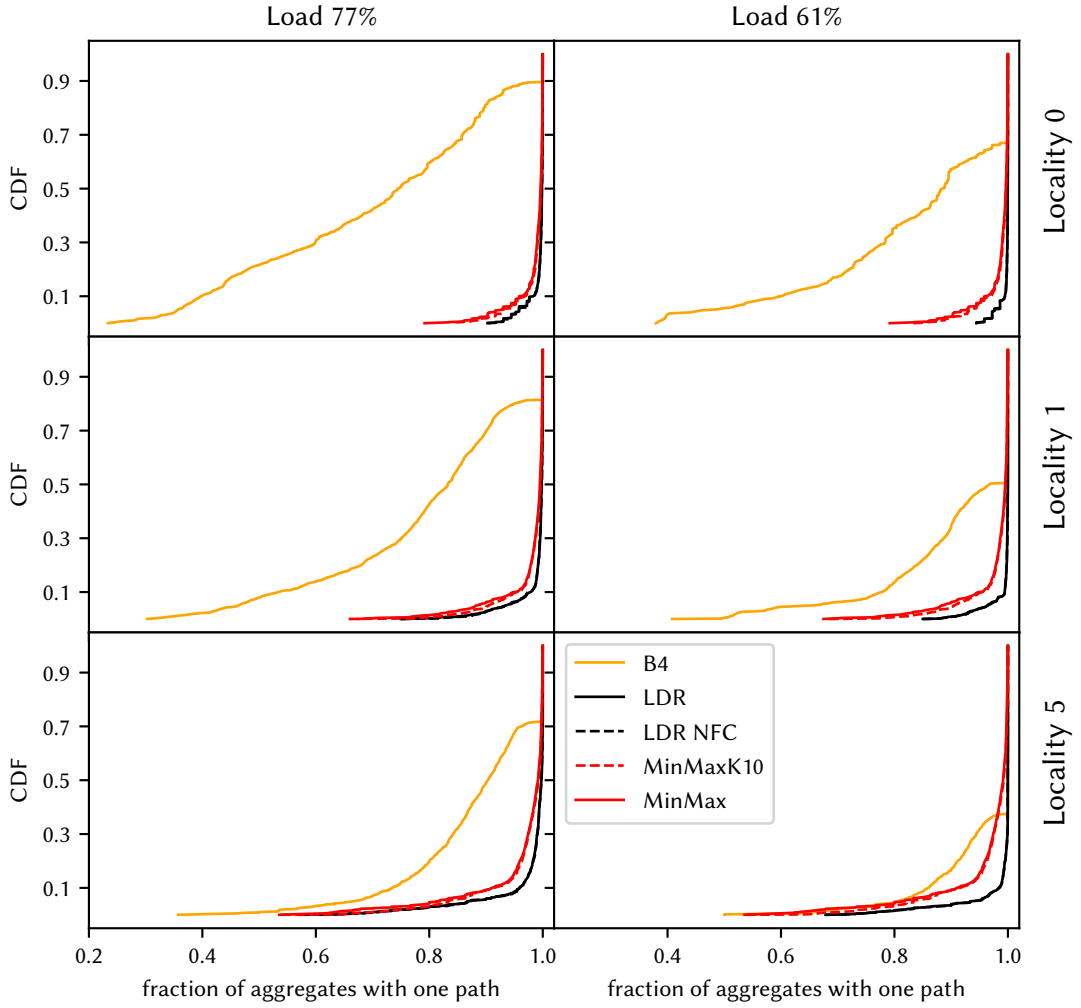



**Figure 7.13:** Fraction of aggregates that have only one path under different load and locality values; LLPD > 0.5; no headroom

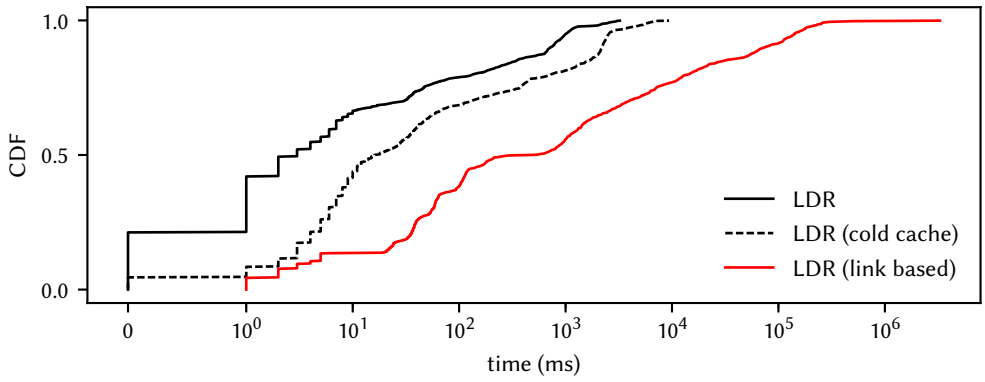



**Figure 7.14:** Runtime of optimization algorithms. Each point is the runtime of running LDR with and without $k$ shortest paths caching on a traffic matrix from the set of results that are shown in Figures 7.8 to 7.13. We also present the runtime of a traditional link-based multi-commodity flow formulation.

### 7.3.6 Runtime

To be practical, LDR's optimization algorithm must be able to calculate paths quickly on large complex networks. Figure 7.14 shows CDFs of the runtime of the LDR algorithm on the networks with LLPD greater than 0.5; these are the hardest to route. Each point on Figure 7.14 is the runtime of running LDR on a traffic matrix from the set of results that are shown in Figures 7.8 to 7.13.

The LDR curve includes caching of the $k$ shortest paths, whereas the COLD CACHE curve shows the first run, before the cache is populated. The difference between those two curves represents the overhead of running the $k$ shortest paths algorithm. In normal operation the $k$ shortest paths for each aggregate are cached, so periodic invocations of the optimizer will not incur this cost. In the case of a network failure some or all of the path cache would have to be reinitialized, so in those cases LDR's performance will lie between the COLD CACHE and LDR curves.

For comparison, we implement a traditional link-based multi-commodity flow formulation of the same objective as LDR and we use an off-the-shelf industry-standard solver [21] to obtain a routing solution. The LINK-BASED curve shows that the link-based multi-commodity flow formulation is about two orders of magnitude slower. We conclude that the optimization approach in Chapter 5 is fast enough to use in online centralized routing systems.

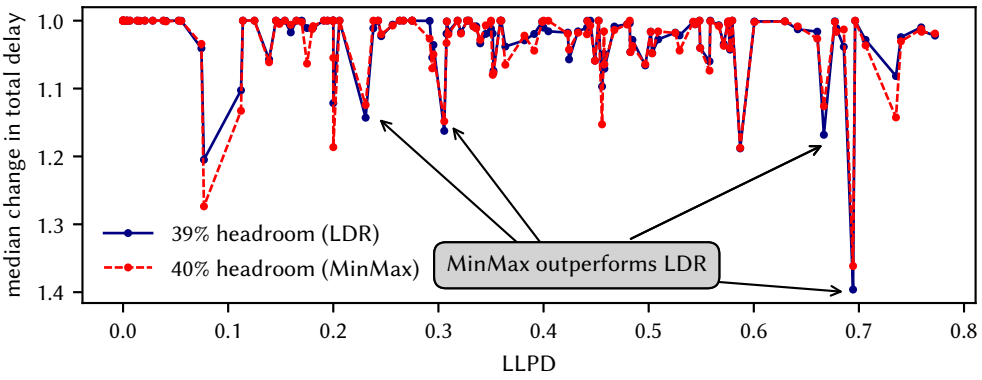



**Figure 7.15:** Sub-optimality with LDR. Plot shows median latency stretch at 39% and 40% headroom. The two curves are for LDR and MinMax, which is the optimal multi-commodity flow solution that minimizes link utilization.

### 7.3.7 Suboptimality of LDR

LDR's optimizer is only guaranteed to provide an optimal solution when the iterative path addition process described in Section 5.3 is able to progressively add the $k$ shortest paths for each aggregate until no path crosses any overloaded link. As explained in Section 5.4, when there are large numbers of high-volume aggregates LDR may skip over some of the $k$ shortest paths in the interest of limiting problem size and lowering runtime. How suboptimal is the result in practice?

To observe LDR under such extreme conditions, on Figure 7.15 we show the median latency stretch for each topology when the network is very close to saturation, when running LDR and MinMax. To generate the results we repeat the experiment from Figure 3.13, but this time instead of optimal routing we run LDR when the network is just 1% away from saturation (the curve labeled 39% headroom). We also run MinMax which minimizes the maximum link utilization and gives the optimal 40% headroom solution. Theoretically when we force 40% headroom both LDR and MinMax should converge to the same solution, as the network will then be completely saturated.

As the figure shows, however, there are topologies where MinMax actually *outperforms* LDR and yields better median path stretch. This is the effect of LDR's path addition heuristic which kicks in when the problem size becomes too large. The effect is clearly more pronounced in networks with higher LLPD, as they tend to have larger path diversity and it is more likely for LDR to experience the situation from Figure 5.4. In normal operation this is unlikely to be a problem as no operator will run their network at 99% of its theoretically maximum flow. In the case of failures which shift congestion within the network controlled by LDR, however, its heuristic may yield sub-optimal delay.

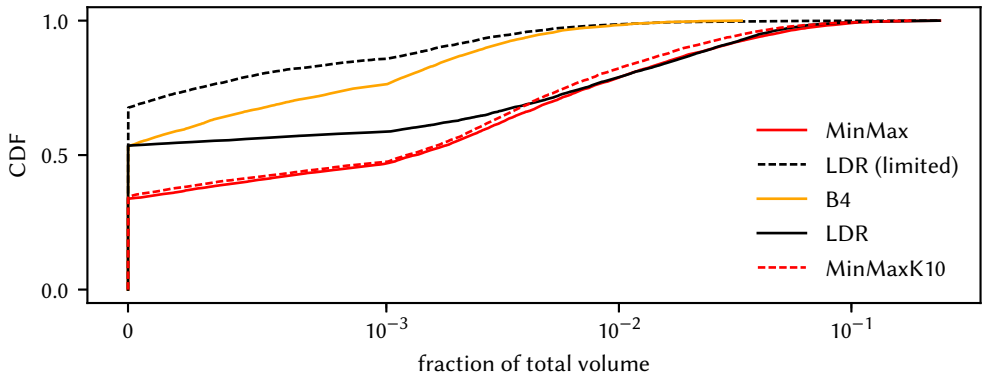



**Figure 7.16:** CDF of the fraction of total network volume that changed paths. Each point is a separate traffic matrix, the load of whose aggregates is randomly uniformly distributed +/- 5%.

### 7.3.8 Reordering and Jitter

To examine the extent to which LDR is likely to cause reordering and jitter we use the topologies and traffic matrices that were used in the experiments from Figure 7.7 and Figure 7.8. From each traffic matrix we generate a different traffic matrix where each aggregate's load and flow counts are randomly uniformly distributed +/- 5%. For each algorithm that we evaluate we perform a run on the original matrix and perform a run on the randomly permuted matrix. We are interested in how the algorithms' output for each of the random matrices changes with respect to the output for the respective original, or base, matrix. In addition to the schemes previously evaluated we also report a version of CTR that in parallel runs an optimization with the churn-limiting mechanism described in Section 5.6. We will label this version LDR (LIMITED). We omit the version of LDR which ignores flow counts, as in the results that we present in this section its performance is very close to that of LDR.

Keep in mind that in all experiments we set each aggregate's flow counts to be proportional to the aggregate's volume. Fractions of traffic volume are, therefore, also representative of fractions of flow counts.

The first metric that we examine is fraction of total volume that the output of each of the randomized runs shifts with respect to the output of its base run. This is the fraction of the network's traffic that can potentially experience jitter.

The output of each routing algorithm is per-aggregate sets of paths and, for each path, the fraction of the aggregate's volume that goes on it. Given each run's output and its base run's output we compute the fraction of each aggregate that remained on the same path ($f$). We then compute for each aggregate the fraction of total volume that changed paths

$$\frac{\sum_a V_a(1-f_a)}{\sum_a V_a}$$

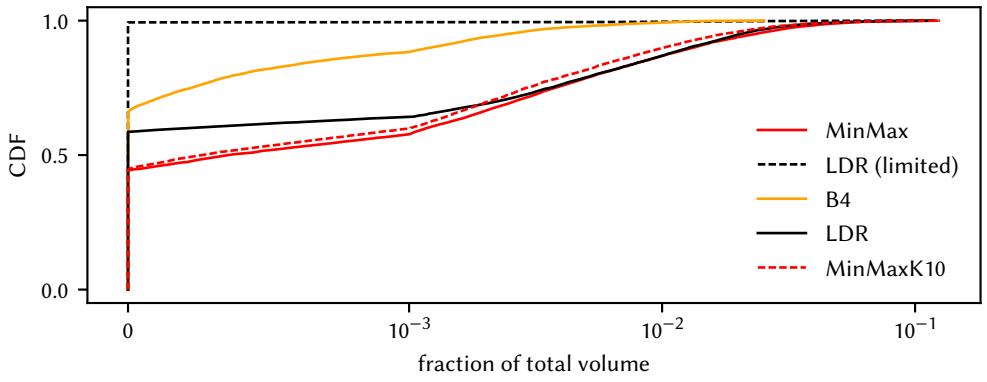



**Figure 7.17:** CDF of the fraction of total network volume that moved to shorter paths.

where $V_a$ is the volume of aggregate $a$.

Figure 7.16 shows a CDF of this quantity, where each point is a separate run. It is obvious that in some cases vanilla LDR causes a lot of jitter—the relatively small 5% changes to the optimization input cause a large fraction of the total volume (and therefore flows) to change paths. B4, on the other hand, is very stable because it stripes traffic, instead of packing it like LDR does. The limited version of LDR does significantly better, even outperforming B4 in some cases. This is because it moves no traffic when an aggregates' load decreases; it does not try to improve total per-flow delay by using the newly available capacity. The tail of LDR (LIMITED) is significantly heavier than that of B4. This is to be expected, as there are cases where the more-constrained optimization will either be unable to fit the demand, or will produce a solution which is not good enough (further than 1% of the optimal one). In those cases LDR (LIMITED) will fall back to using the same optimal solution that LDR uses, causing more change.

Moving flows to shorter paths is potentially more disruptive than moving them to longer paths, as it causes reordering. In Figure 7.17 we only show the fraction of volume from Figure 7.16 that moves to shorter paths. As LDR (LIMITED) is designed to avoid moves to shorter paths, in all cases where the limits are successfully applied (about 99% of all cases) there are no flows moved to shorter paths, and thus it incurs no reordering in those cases.

While jitter and reordering are two metrics that concern end users, network operators are also interested in minimizing the amount of processing that happens at network devices. In Figure 7.18 we show CDFs of the number of paths that needed to be updated for each randomized run. Because B4 stripes traffic, when an aggregate's volume changes it needs to change by a small fraction the splits of all aggregates that share busy links with the changed aggregate. LDR, on the other hand, performs larger changes, but they affect a smaller fraction of the paths. As expected, applying additional limits further reduces the number of paths that see updates for LDR.

The previous graphs indicate that applying extra limits does indeed have the potential to re-

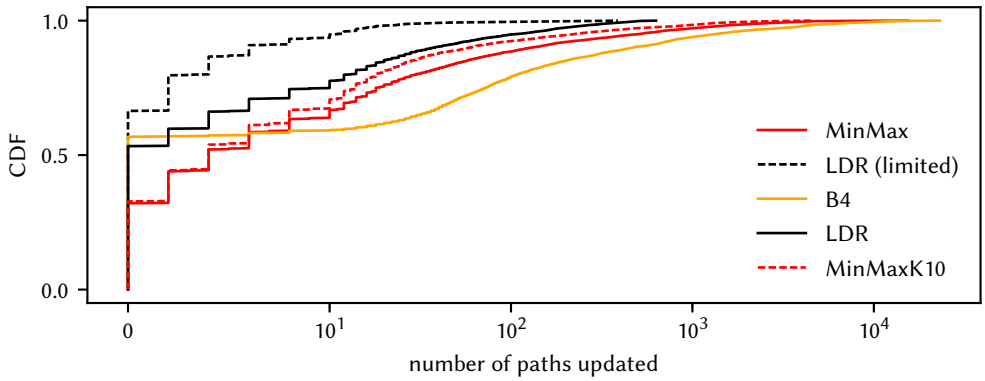



**Figure 7.18:** CDF of the total number of paths updated.

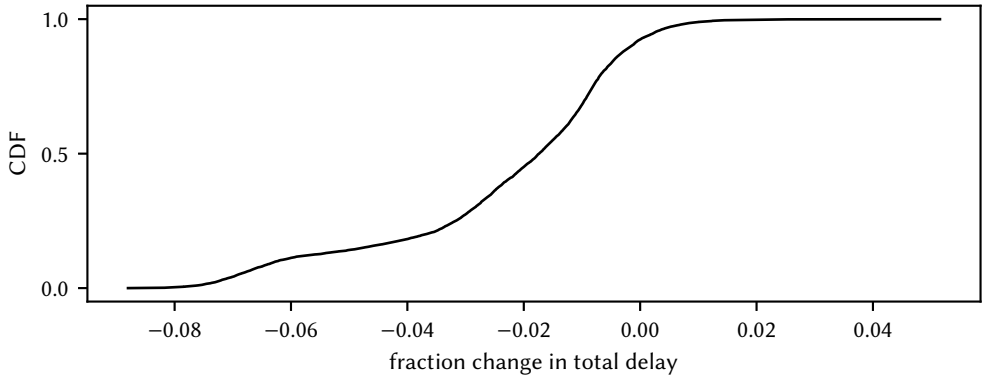



**Figure 7.19:** CDF of max single-aggregate volume change

duce the reordering exhibited by LDR, but what is the cost to total per-flow delay? On Figure 7.19 we plot a CDF with a point for each randomized run's percentage change of total per-flow delay when compared with its base run. In about 90% of the cases the increase in delay is within the 1% limit set by the algorithm in Section 5.6. In the rest of the cases the randomly perturbed matrix cannot be satisfied with limited optimization and the solution of the regular, unlimited, one is used.

### 7.3.9 Prediction Algorithm

Any active routing system that aims to minimize delay while avoiding congestion in an ISP's backbone inherently makes the assumption that traffic must be predictable enough that an allocation of traffic to paths can be maintained for a reasonable amount of time. The data that we presented in Section 3.3.2 suggests that variability is predictable on sub-second timescales, and the mean traffic level usually does not vary by more than 10% minute to minute. There are, however, cases where assuming that next minute's traffic level is 10% more than previous minute's traffic level will fail as a prediction algorithm. How does LDR's prediction algorithm perform in those cases?

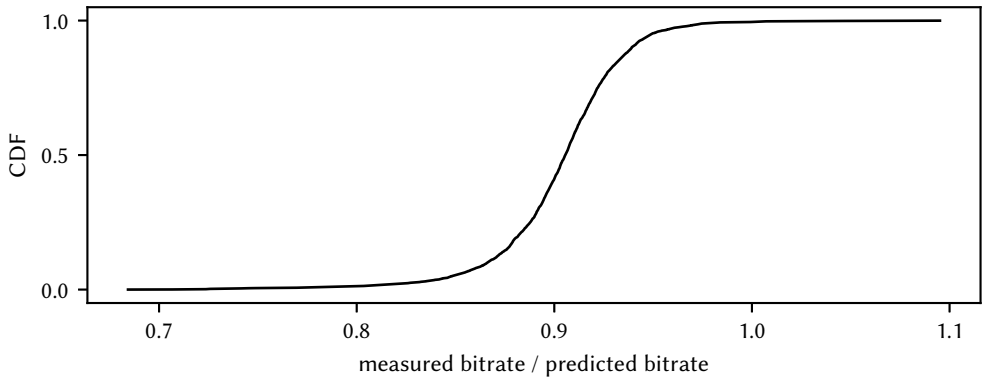



**Figure 7.20:** Predictions of mean traffic level (Tier-1 ISP)

We analyzed CAIDA packet traces from 2013 to 2016 of four 10 Gbps links within a U.S. Tier-1 ISP's backbone [15]. For each link we have 40 one-hour traces. We compute the mean traffic level for each minute, and apply Algorithm 1 to predict the mean rate in the next minute. Figure 7.20 shows a CDF across all the CAIDA traces of measured mean bitrate in the next minute divided by predicted bitrate. If the traffic were constant, all values would be 0.9. The traffic is very predictable on minute-to-minute timescales: only 0.5% of the time does the actual traffic exceed the prediction, and then never by more than 10%. The traffic in these traces is typically in the 1Gbps to 3Gbps range. When several such aggregates are statistically multiplexed to fill a 10Gbps or 100Gbps link, it is unlikely they will all exceed their predicted values simultaneously.

Reserving 10% headroom works well in the traces we have examined, but we envisage this should be configured depending on traffic dynamics and on how averse an ISP is to experiencing transient congestion. Larger values will further reduce the probability of congestion, but will route traffic on longer paths, increasing latency.

## 7.4 Short and Long-Term Variability in Demand

In this section we will evaluate LDR's short and long-term variability detection and prediction techniques described in Chapter 6. We will answer the following questions:

- Is the convolution-based short-term variability detection technique effective in helping LDR position aggregates in a way that both avoids congestion and loads links on low-delay paths to a high utilization? Is the computational cost of convolving aggregates prohibitive?
- Is it possible to predict the following minute's short-term variability based on recorded per-aggregate counters from the previous minute (as described in Section 6.2.4)?
- Is triggered optimization needed? Is it effective in dealing with unexpected changes in traffic level? What is its cost in terms of network updates?
- Is the 10% headroom target of the long-term prediction algorithm in Section 6.2.3 enough to give time to triggered optimization to react when there is a sudden change of traffic level.

To properly answer these question we need to evaluate a full-scale ISP network with a packet-granularity traffic matrix consisting of real-world flows. No such detailed traffic matrix exists, but we can synthesize one using data from the CAIDA traces.

We focus on GTS's central European network, presented in Figure 3.7, which we demonstrated to be challenging to route in Section 3.2.2. We randomly generate a traffic matrix for it with the same min-cut 77% load and locality 1 as Figure 3.9, which are also the same load and locality parameters as in Figures 7.7 and 7.8. We wish to generate aggregates whose mean rate matches this target traffic matrix but which exhibit real-world millisecond-granularity variability. To do so for each of the 11744 aggregates in the traffic matrix we pick one of the real-world packet-level CAIDA traces and we scale the trace so that the mean level of the first minute matches the aggregate's demand in the traffic matrix. We replay all these traces simultaneously, totaling approximately 134 Gbps of offered load. As there are many more aggregates than traces, to avoid synchronization effects, we start replaying each aggregate's trace at a random offset from its start.

For each minute after the first one LDR will update the routes used by our simulated aggregates, either periodically, or as needed by its triggered optimization mechanism. For the first minute, however, we bootstrap the simulation with an "ideal" initial routing configuration, which is produced by LDR, given foreknowledge of both each aggregate's first-minute mean level and each aggregate's first-minute traffic counters.

### 7.4.1 Performance of the Convolution Algorithm

By examining this first minute we can evaluate the effectiveness of convolution in isolation, assuming LDR's long-term mean level estimator from Section 6.2.3 works perfectly and the convolution mechanism sees exactly the data it needs in order to determine variability.

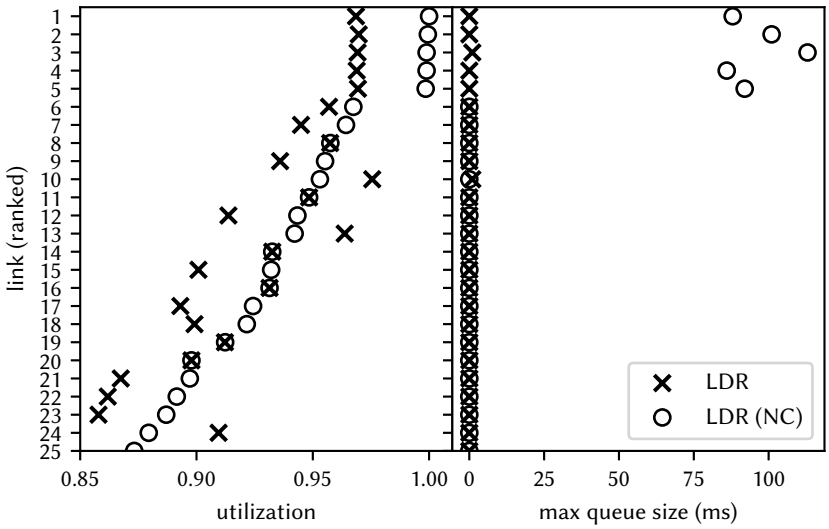



**Figure 7.21:** Perfect next-minute mean level prediction; convolution algorithm has access to current traffic counters. Mean link utilization (left) and maximum queue size (right) for the first minute. Links are ranked based on the utilization of LDR (NC).

Figure 7.21, shows link utilization (left plot) and maximum queue size (right plot) for the 25 most utilized links during the first minute of simulated time. In addition to LDR, we present results for a version of LDR which does not use the convolution mechanism to scale up aggregates as needed, but instead always optimizes at the mean traffic level (*i.e.,* it does not go through the loop in Figure 6.1). We label this version LDR (NC). On the $y$ axis in Figure 7.21 we rank the top 25 links based on their average utilization under LDR (NC). On the $x$ axes we plot mean link utilization over the minute and maximum queue size in milliseconds, as observed by packets that enter the link's queue.

At the top part of the left plot we can see that the top 5 links are loaded at, or very close to, 1.0 by LDR (NC). This is expected, since the mean level estimator is not used, but instead both schemes are bootstrapped with the true mean traffic levels from the first minute. LDR (NC) simply runs the optimization from Chapter 5 which packs a handful of link to saturation. Clearly this is not entirely desirable as those links experience significant transient queuing, as seen on the right-hand side plot. In our simulated environment queues can grow up to a second, but in reality only 50-100ms of queuing can be expected before drops occur. Unlike LDR (NC), LDR uses the convolution-based technique from Section 6.2.4 to avoid queues, while at the same time maintaining very high link utilization. The link utilization LDR achieves varies from link to link as different links are on the paths of aggregates with different levels of short-term variability.

Notice that in the low-delay solutions given by both LDR (NC) and LDR there are only a handful of heavily loaded links in the entire network, despite there being well above 10K aggregates. This is a realistic distribution of link utilization [41], which positively affects the performance of

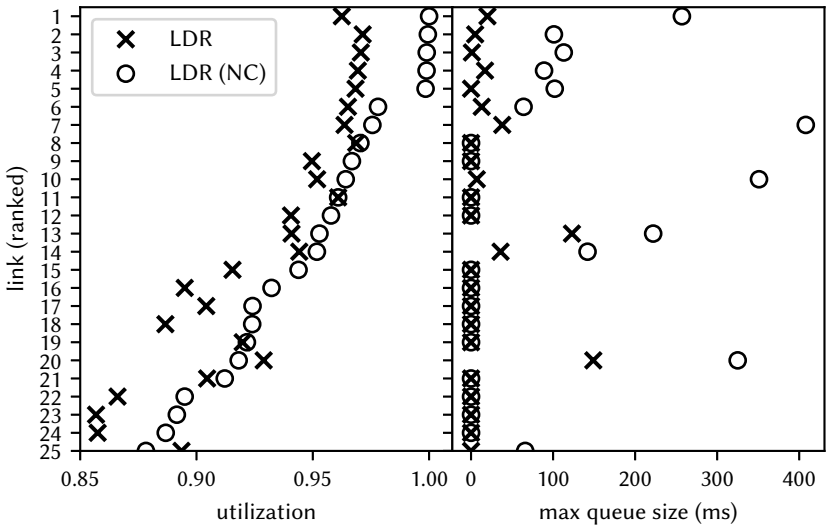



**Figure 7.22:** Perfect next-minute mean level prediction; convolution algorithm uses previous minute's traffic to assess short-term variability. Mean link utilization (left) and max queue size (right) for the first three minutes. Links are ranked based on the utilization of LDR (NC).

running the convolution-based mechanism—for most links the loop in Figure 6.1 can be avoided altogether as it is trivial to determine if traffic fits on a link with low utilization. In all results presented in this section the time to apply the convolution-based algorithm to the entire network was below one second.

### 7.4.2 Predictability of Short-term Variability

Figure 7.21 demonstrates that, for the first minute, LDR, with the aid of the convolution technique, is able to both avoid congestion and saturate desirable low-delay links. In reality, however, LDR will not have the luxury of operating on current information, but will always have to use the previous minute's measurements to compute routing for the next minute-long period.

Crucially, then, LDR's ability to maintain low queuing depends on how predictable sub-second variability is minute-to-minute. Previously, in Section 3.3.2, we presented results which suggest that short-term variability is predictable. To verify that this is indeed the case we advance the simulation past the first minute—for every subsequent minute LDR's convolution mechanism will use the previous minute's measurements when estimating short-term variability. To separate the effects of short-term variability from those of long-term variability, we keep the mean-level prediction mechanism from Section 6.2.3 disabled, just like during the first minute. Instead, every time LDR runs its optimization, we provide it with the accurate mean level for the following minute. For example when LDR's optimization runs at the end of minute 2 it will use the measurements from minute 2 to estimate variability, but scale those measurements according to the true mean aggregate levels from minute 3.

In Figure 7.22 we show average link utilization and maximum queue size from the first three

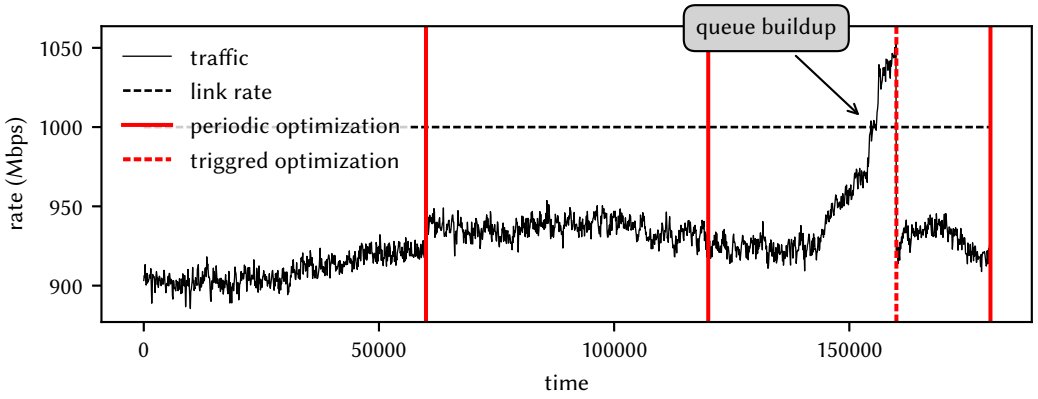



**Figure 7.23:** Traffic that crosses link rank 20 from Figure 7.22; time is in milliseconds; traffic is binned in 100 ms bins and each point is the mean of a bin. In this experiment LDR is given the exact mean traffic levels for the upcoming minute, but this knowledge of the future is of little use as the unexpected change happens mid-minute.

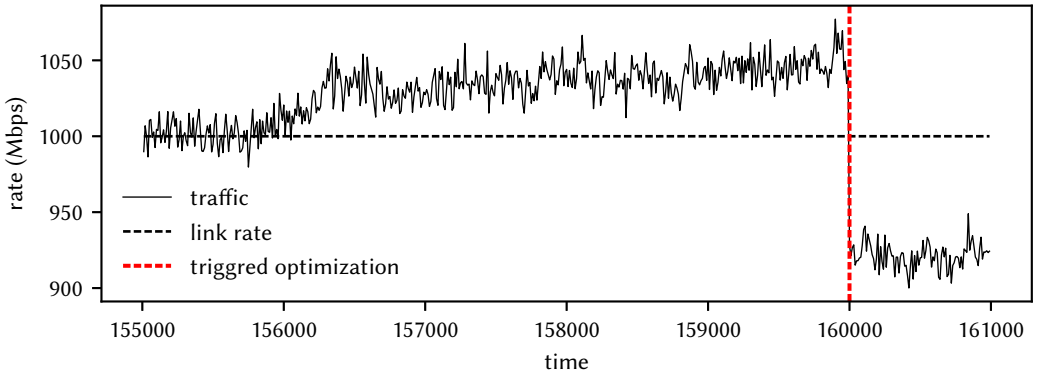



**Figure 7.24:** 155 sec to 161 sec zoomed in from Figure 7.23; time is in milliseconds; bin size is 10ms.

minutes of simulated time. The convolution mechanism definitely has a huge effect, dramatically lowering queue sizes across the board. In 21 of the top 25 links LDR's convolution mechanism is mostly able to maintain low queues, similarly to the results from the first minute above.

In links ranked 7,13,14 and 20, however, the maximum queue size exceeds 50 ms, even when the convolution mechanism is used. Does this imply that short-term variability is not constant minute-to-minute, or is there another factor in play?

In Figure 7.23 we investigate LDR's utilization of the link where the largest queue forms (ranked 20). On the $x$ axis we show simulation time in milliseconds and on the $y$ axis we display the rate of traffic *before* it enters the queue. The service rate of the queue is indicated by a horizontal line, and periodic and triggered optimization events are marked with vertical lines.

Even though the optimization from Chapter 5 aims to load this link to full capacity, the convolution process artificially increases the levels of aggregates that cross the link, providing just enough headroom so that the traffic does not cause queues to build up. This works for the first

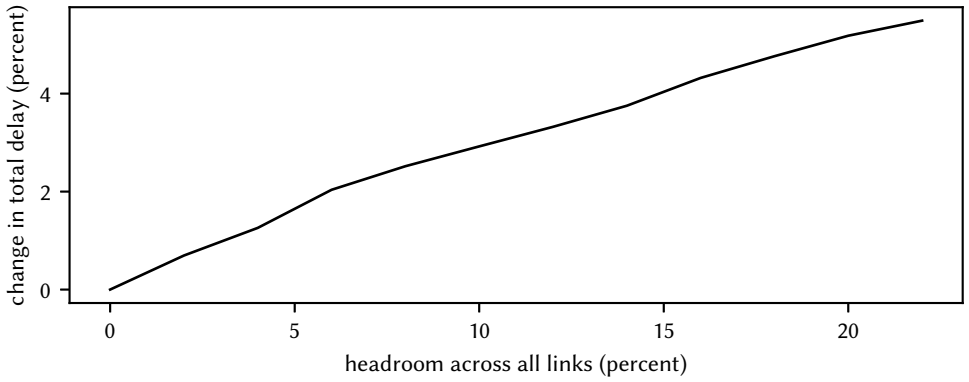



**Figure 7.25:** Increase in delay due to adding a fixed amount of headroom to all links.

couple of minutes, but during the third minute an unexpected event happens that causes one or more of the aggregates that cross the link to increase their level. The new traffic level overshoots the link's capacity and causes a queue to form. It takes a couple of seconds for LDR to trigger an optimization, which immediately moves some aggregates away from the congested link, but in the meantime a queue forms, and potentially drops may occur. In this experiment we always provide the optimizer with all aggregates' exact mean traffic levels for the upcoming minute, but this knowledge of the future is of little use as the unexpected change happens mid-minute.

Events like the one in Figure 7.23 are the result of sudden changes in the mean traffic level which are often unpredictable and completely out of the control of the routing system. The only reasonable way to deal with such unexpected changes in the traffic pattern is to add headroom on top of the one that is needed to deal with short-term variability.

### 7.4.3 Long-term Variability and Headroom

The question then becomes how much headroom to add. On one hand, adding headroom will reduce transient queuing in the event of an unexpected change in the traffic's mean level. On the other hand, adding headroom will, by design, increase the propagation delay of flows within the network. On Figure 7.25 we examine the magnitude of this fundamental tradeoff for the particular topology and the traffic matrix used in this section.

For every point on this plot we add a fixed amount of headroom to *all* links in the network, and observe the negative effect on delay by running LDR's optimization from Chapter 5. On the $x$ axis we vary the headroom added in increments of 2%; on the $y$ axis we plot the increase in total delay experienced by all flows in the network.

In this particular case the trend is linear with delay increasing roughly by .5% for every 2% of additional headroom. In general, however, the shape of the curve is very dependent on the topology and the traffic matrix. It is up to the network administrator to decide what an acceptable tradeoff is for their network and workload. As we have previously stated, we believe that 10% headroom

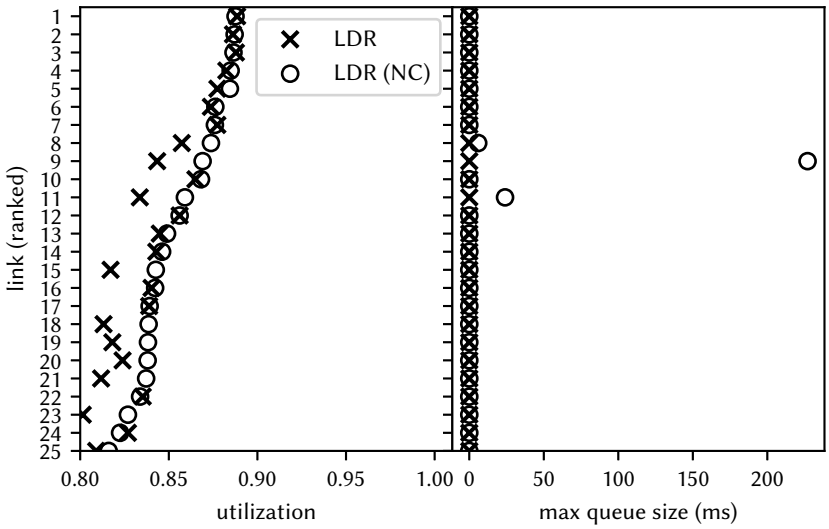



**Figure 7.26:** Mean link utilization (left) and max queue size (right) for the first ten minutes; 10% headroom target for the mean level estimation algorithm. Links are ranked based on the utilization of LDR (NC).

provides a reasonable default tradeoff between propagation delay and queuing delay.

We now verify this assumption. The mean-level estimation algorithm from Section 6.2.3 attempts to constantly maintain about 10% of headroom in hope that it will be enough to either completely absorb unexpected changes or to give LDR enough time to trigger an optimization. In order to test its effectiveness, along with its interaction with all other components of LDR previously tested, we enable the long-term prediction algorithm and repeat the previous experiment, but this time we simulate a longer period of time. In Figure 7.26 we show the results from the first 10 minutes of simulated time. Clearly LDR is effective in providing low queuing delay, while still loading links in the network up to the 90% target. Notice that even with 10% headroom the convolution mechanism is still needed—attempting to optimize the network by using mean traffic levels alone results in a link being congested.

Looking at Figure 7.26, an obvious question is whether it is possible to safely further reduce the amount of headroom. We repeat the experiment, but this time we set the mean-level estimation algorithm's headroom target to be 5% instead of 10%. The results are shown in Figure 7.27. Clearly, the network is running closer to the edge, with several links being congested in the absence of the convolution scheme. Even with the convolution scheme a couple of links run up queues in the 10-15ms range, which suggests that it may be safer to use the 10% variant if the network operator is willing to tolerate the increase in delay (which is about 2% according to Figure 7.25).

### 7.4.4 Triggered Optimization and Limited Optimization

Clearly as Figure 7.23 shows, in the event of an unexpected change in traffic level triggered optimization limits the amount of transient queuing. The downside of triggered optimization is that

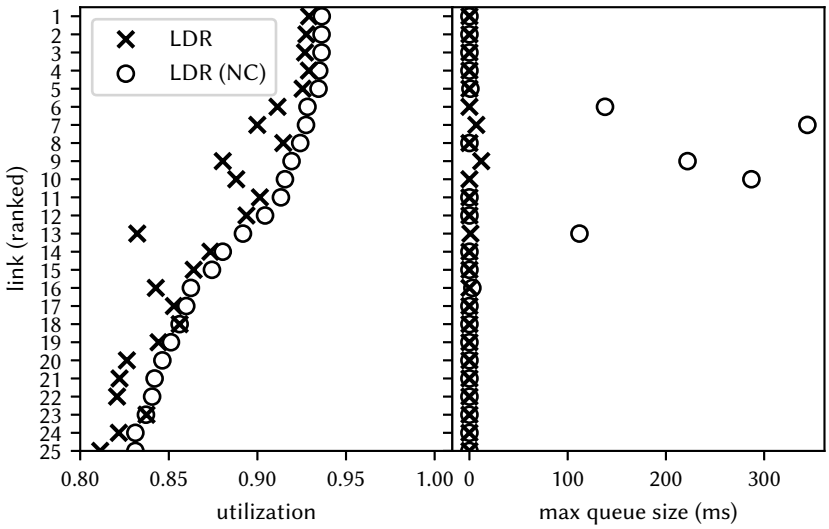



**Figure 7.27:** Mean link utilization (left) and max queue size (right) for the first ten minutes; 5% headroom target for the mean level estimation algorithm. Links are ranked based on the utilization of LDR (NC).

**Table 7.1:** Route changes sent by the controller during the simulation from Figure 7.26.

| time (sec) | routes added | routes updated | routes removed | optimization type |
|---|---|---|---|---|
| 60 | 0 | 0 | 0 | scheduled |
| 120 | 49 | 13 | 48 | scheduled |
| 155 | 40 | 15 | 39 | triggered |
| 160 | 112 | 17 | 110 | triggered |
| 180 | 36 | 20 | 36 | scheduled |
| 240 | 49 | 20 | 48 | scheduled |
| 300 | 44 | 20 | 43 | scheduled |
| 360 | 31 | 23 | 32 | scheduled |
| 420 | 80 | 20 | 82 | scheduled |
| 480 | 30 | 17 | 31 | scheduled |
| 540 | 30 | 18 | 28 | scheduled |
| 595 | 171 | 22 | 169 | triggered |
| 600 | 16 | 23 | 17 | scheduled |

it may cause the controller to update a large volume of network state. To quantify this cost we present in Table 7.1 the number of routes added, updated and removed by the controller in the entire network as the result of each optimization in the 10-minute long 10% headroom experiment from last section. Route additions are the most expensive, as they will cause an entirely new tag-switched path to be installed in the network. We believe that the number of route additions is reasonable for a network of this size, but the administrator may wish to further reduce that number by enabling LDR's limited optimization mechanism described in Section 5.6 and evaluated in isolation in Section 7.3.8. To observe its effects on message count and queue sizes we repeat the 10-minute 10% headroom experiment, but this time we enable the limited optimization mechanism.

On Table 7.2 we present the message counts and on Figure 7.28 we present queue sizes and

**Table 7.2:** Route changes sent by the controller during the simulation from Figure 7.26, but with limited optimization enabled.

| time (sec) | routes added | routes updated | routes removed | optimization type |
|---|---|---|---|---|
| 60 | 0 | 0 | 0 | scheduled |
| 120 | 44 | 12 | 42 | scheduled |
| 155 | 28 | 15 | 21 | triggered |
| 160 | 78 | 22 | 67 | triggered |
| 180 | 13 | 13 | 9 | scheduled |
| 240 | 3 | 5 | 0 | scheduled |
| 300 | 25 | 22 | 21 | scheduled |
| 360 | 0 | 1 | 1 | scheduled |
| 420 | 0 | 0 | 0 | scheduled |
| 480 | 0 | 0 | 0 | scheduled |
| 540 | 0 | 0 | 0 | scheduled |
| 600 | 10 | 9 | 4 | scheduled |

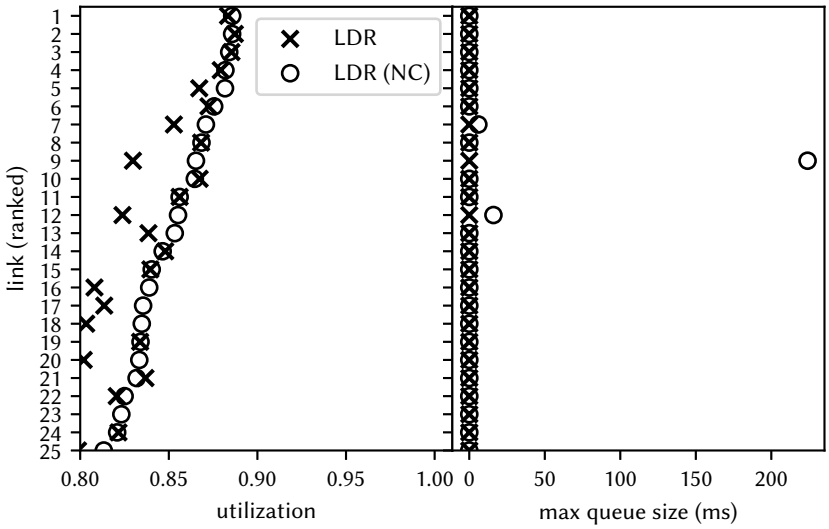



**Figure 7.28:** Mean link utilization (left) and max queue size (right) for the first ten minutes; 10% headroom target for the mean level estimation algorithm; limited optimization enabled. Links are ranked based on the utilization of LDR (NC).

link utilizations for this experiment. The effect of limited optimization is obvious, with a significant reduction in message counts compared to Table 7.1. Notice that limited optimization not only resulted in overall reduction in all types of control messages, but also completely elided the triggered optimization at 595 sec.

## Chapter 8

# Conclusions

Given today's applications' increasing reliance on low latency, in this thesis we set out to explore the interplay between the diversity of low-latency paths in WAN backbones and the ability of a routing scheme to exploit that diversity to achieve congestion-free, low-delay traffic delivery. We demonstrated that on topologies with diverse low-latency paths, both legacy and current routing schemes arrive at traffic placements that suffer congestion or high latency stretch.

We then designed LDR, a novel routing system that achieves low latency by minimizing propagation delay while avoiding queuing delay. To use ISP resources efficiently, LDR runs links at high utilization. Because doing so under variable, uncontrolled demand risks congestion, LDR adapts quickly when demand for a link bumps up against that link's capacity, and runs low-delay paths in the network as close to "the edge of congestion" as it safely can. As there is a limit to how frequently any demand-sensitive routing scheme can change the placement of traffic, to avoid congestion, LDR predicts whether aggregates' variable demands will statistically multiplex on a path in between changes in traffic placement.

We evaluated LDR in simulation and showed that LDR's optimizer can place thousands of aggregates in a backbone of hundreds of links in less than a second. We demonstrate that while under realistic load in today's networks *any* routing scheme will route 90% of flows on their shortest paths, the choice of routing scheme can profoundly affect the path stretch of the other 10%. LDR consistently yields very low path stretch for all flows in the network, across diverse load levels and locality degrees of traffic matrix. LDR is also free of the sub-optimal greedy behavior of prior systems which can cause congestion. Using real-world packet traces we demonstrated that LDR is able to safely run key links at high utilization, while adapting quickly enough to avoid causing queuing delays, even under the relatively uncontrolled demands in an ISP backbone.

In the rest of this chapter we first discuss limitations in LDR's evaluation. We then focus on open questions in resilience to failures and the general question of interdepedence between routing and topology—an interesting discussion which we touch upon in this thesis, but do not completely address. We conclude with further questions for future work.

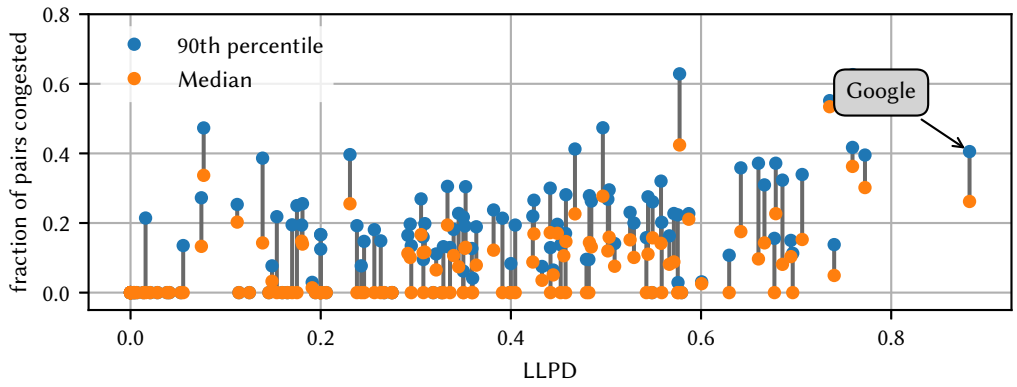



**Figure 8.1:** Same shortest-path routing data as in Figure 3.8, but with Google's topology (LLPD = 0.875) added.

## 8.1 Limitations in LDR's Evaluation

When evaluating LDR in Chapter 7 we used many real-world topologies, but we cannot claim that LDR will perform well on every possible topology. In particular, as shown in Section 7.3.7, highly loaded and densely connected networks may cause the optimizer to either take a long time to produce a solution or produce a sub-optimal one.

Similarly, we have tested LDR's convolution-based mechanism using a number of real-life backbone traces, but we can not claim that these traces are representative of all backbone traffic. It may be that in some networks aggregates experience significantly more short and long-term variability than in our traces, which would require more headroom than the 10% default. As we discussed in Section 6.3.1, we envision that in such cases increased variability will result in a high number of triggered optimizations, which can be taken as an indication that the headroom target should be increased, but we have not fully designed or evaluated such a mechanism.

The traffic matrices that we evaluated LDR with are synthetic. We attempted to produce realistic traffic matrices and we evaluated LDR under workloads with a range of different load and locality parameters, but ultimately we cannot claim that real-world traffic matrices will always be similar to the ones we tested.

In Section 3.3.2 we demonstrated that mean traffic levels are predictable minute to minute, which motivated LDR's minute-long optimization period. We believe that a minute is a reasonable default for most networks, but we have not evaluated longer or shorter periods. Because of triggered optimization we do not believe that performing periodic optimization passes more often would result in lower queuing delay, but it is possible that doing so may yield lower propagation delay, as it would make LDR react faster when an aggregate's demand decreases.

## 8.2 Modern enterprise networks.

The Topology Zoo consists largely of transit networks from recent decades, most of which were not designed with dynamic, latency-minimizing routing in mind. How do state-of-the-art enter-

prise networks compare? We examined a recent wide-area global enterprise network owned by Google [43]. In Figure 8.1 we revisit the behavior of delay-proportional shortest-path routing by augmenting Figure 3.8 with results for Google's network. The new datapoint clearly exhibits the greatest LLPD among all topologies and, unsurprisingly, cannot be routed using shortest paths alone. Google's own B4 in fact performs nearly optimally on this network without exhibiting the pathologies in Section 3.2.2. We conjecture that this topology was explicitly designed for dynamic latency-minimizing routing. We believe it to be an important existence proof that it is possible and economically viable to build a high-LLPD network that spans the globe. We note though that an enterprise network can control traffic at endpoints, so demand may be more predictable than at an ISP.

## 8.3 Resilience to Failures

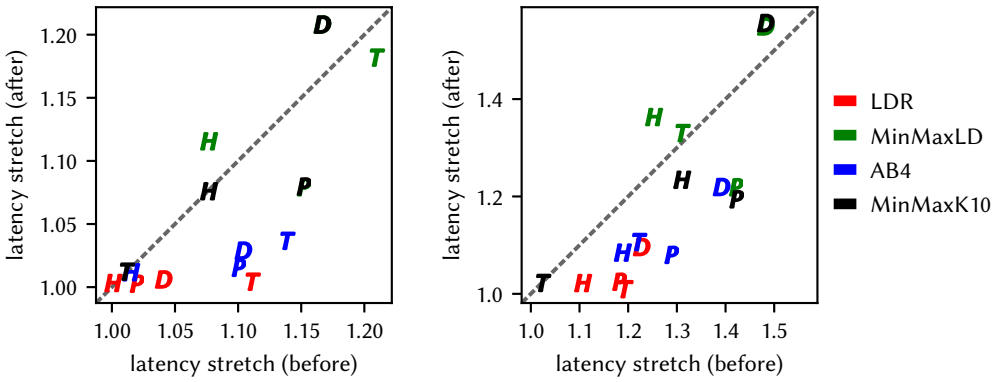
There are primarily two types of failures a centralized load-dependent routing system is concerned with:

- Failures of components in the forwarding plane—*e.g.,* a transoceanic fiber is cut by a ship's anchor.
- Failures in the control plane—*e.g,* the central controller is destroyed or disconnected.

In general, failures of components of the forwarding plane are less problematic. As we explain in Section 4.2, LDR, like similar systems, runs on top of link-state routing which will guarantee connectivity to the controller if the network is not partitioned. Once a failure occurs the controller will have to react quickly to update the forwarding state in devices. As failure can be seen as an extreme case of an unexpected change in traffic level, a mechanism similar to triggered optimization can help in this case, as shown in Figure 7.23.

More concerning is the potential for network partition. Even though low-delay routing targets networks that have high path diversity, in wide-area networks long-haul transoceanic links pose a high risk to connectivity, as cutting only a handful of those is likely to partition the network. In this case we envision positioning a number of "local" controller replicas that can take over the central controller's functions. In normal operation, devices send measurements to their local controller and the local controllers forward traffic measurements to the global one, so that local controllers have an up-to-date view of the aggregates in their part of the network. In case a partition does not contain a local controller, the underlying link-state IGP can still ensure single shortest-path connectivity after LSPs eventually time out.

What about the control plane? The global controller clearly is a single point of failure and its loss will potentially be devastating to the system. Traditionally in SDN networks this problem is handled via state-machine replication [51] or a lighter-weight version of state machine replication [48]. More recent work in the field [64] suggests that an even lighter-weight approach based




**(a)** Median **(b)** 90th percentile

**Figure 8.2:** Latency benefits of network growth; graph shows median and 90th percentile of path stretch before and after growing networks to increase their LLPD; each letter is a different topology: Packetexchange (P), Deutsche Telekom (D), Hurricane Electric (H) and Tinet (T).

on eventual correctness may also be viable and yield failover times of under 100 ms.

In practice, given our experience with LDR, we believe even the simpler heavier-weight approaches might be enough to render a centralized load-dependent routing system fault-tolerant, as the central controller does not require a very low failover time. If the controller dies at any point between update cycles it is likely that all aggregates in the network still have at least one valid path. In this case during failover the network may experience sub-optimal propagation delay or temporary transient congestion. If the controller dies in the middle of installing new network state for a set of aggregates their ingresses may switch to single shortest path routing for those aggregates during the failover period. Neither of those events actually results in loss of connectivity.

## 8.4 Influence of Routing on Topology

The topologies we have studied in this work were designed to be used with existing routing schemes. Have today's routing systems' limitations constrained how networks themselves have grown? We cannot definitively answer this question without deploying an optimal routing system and waiting a decade or so to see how ISPs upgrade their networks. Nor can we accurately determine which topology upgrades might be likely; we have no model for the economic and geopolitical constraints that gate new link deployment. We can, however, examine the extent to which topology upgrades enable better service from today's routing systems. When adding links to a topology in principle ought to improve service but in practice does not, an ISP wouldn't likely choose to grow the network in that way. If, however, the ISP had a routing system that could harness those added links to improve service, the ISP would see benefit in adding them.

### 8.4.1 Multi-step Upgrade Using LLPD

As there is no point in upgrading a network topology that already works well, we examined four networks that are difficult to route with low latency, even with optimal traffic placement. The networks chosen are those from Figure 3.9a with high latency, but we exclude those with clique topologies, to which we cannot add links. We use the LLPD metric to determine which additional links might confer the greatest benefit.

We consider every pair of POPs not already directly connected, and evaluate how LLPD would change if we added a link between that pair of POPs. Note that some of these links might *decrease* LLPD. Of all these links, we add the one that gives the greatest increase in LLPD. We then repeat this process until the number of links has increased by 5%. As each link added improves LLPD, the resulting network will, in principle, be more amenable to low-latency routing.

Figure 8.2 shows how much the different routing schemes benefit. As before, load is 77%, and locality 1. The $x$-axis and $y$-axis respectively show the latency stretch on the original and enhanced networks. The baselines for the $x$ axis are shortest paths before the repeated addition of links, and the baselines for the $y$ axis are the (possibly different) shortest paths after the new links were added. We show two plots where each point is either the median (left plot) or the 90th percentile (right plot) of the latency stretch when running different schemes with the same set of traffic matrices on the original and enhanced topologies. Different colors are different schemes and different letters are different topologies. Ideally, all points would be close to the $x$-axis.

Only LDR truly manages to completely take advantage of the new links, giving median latency stretch very close to unity. For three of the networks, LDR's 90th percentile is less than all other routing systems' median latency.

B4 is also good at taking advantage of new links, though far from perfect. Both MinMax algorithms fare much worse. In some cases, adding new links that improve LLPD actually *increases* latency, as both algorithms use new links to load-balance more widely. We also note that with pure shortest-path routing (not shown on the figure), adding links sometimes causes congestion, if the link has insufficient capacity to cope with any traffic concentration it causes.

We conjecture on the basis of these preliminary experiments that the routing scheme does determine which links are best for an ISP to add. Although we cannot be sure that limitations in today's routing systems prevent ISPs from deploying lower-latency topologies, it seems that may be the case.

### 8.4.2 Single-step Upgrade

One can argue that LLPD is not the right metric an operator should use to determine how to grow their network. In fact, geographic, geopolitical, and economic constraints limit where links can reasonably be provisioned and one can argue that no single metric is right for all operators. At bottom, we observe that regardless of the operator's agenda or resources, adding capacity should

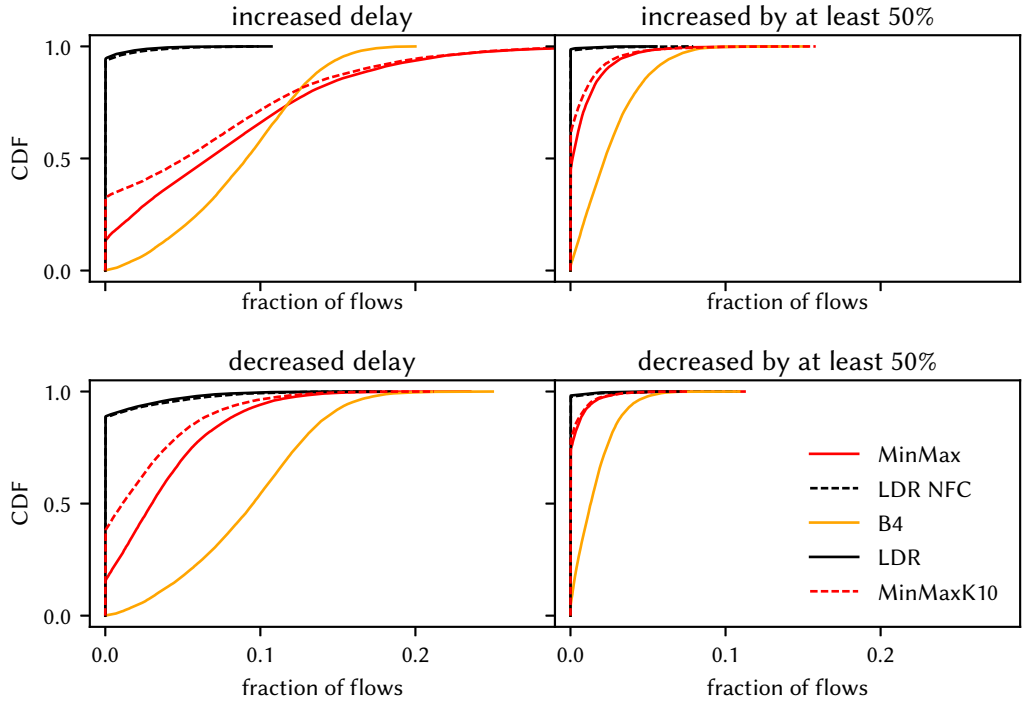



**Figure 8.3:** Fraction of flows whose delay increases/decreases when an existing link is upgraded in Hurricane Electric's network

not actually be *detrimental* to latency for many flows.

To capture the extent to which routing schemes hamper network growth in a way that does not depend on any single metric, we focus on Hurricane Electric's (HE) network, one of the networks from Figure 8.2. HE is a global ISP with 24 POPs and median link size of 10 Gbps. Instead of repeatedly adding a single link over multiple steps, this time we tested *all* possible single-step upgrades to this topology: either we added a new 10 Gbps link between one of the 240 possible pairs of POPs, or we upgraded one of the 36 existing links by 10 Gbps. We evaluated the upgrade using all the HE traffic matrices used in Figure 8.2, testing all the routing schemes. Figures 8.3 and 8.4 are CDFs across all possible link upgrades and traffic matrices showing the fraction of flows whose propagation delay increased (top graphs) or decreased (bottom graphs) as a result of the upgrade; Figure 8.4 shows the effect of adding new links, Figure 8.3 shows the effect of increasing capacity on existing links.

It is interesting to see that adding capacity to a MinMax network causes many more flows to increase in latency than to decrease. MinMax will, as designed, spread traffic out onto the new link, even if doing so hurts latency. The ISP is, in fact, very constrained in how it grows its network over time if it wants to do no harm.

With B4, many flows change path when links are added, though the number whose latency increases almost exactly equals those whose decreases. Although the net change is a slight reduc-

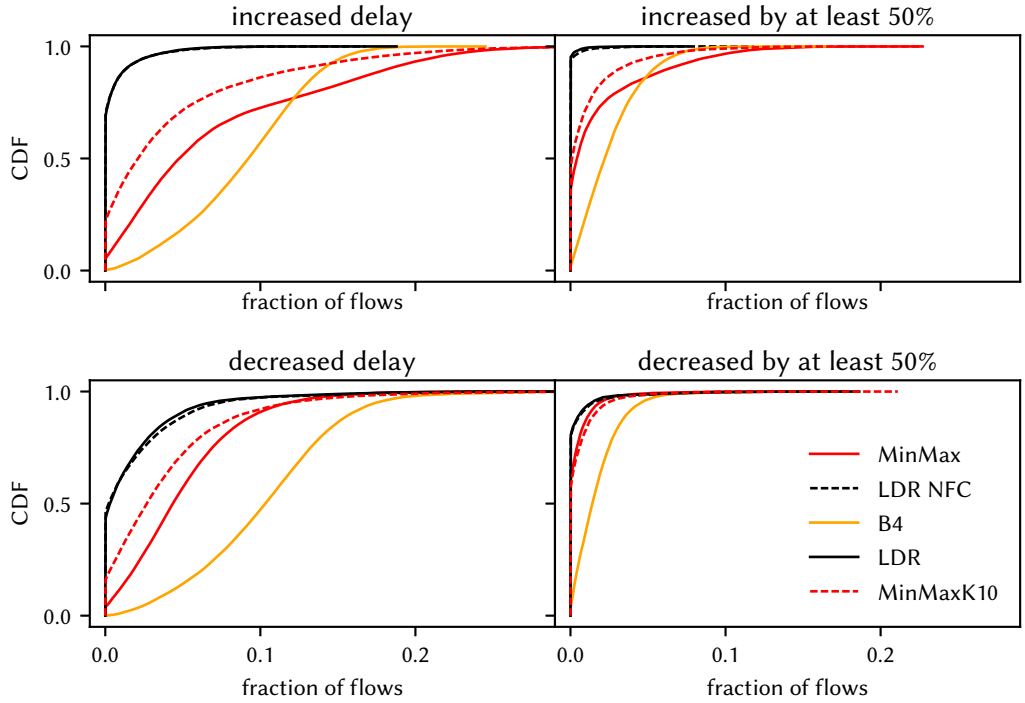



**Figure 8.4:** Fraction of flows whose delay increases/decreases when a new link is added to Hurricane Electric's network

tion in total latency, an ISP planning a change will have difficulty predicting which customers will be adversely affected. This same churn is also seen when B4 reacts to link state changes.

LDR minimizes both total latency and per-aggregate latency stretch. As a result, although many possible link upgrades or additions are beneficial, very few cause any flows at all to increase in latency. We speculate that this property will allow network operators extra freedom to grow their networks in new ways that are not possible today due to the constraints imposed by legacy routing systems.

## 8.5 Future Research

In LDR we have taken a step toward routing that better harnesses the diversity in today's ISP topologies, but important questions and avenues for further work in this area remain.

### Can we guarantee stability?

Given a set of input demands, the routing system should eventually settle on a stable assignment of traffic to paths. Unfortunately, it is difficult for delay-minimizing routing systems to guarantee stability in general.

As already discussed, any delay-minimizing routing system must dynamically move traffic from lower-delay paths to higher-delay ones when demand increases and no longer fits, and vice-versa when demand shrinks. Moving an aggregate to a higher-delay path can, however, reduce

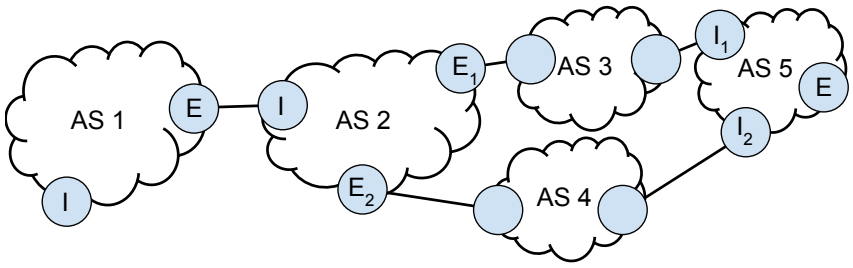



**Figure 8.5:** AS-level topology, ingress and egress devices shown.

its throughput—*e.g.,* when (i) its flows are competing with flows on a bottleneck link outside the ISP's network (as the TCP throughput equation tells us [63]), (ii) the traffic is generated by delay-sensitive applications, or (iii) delay-based congestion control is applied [17]. In turn, reducing the throughput of the moved aggregate increases the possibility that the routing system will shift that aggregate to a lower-delay path, increasing its bandwidth again. This pattern of behavior can result in temporary instability or a permanent oscillation—never settling on a stable routing state. Oscillations of this type are fundamentally different from the widely studied ones that affect distributed routing [49, 9].

The magnitude of these instabilities depends on how strong the correlation between throughput and delay is. Measuring the magnitude of dependencies of this sort in real-world flows, and characterizing and dealing with oscillations they cause is an interesting open problem. The study of practical mechanisms that guarantee stability, as well as techniques to mitigate instability when it cannot be avoided, are fertile ground for the community to explore.

### Can we provide low latency end-to-end?

So far, we have considered a single ISP, and a routing system optimizing paths for flow aggregates defined by a fixed pair of ingress and egress routers within a single administrative domain. How does such an optimization fit in the larger end-to-end picture? If all autonomous systems (AS) on the path of a flow in the Internet were each to perform a latency-minimizing optimization locally, what would the global outcome be? Clearly, if for a given destination each AS has only one ingress and egress point, minimizing the propagation delay of each segment of the end-to-end path will also minimize the total end-to-end propagation delay.

In the real world, however, a single destination can be advertised from multiple neighboring autonomous systems, resulting in many potential egresses for an ingress. A small AS-level topology illustrating this is shown in Figure 8.5. Imagine a flow from the ingress device of AS1 to the egress device of AS5. Such flow would have to cross AS2, where the routing system has to make a choice between two possible egress points. If it makes a purely local decision and sends the flow to $E_2$, the end-to-end propagation delay may suffer if the path via AS4 is longer than the one in AS3. The only way to ensure minimal end-to-end delay here is to make AS2 aware of the optimization processes in other domains.

We believe it will be worthwhile to explore how to achieve end-to-end minimal latency with and without coordination between ISPs in the hot-potato routing scenario—*e.g.,* through tailored ISP interfaces for automated interactions, or independent decisions made by single ISPs on the basis of external latency measurements.

### How does routing react to malicious traffic?

We have yet to explore how load-dependent routing schemes react to malicious traffic patterns. For example, although LDR will blunt a DDoS attack by spreading its bandwidth, flow counts might also be inflated, causing priority inversion. The flow-agnostic variant of LDR would not be affected in such cases, but the flow-aware variant of LDR would favor lower-delay paths for an aggregate containing lots of small flows, essentially prioritizing DDoS or malicious traffic. It seems likely that flow counting might be useful in detecting such attacks and deprioritizing their traffic, or alerting the operator to do so.

### Can we apply LDR's techniques to other routing systems?

We believe that LDR's iterative growth of the set of paths used to route an aggregate and its convolution technique for determining headroom should both be of use in other low-delay routing systems. For example, B4 assumes no variation in traffic demands, as it is designed for an enterprise setting in which the routing controller has global knowledge of all sources' exact rates. The convolution approach to headroom could be useful in adapting B4 to the ISP setting. And while MinMax $K10$'s fixed choice of the ten lowest-delay paths will often be too great or too small for some aggregates, iteratively growing the path set for MinMax per aggregate, subject to a bound on delay stretch, should help MinMax avoid needless detours.

## 8.6 Closing Remarks

Users' expectations of the Internet have evolved beyond mere capacity. Today's Internet applications offer the best quality of experience when end-to-end communication incurs low latency. A web page finishes loading when TCP transfers of many small objects complete; as these short TCP flows often finish while still in slow start, round-trip time gates page load time. And the lower the communication latency between two users, the more responsive they will find interactive applications, such as VoIP, instant messaging, and gaming.

As the volume of delay-sensitive traffic grows, in the near future this desire for low latency will shape the way networks are built and operated. In this thesis we focused on the fundamental tradeoff between propagation delay and queuing delay in the context of wide-area routing. We close by noting that the two main prerequisites to any demand-adaptive latency-minimizing routing system are *predictability of demand at the timescale of that system's control loop* and *low-latency path diversity*.

Our experience suggests that the level of aggregation evident in today's large ISPs is now

sufficient for traffic to be relatively predictable on minute-to-minute timescales. We believe this regime presents exciting opportunities for building practical demand-adaptive routing systems, such as the one we presented in this work. Whereas a large fraction of today's networks do not exhibit high path diversity, we hope that such routing systems will provide network operators with greater freedom to upgrade their networks, and help unlock the low-latency potential of path-diverse topologies.

# Bibliography


[1] NTT's network topology. `http://www.us.ntt.net/about/network-map.cfm`. Accessed: 2017-08-04.

[2] Al-Fares, M., Radhakrishnan, S., Raghavan, B., Huang, N., and Vahdat, A. Hedera: Dynamic flow scheduling for data center networks. In *USENIX NSDI 2010*.

[3] Albrightson, R., Garcia-Luna-Aceves, J., and Boyle, J. Eigrp–a fast routing protocol based on distance vectors. *Interop 94* (1994).

[4] Alizadeh, M., Greenberg, A., Maltz, D. A., Padhye, J., Patel, P., Prabhakar, B., Sengupta, S., and Sridharan, M. Data center tcp (dctcp). In *ACM SIGCOMM computer communication review* (2010), vol. 40, ACM, pp. 63–74.

[5] Anthony, S. $1.5 billion: the cost of cutting london-tokyo latency by 60ms. https://news.slashdot.org/story/12/03/21/004219/15-billion-the-cost-of-cutting-london-tokyo-latency-by-60ms, 2012.

[6] Anthony, S. The secret world of microwave networks. https://arstechnica.com/information-technology/2016/11/private-microwave-networks-financial-hft, 2016.

[7] Applegate, D., and Cohen, E. Making intra-domain routing robust to changing and uncertain traffic demands: Understanding fundamental tradeoffs. In *ACM SIGCOMM 2003*.

[8] Azar, Y., Cohen, E., Fiat, A., Kaplan, H., and Racke, H. Optimal oblivious routing in polynomial time. In *Proceedings of the thirty-fifth annual ACM symposium on Theory of computing* (2003), ACM, pp. 383–388.

[9] Basu, A., and Riecke, J. Stability issues in OSPF routing. *ACM SIGCOMM Computer Communication Review 31*, 4 (2001), 225–236.

[10] Beck, M., and Kagan, M. Performance evaluation of the rdma over ethernet (roce) standard in enterprise data centers infrastructure. In *Proceedings of the 3rd Workshop on Data Center-Converged and Virtual Ethernet Switching* (2011), International Teletraffic Congress, pp. 9–15.

[11] Bellman, R. On a routing problem. *Quarterly of applied mathematics 16*, 1 (1958), 87–90.

[12] Bertsekas, D., Gafni, E., and Gallager, R. Second derivative algorithms for minimum delay distributed routing in networks. *IEEE Transactions on Communications 32*, 8 (1984), 911–919.

[13] Bertsekas, D. P., Gallager, R. G., and Humblet, P. *Data networks*, vol. 2. Prentice-hall Englewood Cliffs, NJ, 1987.

[14] Braess, D. Über ein paradoxon aus der verkehrsplanung. *Unternehmensforschung 12*, 1 (Dec 1968), 258–268.

[15] CAIDA. Internet data – passive data sources.

[16] Callon, R. Rfc 1195, use of osi isis for routing in tcp. *IP and Dual Environments* (1990), 1–80.

[17] Cardwell, N., Cheng, Y., Gunn, C. S., Yeganeh, S., and Jacobson, V. Bbr: Congestion-based congestion control. *Communications of the ACM 60*, 2 (2017), 58–66.

[18] Chen, Y., Griffith, R., Liu, J., Katz, R. H., and Joseph, A. D. Understanding tcp incast throughput collapse in datacenter networks. In *Proceedings of the 1st ACM workshop on Research on enterprise networking* (2009), ACM, pp. 73–82.

[19] Cisco. Implementing mpls traffic engineering.

[20] Clegg, R. G. *The statistics of dynamic networks*. PhD thesis, University of York, 2004.

[21] CPLEX, I. I. V12. 1: User's manual for cplex. *International Business Machines Corporation 46*, 53 (2009), 157.

[22] Curtis, A. R., Mogul, J. C., Tourrilhes, J., Yalagandula, P., Sharma, P., and Banerjee, S. Devoflow: Scaling flow management for high-performance networks. *ACM SIGCOMM Computer Communication Review 41*, 4 (2011), 254–265.

[23] D. Awduche and J. Malcolm. Requirements for Traffic Engineering Over MPLS. *RFC 2702* (2009).

[24] Davie, B. S., and Rekhter, Y. *MPLS: technology and applications*. Morgan Kaufmann Publishers Inc., 2000.

[25] Dijkstra, E. W. A note on two problems in connexion with graphs. *Numerische mathematik 1*, 1 (1959), 269–271.

[26] Dijkstra, E. W., and Scholten, C. S. Termination detection for diffusing computations. *Information Processing Letters 11*, 1 (1980), 1–4.

[27] Duffield, N., Lund, C., and Thorup, M. Estimating flow distributions from sampled flow statistics. In *ACM SIGCOMM 2003*.

[28] Dukkipati, N. Tail Loss Probe (TLP): An Algorithm for Fast Recovery of Tail Losses. *Internet Draft draft-dukkipati-tcpm-tcp-loss-probe-01.txt* (2013).

[29] Elwalid, A., Jin, C., Low, S., and Widjaja, I. Mate: Mpls adaptive traffic engineering. In *Proc. INFOCOM 2001*, vol. 3, pp. 1300 – 1309.

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

[31] Fortz, B., and Thorup, M. Optimizing OSPF/IS-IS weights in a changing world. *IEEE J.Sel. A. Commun. 20*, 4 (Sept. 2006), 756–767.

[32] Foundation, O. N. Openflow switch specification, version 1.5.1.

[33] games, R. Fixing the internet for real time applications: part ii. https://engineering.riotgames.com/news/fixing-internet-real-time-applications-part-ii, 2016.

[34] Garcia-Lunes-Aceves, J. J. Loop-free routing using diffusing computations. *IEEE/ACM Transactions on Networking (TON) 1*, 1 (1993), 130–141.

[35] Gay, S., Schaus, P., and Vissicchio, S. REPETITA: Repeatable Experiments for Performance Evaluation of Traffic-Engineering Algorithms. *CoRR abs/1710.08665* (2017).

[36] Gong, W.-B., Liu, Y., Misra, V., and Towsley, D. Self-similarity and long range dependence on the internet: a second look at the evidence, origins and implications. *Computer Networks 48*, 3 (2005), 377–399.

[37] Gvozdiev, N., Karp, B., and Handley, M. FUBAR: Flow utility based routing. In *Proc. ACM Hotnets* (October 2014).

[38] Haddadi, H., Bonaventure, O., et al. Recent advances in networking.

[39] Handley, M., Raiciu, C., Agache, A., Voinescu, A., Moore, A. W., Antichi, G., and Wójcik, M. Re-architecting datacenter networks and stacks for low latency and high performance. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication* (2017), ACM, pp. 29–42.

[40] Handley, M., Raiciu, C., and Wischik, D. htsim. *http://nrg.cs.ucl.ac.uk/mptcp/implementation.html*.

[41] Hassidim, A., Raz, D., Segalov, M., and Shaqed, A. Network utilization: The flow view. In *INFOCOM, 2013 Proceedings IEEE* (2013), IEEE, pp. 1429–1437.

[42] Hong, C.-Y., Kandula, S., Mahajan, R., Zhang, M., Gill, V., Nanduri, M., and Wattenhofer, R. Achieving high utilization with software-driven WAN. In *ACM SIGCOMM 2013*.

[43] Hong, C.-Y., Mandal, S., Al-Fares, M., Zhu, M., Alimi, R., B., K. N., Bhagat, C., Jain, S., Kaimal, J., Liang, S., Mendelev, K., Padgett, S., Rabe, F., Ray, S., Tewari, M., Tierney, M., Zahn, M., Zolla, J., Ong, J., and Vahdat, A. Managing Hierarchy, Partitioning, and Asymmetry for Availability and Scale in a Software-Defined WAN. In *Proc. ACM SIGCOMM* (2018).

[44] Hurst, H. E. Long-term storage capacity of reservoirs. *Trans. Amer. Soc. Civil Eng. 116* (1951), 770–808.

[45] Jain, S., Kumar, A., Mandal, S., Ong, J., Poutievski, L., Singh, A., Venkata, S., Wanderer, J., Zhou, J., Zhu, M., Zolla, J., Hölzle, U., Stuart, S., and Vahdat, A. B4: Experience with a globally-deployed software defined wan. In *ACM SIGCOMM 2013*.

[46] Jin, X., Liu, H. H., Gandhi, R., Kandula, S., Mahajan, R., Zhang, M., Rexford, J., and Wattenhofer, R. Dynamic scheduling of network updates. In *ACM SIGCOMM Computer Communication Review* (2014), vol. 44, ACM, pp. 539–550.

[47] Kandula, S., Katabi, D., Davie, B., and Charny, A. Walking the tightrope: Responsive yet stable traffic engineering. In *Proc. ACM SIGCOMM 2005*.

[48] Katta, N., Zhang, H., Freedman, M., and Rexford, J. Ravana: Controller fault-tolerance in software-defined networking. In *Proceedings of the 1st ACM SIGCOMM symposium on software defined networking research* (2015), ACM, p. 4.

[49] Khanna, A., and Zinky, J. The revised ARPANET routing metric. *ACM SIGCOMM Computer Communication Review 19*, 4 (1989), 45–56.

[50] Knight, S., Nguyen, H., Falkner, N., Bowden, R., and Roughan, M. The Internet Topology Zoo. *IEEE Journal on Selected Areas in Communications 29*, 9 (october 2011), 1765 –1775.

[51] Koponen, T., Casado, M., Gude, N., Stribling, J., Poutievski, L., Zhu, M., Ramanathan, R., Iwata, Y., Inoue, H., Hama, T., et al. Onix: A distributed control platform for large-scale production networks. In *OSDI* (2010), vol. 10, pp. 1–6.

[52] Kumar, A., Jain, S., Naik, U., Raghuraman, A., Kasinadhuni, N., Zermeno, E., Gunn, C. S., Ai, J., Carlin, B., Amarandei-Stavila, M., Robin, M., Siganporia, A., Stuart, S., and Vahdat, A. Bwe: Flexible, hierarchical bandwidth allocation for wan distributed computing. In *ACM SIGCOMM 2015*.

[53] Langley, A., Riddoch, A., Wilk, A., Vicente, A., Krasic, C., Zhang, D., Yang, F., Kouranov, F., Swett, I., Iyengar, J., et al. The quic transport protocol: Design and internet-scale deployment. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication* (2017), ACM, pp. 183–196.

[54] Leland, W. E., Taqqu, M. S., Willinger, W., and Wilson, D. V. On the self-similar nature of ethernet traffic (extended version). *IEEE/ACM Transactions on networking 2*, 1 (1994), 1–15.

[55] McKeown, N. Software-defined networking. *INFOCOM keynote talk 17*, 2 (2009), 30–32.

[56] McQuillan, J., Falk, G., and Richer, I. A review of the development and performance of the arpanet routing algorithm. *IEEE Transactions on Communications 26*, 12 (1978), 1802–1811.

[57] McQuillan, J., Richer, I., and Rosen, E. The new routing algorithm for the arpanet. *IEEE Transactions on Communications 28*, 5 (1980), 711–719.

[58] Mogul, J. C., and Congdon, P. Hey, you darned counters!: get off my asic! In *Proceedings of the first workshop on Hot topics in software defined networks* (2012), ACM, pp. 25–30.

[59] Moy, J. OSPF Version 2. RFC 2328, Apr. 1998.

[60] Moy, J. Rfc 2328. *OSPF version 2* (1998).

[61] Nichols, K., Black, D. L., Blake, S., and Baker, F. Definition of the differentiated services field (ds field) in the ipv4 and ipv6 headers.

[62] Oran, D. OSI IS-IS Intra-domain Routing Protocol. RFC 1142, 1990.

[63] Padhye, J., Firoiu, V., Towsley, D., and Kurose, J. Modeling tcp throughput: A simple model and its empirical validation. In *Proc. ACM SIGCOMM 1998* (New York, NY, USA, 1998), SIGCOMM '98, ACM, pp. 303–314.

[64] Panda, A., Zheng, W., Hu, X., Krishnamurthy, A., and Shenker, S. Scl: Simplifying distributed sdn control planes. In *NSDI* (2017), pp. 329–345.

[65] Perlman, R. *Interconnections: bridges, routers, switches, and internetworking protocols*. Pearson Education India, 1999.

[66] Qiao, Y., Skicewicz, J., and Dinda, P. An empirical study of the multiscale predictability of network traffic. In *High performance Distributed Computing, 2004. Proceedings. 13th IEEE International Symposium on* (2004), IEEE, pp. 66–76.

[67] Racke, H. Minimizing congestion in general networks. In *Foundations of Computer Science, 2002. Proceedings. The 43rd Annual IEEE Symposium on* (2002), IEEE, pp. 43–52.

[68] Roughan, M. Simplifying the synthesis of internet traffic matrices. *Computer Communication Review 35*, 5 (2005), 93–96.

[69] Santana, G. A. *Data Center Virtualization Fundamentals: Understanding Techniques and Designs for Highly Efficient Data Centers with Cisco Nexus, UCS, MDS, and Beyond.* Cisco Press, 2013.

[70] Schlinker, B., et al. Engineering Egress with Edge Fabric: Steering Oceans of Content to the World. In *SIGCOMM* (2017).

[71] Schüller, T., Aschenbruck, N., Chimani, M., Horneffer, M., and Schnitter, S. Traffic engineering using segment routing and considering requirements of a carrier ip network. In *IFIP Networking Conference (IFIP Networking) and Workshops, 2017* (2017), IEEE, pp. 1–9.

[72] sFlow. *http://www.sflow.org*.

[73] Shaikh, A., and Greenberg, A. G. OSPF monitoring: Architecture, design, and deployment experience. In *NSDI* (2004).

[74] Shalom, N. Amazon found every 100ms of latency cost them 1% in sales. https://blog.gigaspaces.com/amazon-found-every-100ms-of-latency-cost-them-1-in-sales, 2008.

[75] Singla, A., Chandrasekaran, B., Godfrey, P. B., and Maggs, B. the internet at the speed of light.

[76] Steenbergen, R. MPLS RSVP-TE auto-bandwidth: Practical lessons learned. `https://www.nanog.org/sites/default/files/tues.general.steenbergen.autobandwidth.30.pdf`. Accessed: 2017-10-31.

[77] Studio, I. I. C. O. Numeric difficulties. `https://www.ibm.com/support/knowledgecenter/en/SSSA5P_12.6.2/ilog.odms.cplex.help/CPLEX/UsrMan/topics/cont_optim/simplex/20_num_difficulty.html`. Accessed: 2017-12-07.

[78] Templin, P. MPLS traffic engineering. `https://www.nanog.org/meetings/nanog37/presentations/pete-templin.pdf`. Accessed: 2017-10-31.

[79] Wang, H., Xie, H., Qiu, L., Yang, Y. R., Zhang, Y., and Greenberg, A. COPE: Traffic engineering in dynamic networks. In *ACM SIGCOMM 2006*.

[80] Wolf, J. V., and Brenkosh, J. P. The need for speed: 40 gigabit ethernet and beyond. Tech. rep., Sandia National Laboratories (SNL-NM), Albuquerque, NM (United States), 2016.

[81] Yap, K.-K., et al. Taking the Edge off with Espresso: Scale, Reliability and Programmability for Global Internet Peering. In *SIGCOMM* (2017).

[82] Yen, J. Y. An algorithm for finding shortest routes from all source nodes to a given destination in general networks. *Quarterly of Applied Mathematics 27*, 4 (1970), 526–530.

[83] Zhang, L., Deering, S., Estrin, D., Shenker, S., and Zappala, D. Rsvp: A new resource reservation protocol. *IEEE network 7*, 5 (1993), 8–18.

[84] Zhang, M., Karp, B., Floyd, S., and Peterson, L. Rr-tcp: a reordering-robust tcp with dsack. In *Network Protocols, 2003. Proceedings. 11th IEEE International Conference on* (2003), IEEE, pp. 95–106.