Low-Latency Routing on Mesh-Like Backbones

Nikola Gvozdiev, Stefano Vissicchio, Brad Karp, Mark Handley
University College London (UCL)

ABSTRACT

Early in in the Internet's history, routing within a single provider's WAN centered on placing traffic on the shortest path. More recent traffic engineering efforts aim to reduce congestion and/or increase utilization within the status quo of greedy, shortest-path first routing on a sparse topology. In this paper, we argue that this status quo of routing and topology is fundamentally at odds with placing traffic so as to *minimize latency for users* while avoiding congestion. We advocate instead provider backbone topologies that are more *mesh-like,* and hence better at providing *multiple* low-latency paths, and a routing system that directly considers latency minimization and congestion avoidance while dynamically placing traffic on multiple *unequal-cost* paths. We offer a research agenda for achieving this new low-latency approach to WAN topology design and routing.

1 INTRODUCTION

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 [20], then link-state [21, 24, 25]. 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 SP routing with traffic engineering (TE) mechanisms such as MPLS-TE [3] 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 2. 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.

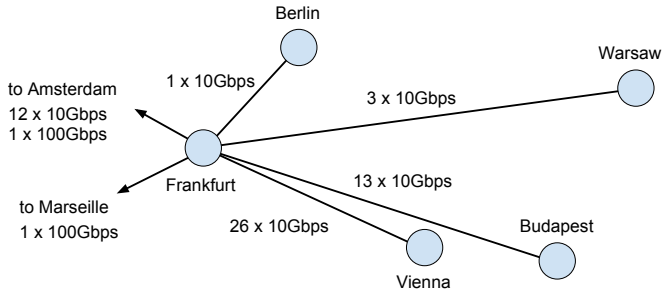


Figure 1: A subset of NTT's European network. Drawn to scale.

Users' expectations of the Internet have evolved beyond mere capacity, however. Web page load time is determined by completion times for short web TCP flows, which are gated by slow start, and thus round-trip time (RTT). Interactive applications, from gaming to voice and videoconferencing, offer the best quality of experience when latency is low. Providers need means to deliver not only capacity to end users, but also low latency [6, 28], and have an economic incentive to do so [23]. One seemingly promising strategy for cutting latency is to build more *mesh-like* backbones: to introduce links that carry demand along a more direct geographic path, shortcutting a more circuitous one. Unfortunately, SP routing *hampers the introduction of latency-cutting links into a backbone's topology,* making it hard to build low-latency, mesh-like 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, 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

HotNets-XVI, November 30–December 1, 2017, Palo Alto, CA, USA

ACM ISBN 978-1-4503-5569-8/17/11…$15.00
https://doi.org/10.1145/3152434.3152453

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.[1] At off-peak times, 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. In this way, capacity is provided cheaply, and at least some traffic sees improved latency.

TE schemes [2, 9, 10, 13, 16, 31] can, in principle, solve this problem. To do so, they need to split the Budapest → Vienna traffic unequally, and do so automatically and dynamically depending on the time of day and level of traffic. However, these schemes primarily concern themselves with *capacity*; none places traffic within a backbone so as to minimize delay *and* fit offered load. B4 [14] comes close, albeit in a private network environment in which the operator controls sources' demands. Indeed, as we explore in Section 2, 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.

In the remainder of this paper we expound upon the nature of the delay-minimizing routing problem for mesh-like backbones; identify goals for a routing system that solves this problem; discuss the practical design constraints that fall out of these goals, including optimal path selection and response to short-timescale traffic changes; and finally articulate open research challenges that the community must address to make low-delay routing on mesh-like backbones a reality.

2 Routing on Mesh-like Nets

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 [18], 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, but that reduce latency by offering more direct paths than a spanning tree. The value of f is 0 for a tree topology; a rectilinear 2D grid topology's value will be close to 0.5. Figure 2 shows a CDF of f across backbones.

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

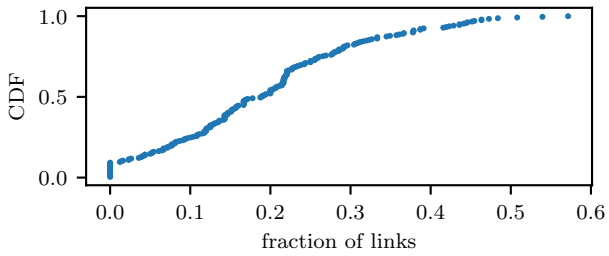


Figure 2: CDF of the fraction of links that, when removed from each topology, converts it into a spanning tree (237 topologies).

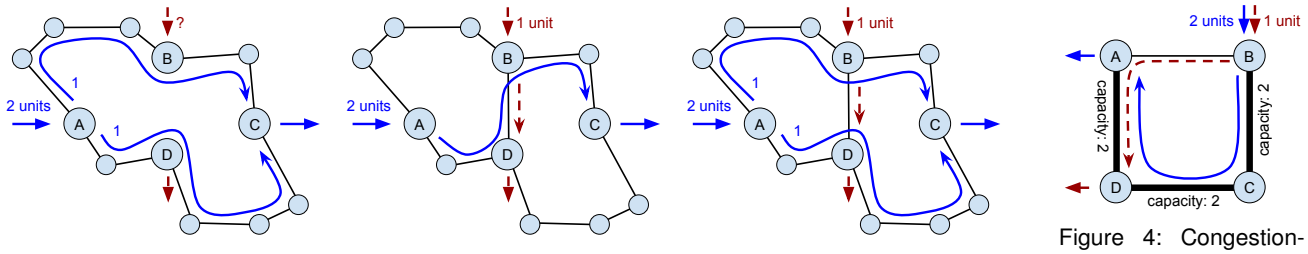
Most of the backbones in this dataset are not mesh-like: half of them are rendered trees by removing fewer than 20% of their links. Moreover, it is likely that a significant fraction of the topologies in the dataset include virtual links, which exaggerate a topology's "meshiness." Few topologies approach 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.

2.1 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 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 currently deployed, and state-of-the-art routing systems like B4.[2] 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.*

[2] In this paper we assume automatic bandwidth allocation for MPLS-TE, as it is often used in practice [29, 30].



(a) Initial scenario. **(b) Minimal propagation delay; congestion at A.** **(c) Longer propagation delay; no congestion.**

Figure 3: Simple scenario drawn to scale; all links have capacity of one unit. SP routing, MPLS-TE, and B4 exhibit congestion when a new link is added whether cost is hop count or propagation delay.

Figure 4: Congestion-free solution unattainable by SP routing regardless of weights; thick lines have 2x capacity.

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 3a. All links have unit capacity; arrows denote aggregates. Let us assume that the operator has set link weights either to all be equal or to be proportional to propagation delay in an effort to minimize latency. Regardless of which of SP routing, MPLS-TE, or B4 one runs on the topology in Figure 3, one obtains exactly the same result.

Initially, as shown in Figure 3a, suppose there are flows from $A \rightarrow C$ totaling 2 units of demand, which the routing spreads among the two equal-cost paths $A \rightarrow B \rightarrow C$ and $A \rightarrow D \rightarrow C$. 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 $A \rightarrow D$ and $B \rightarrow C$. 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 3b, the new link causes congestion in seemingly unrelated part of the network—at A.

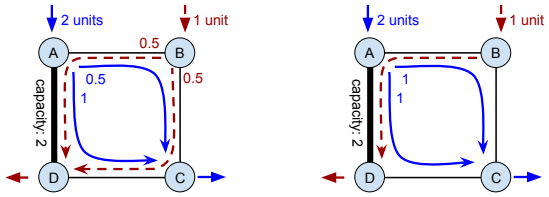
Why should *adding* capacity cause congestion? Because the new link's delay is low, its provisioning reduces the delay of the shortest path for the $A \rightarrow C$ aggregate. Any greedy SP-based routing scheme will thus dutifully place all of the $A \rightarrow C$ flows on the $A \rightarrow D \rightarrow B \rightarrow C$ path. Doing so saturates both links $A \rightarrow D$ and $B \rightarrow C$. The reverse path—$D \rightarrow B$—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 $A \rightarrow C$, 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 3c. This placement of traffic avoids congestion by keeping only traffic for $B \rightarrow D$ on the direct link, and spreads $A \rightarrow C$ 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 3b) and choosing longer-delay paths, and hence increasing propagation delay (as in Figure 3c), we posit that the routing system should avoid queuing if the topology as a whole allows doing so. While prior work outlined an attempt at trading off propagation delay and queuing delay [11], 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.

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 [13, 14]. 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 4 we discuss the challenges associated with coping with each aggregate's variability, and the consequences should the routing system fail to do so adequately.

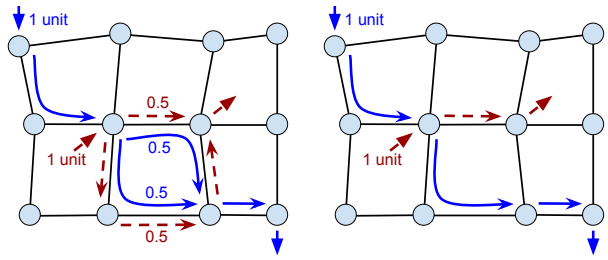
2.2 Greedy Routing and Varied Link Capacities

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 5, all links have the same delay and the same unit capacity, except for $A \rightarrow D$, whose capacity is 2 units. In this case greedy SP routing starts by filling each aggregate's shortest paths evenly: $B \rightarrow C \rightarrow D$ and $B \rightarrow A \rightarrow D$ for the $B \rightarrow D$ aggregate and $A \rightarrow B \rightarrow C$ and $A \rightarrow D \rightarrow C$ for the



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

Figure 5: Greedy routing gets stuck in local minimum, fails to avoid congestion.



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

Figure 6: Greedy routing yields high delay on mesh-like networks.

$A \rightarrow C$ aggregate. It will assign 0.5 units of capacity to each one of these paths, at which point the $B \rightarrow C$ link will saturate. At this point the $B \rightarrow D$ aggregate's demand has fully been met, but there is no way for greedy routing to meet $A \rightarrow C$'s one more remaining unit of demand, as there are only 0.5 units of capacity left on the $D \rightarrow C$ link. This solution will result in persistent congestion at A. Figure 5b shows a solution that avoids congestion, though centralized greedy solutions such as B4 can't 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.

2.3 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 and 5 is that there is not enough path diversity for it to find alternative paths. Will making the network more mesh-like cure that problem?

Alas, even in networks with great path diversity, greedy SP routing exhibits undesirable behavior. In Figure 6 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 6a) 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 6b). In essence, greedy SP routing tends to "punish" local aggregates in propagation delay.

2.4 The Need for Non-Greedy Routing

In Figure 3 it is possible for an experienced network operator to artificially increase the cost of one or more links, in the spirit of [10], in order to nudge even basic single-path SP routing into finding the congestion-free solution from Figure 3c. 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 4. To see why, note that 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 4; 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 5 and 6 respectively.

These examples lead us to conclude that today's routing and TE systems can't place traffic onto a mesh-like backbone's multiple alternative paths so as to 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) [27] to correctly handle the scenario in Figure 4; and SDN-based forwarding [19] 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.*

An intra-domain routing system capable of providing low latency over mesh-like WAN backbones should:

- Minimize end-to-end delay within a provider's backbone;
- Avoid congesting links, which is at odds with delay minimization, given queuing;
- Let the operator add a link of any capacity anywhere in the network, with the certainty that if that link carries traffic, overall service will improve;
- Drive links at high utilization, splitting aggregates among multiple, unequal delay paths as necessary;
- Achieve all the above goals while coping with variability in per-aggregate demand;
- Be tractable in state, measurement overhead, and control traffic at backbone routers.

3 Solution Space

Two complementary approaches will help advance our goals. On the one hand, we can adapt the topology on the basis of demand—overprovisioning can be seen as demand-driven

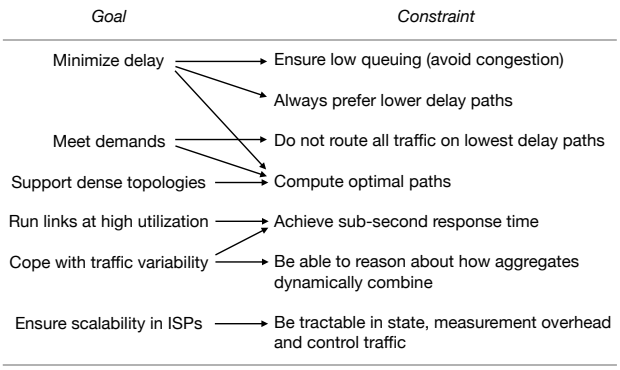


Figure 7: Constraints on the solution space induced by our goals.

topology adaptation at long timescales. Recently the research community has explored dynamic adaptation of the topology using reconfigurable optical devices; thus far, this work has focused on completion times for bulk transfers [15]. On the other hand, we can adapt routing dynamically as demand changes—such approaches optimize the paths that aggregates take within a fixed topology.

A demand-fitting, delay-minimizing routing system might combine both these approaches. We focus herein on the latter, which is usable with or without the former.

3.1 Overview

Figure 7 summarizes how the goals articulated in the previous section constrain the design space of possible solutions.

What might a routing system subject to these constraints look like? Given SDN and MPLS, we today have the low-level primitives to enable low-latency routing on mesh-like networks; how can we use these primitives effectively?

To pack as much traffic as possible onto the lowest-delay paths, and route the rest on longer paths, we must first have a good estimate of the demand between each backbone ingress and egress. This itself is a challenge, as we discuss later.

Given a topology (including link delays) and estimates of demands, our ideal routing system should then optimize the allocation of demands over paths, so as to offer low delay to users. Provided that every demand can be represented by a single estimated value, it is relatively straightforward to cast fitting traffic with minimal delay as an optimization problem, and use standard solvers to solve it, in the vein of Bertsekas [5], with link costs proportional to link propagation delay instead of link utilization. If we add constraints that the demands must fit, and optimize subject to these constraints, we have the essence of a dynamic centralized routing system that can avoid the local minima discussed in Section 2. Today's linear optimization software is fast enough to be able to cope with millions of possible paths in networks with thousands of links, and find an optimal solution in a few seconds.

Unfortunately, formulating this optimization problem is the easy part – and there already exist TE-oriented frameworks that can help us solve the problem [12]. Building a *practical* routing system requires making choices about how to balance mutually contradictory constraints in Figure 7. In the following, we discuss how these constraints trade off against each other, and delve into associated challenges the research community must address. Only then will it be possible to build low-latency, mesh-like networks that run at high-enough utilization to be economically viable.

3.2 Trade-Offs

Nearly any constraint in Figure 7 can be relaxed in the interest of better supporting another constraint. For example, Figure 3 provides a concrete example of how to balance between the second and the third constraints by suffering longer propagation delay to avoid congestion. Doing so is easy if we know precise traffic demands—an optimizer can minimize for total per-flow delay without causing persistent congestion by avoiding overcomitting any link. However, traffic demands are not constant, nor are they precisely known, so any such routing system needs to leave some capacity unallocated on even the busiest links if queuing is to be avoided.

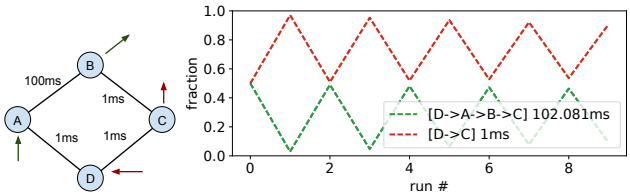
Thus there is a trade-off between high link utilization (which allows more flows to achieve lower propagation delay), and responsiveness to demand changes. If a system can respond rapidly to traffic changes, it can run links at higher utilization without risking excessive queuing delay. However, responding rapidly incurs more control traffic, which may limit scalability. Conversely, if the operator can tolerate lower utilization (more "headroom") on low-delay paths, demand increases can initially be absorbed by this spare capacity.

These trade-offs also impact a low-delay routing system's architecture. In a fully centralized system, one can ensure optimality by notifying the controller every time an aggregate's demand changes even slightly. This approach, however, has intrinsic limitations in responsiveness if the controller is geographically distant from the devices it manages, and in scalability owing to measurement and control traffic overhead. An alternative is to delegate some autonomy to network switches, which can respond more rapidly. For example, switches might autonomously react to significant demand changes, using heuristics to reason about how aggregates combine and which paths ensure low queuing, without always contacting the central controller. Such a hybrid distributed/centralized approach is more scalable and can respond more rapidly when acting to avoid congestion upon changes in demand. However, such heuristics come at the expense of optimality, at least until the central controller learns of demand changes and re-optimizes.

4 Research Agenda

We now outline a research agenda focusing on central challenges in achieving low-delay routing on mesh-like networks.

Determine how aggregates combine. Most TE work uses the mean traffic level as an indicator of whether or not persistent queues will form—when the mean level exceeds a link's capacity, traffic will be dropped. However, a latency-minimizing system that runs some links at high utilization must avoid not only persistent queues, but also transient ones.



(a) Topology **(b) Oscillation**

Figure 8: Oscillation with B4—all links have the same capacity, all aggregates the same initial demand.

To do so, it needs to allocate extra headroom to those links whose aggregates are more variable. The challenge is twofold:

- Measure aggregates so as to capture their variability.
- Perform optimization that is aware of each aggregate's variability, not only its mean traffic level.

The simplest way to capture variability is to periodically read traffic counters. The frequency depends on flows' RTTs: in a data center, one would need to query multiple times each millisecond to get a meaningful estimate. Luckily, in WANs, most flows have an RTT on the order of tens of milliseconds, so reading counters multiple times per second should be enough. Recent advances [8] and proposals [22] demonstrate that this is becoming possible at scale.

Given this extra information, the controller needs to predict how aggregates will *combine—i.e.,* how the traffic levels of aggregates that share links will sum at any time. Incorporating aggregates' variability into the optimization process itself is an open challenge. To do so, one would need to represent every demand with a *range* of values, representing the demand's statistical distribution of traffic levels over time (rather than a single value as assumed in Section 3). Considering distributions of values for every demand, however, causes path optimization to become a non-linear problem, which is computationally intractable. An alternative is to independently optimize path delays and check how aggregates combine, in separate steps. The downside is that a circular dependency is inherent to this two-step process. The system needs to know which aggregates combine well in order to assign them to paths, but it will not know how aggregates' various distributions multiplex before it produces an assignment.

Guarantee stability. Given a set of input demands, the routing system should eventually settle on a stable assignment of traffic to paths. Unfortunately, delay-minimizing routing systems cannot guarantee stability in general.

As already discussed, any delay-minimizing routing system has to 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 its throughput—*e.g.* when (i) its flows are competing with flows on a bottleneck link outside the network (as the TCP throughput equation tells us [26]), (ii) the traffic is generated by delay-sensitive applications, or (iii) delay-based congestion control is applied [7]. 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 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 which affect distributed routing [4, 17].

Figure 8 demonstrates a simple example where these oscillations occur with B4 (but are likely to happen with any load-dependent routing system). We perform ten runs of the TE component of B4, where after each run we modify each aggregate's demand, assuming its flows exhibit a linear dependency between throughput and delay—for example, if half of its flows move to a path that is twice as long, the aggregate's throughput is reduced by a third. Initially the $D \to C$ aggregate (whose paths are shown in Figure 8b) is split evenly among $D \to C$ and the longer $D \to A \to B \to C$. The flows on the long path then reduce their throughput, and on the next optimization pass B4 moves all of them back to the direct $D \to C$ link. The $A \to B$ aggregate also oscillates, but is not shown, as it always mirrors $D \to C$.

Even though in the real world flows are unlikely to exhibit such strong correlation between throughput and delay, characterizing and dealing with those oscillations 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.

Provide Internet-wide benefits. So far, we 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 optimization fit in the big 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 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, neighboring ASs often peer with each other in multiple places, with many potential egresses for an ingress. If path choice within each AS is a purely local decision, it is possible to end up with a sub-optimal end-to-end path.

We believe that interesting research directions can arise from studying 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.

While these challenges are substantial, we argue that building delay-minimizing routing systems that can dynamically leverage path diversity in richly connected, mesh-like topologies is essential to meeting users' demands for low latency in tomorrow's networks. This line of research has the potential to break the ossified status quo of tree-like topologies and shortest-path routing, moving the state of the art to a regime where flexible routing systems can extract benefit from complex topologies, and vice-versa.

REFERENCES

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

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

[3] D. O. Awduche and J. Agogbua. Requirements for traffic engineering over MPLS. *RFC 2702*, September 1999.

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

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

[6] I. N. Bozkurt, A. Aguirre, B. Chandrasekaran, P. B. Godfrey, G. Laughlin, B. Maggs, and A. Singla. Why is the internet so slow?! In *Proc. Passive and Active Measurements Conference*, March 2017.

[7] N. Cardwell, Y. Cheng, C. S. Gunn, S. H. Yeganeh, and V. Jacobson. BBR: Congestion-based congestion control. *ACM Queue*, 14(5):50, December 2016.

[8] A. R. Curtis, J. C. Mogul, J. Tourrilhes, P. Yalagandula, P. Sharma, and S. Banerjee. DevoFlow: Scaling flow management for high-performance networks. In *Proc. ACM SIGCOMM*, August 2011.

[9] A. Elwalid, C. Jin, S. Low, and I. Widjaja. MATE: MPLS adaptive traffic engineering. In *Proc. IEEE INFOCOM*, April 2001.

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

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

[12] R. Hartert, S. Vissicchio, P. Schaus, O. Bonaventure, C. Filsfils, T. Telkamp, and P. Francois. A declarative and expressive approach to control forwarding paths in carrier-grade networks. In *Proc. ACM SIGCOMM*, August 2015.

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

[14] S. Jain, A. Kumar, S. M, J. Ong, L. Poutievski, A. Singh, S. Venkata, J. W, J. Zhou, M. Zhu, J. Zolla, U. Holzle, S. Stuart, and A. Vahdat. B4: Experience with a globally-deployed software defined WAN. In *Proc. ACM SIGCOMM*, 2015.

[15] X. Jin, Y. Li, D. Wei, S. Li, J. Gao, L. Xu, G. Li, W. Xu, and J. Rexford. Optimizing bulk transfers with software-defined optical WAN. In *Proc. ACM SIGCOMM*, August 2016.

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

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

[18] S. Knight, H. X. Nguyen, N. Falkner, R. Bowden, and M. Roughan. The Internet topology zoo. *IEEE Journal on Selected Areas in Communications*, 29(9):1765–1775, 2011.

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

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

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

[22] J. C. Mogul and P. Congdon. Hey, you darned counters!: get off my ASIC! In *Proc. ACM HotSDN*, August 2012.

[23] T. P. Morgan. How Google wants to rewire the internet, July 2017. [Interview with Amin Vahdat; https://www.nextplatform.com/2017/07/17/google-wants-rewire-internet/].

[24] J. Moy. OSPF version 2. *RFC 2328*, April 1998.

[25] D. Oran. OSI IS-IS intra-domain routing protocol. *RFC 1142*, February 1990.

[26] J. Padhye, V. Firoiu, D. Towsley, and J. Kurose. Modeling tcp throughput: A simple model and its empirical validation. In *Proc. ACM SIGCOMM*, August 1998.

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

[28] A. Singla, B. Chandrasekaran, P. Godfrey, and B. Maggs. The internet at the speed of light. In *Proc. ACM Hotnets*, October 2014.

[29] R. Steenbergen. 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.

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

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