OneWAN is better than two: Unifying a split WAN architecture

Umesh Krishnaswamy, *Microsoft;* Rachee Singh, *Microsoft and Cornell University;* Paul Mattes, Paul-Andre C Bissonnette, Nikolaj Bjørner, Zahira Nasrin, Sonal Kothari, Prabhakar Reddy, John Abeln, Srikanth Kandula, Himanshu Raj, Luis Irun-Briz, Jamie Gaudette, and Erica Lan, *Microsoft*

https://www.usenix.org/conference/nsdi23/presentation/krishnaswamy

This paper is included in the Proceedings of the 20th USENIX Symposium on Networked Systems Design and Implementation.

April 17–19, 2023 • Boston, MA, USA

978-1-939133-33-5

ONEWAN is better than two: Unifying a split WAN architecture

Umesh Krishnaswamy[*], Rachee Singh[*†], Paul Mattes[*], Paul-Andre C Bissonnette[*], Nikolaj Bjørner[*], Zahira Nasrin[*], Sonal Kothari[*], Prabhakar Reddy[*], John Abeln[*], Srikanth Kandula[*], Himanshu Raj[*], Luis Irun-Briz[*], Jamie Gaudette[*], Erica Lan[*]

[*]*Microsoft,* [†]*Cornell University*

Abstract

Many large cloud providers operate two wide-area networks (WANs) — a software-defined WAN to carry inter-datacenter traffic and a standards-based WAN for Internet traffic. Our experience with operating two heterogeneous planet-scale WANs has revealed the operational complexity and cost inefficiency of the split-WAN architecture. In this work, we present the unification of Microsoft's split-WAN architecture consisting of SWAN and CORE networks into ONEWAN. ONEWAN serves both Internet and inter-datacenter traffic using software-defined control. ONEWAN grappled with the order of magnitude increase in network and routing table sizes. We developed a new routing and forwarding paradigm called traffic steering to manage the increased network scale using existing network equipment. Increased network and traffic matrix size posed scaling challenges to SDN traffic engineering in ONEWAN. We developed techniques to find paths in the network and chain multiple TE optimization solvers to compute traffic allocations within a few seconds. ONEWAN is the first to apply software-defined techniques in an Internet backbone and scales to a network that is 10× larger than SWAN.

1 Introduction

The large-scale commercialization of cloud computing led cloud providers to provision private wide-area networks (WANs). These initial deployments connected both datacenters and Internet peering edges of the cloud using a unified cloud WAN. For instance, Microsoft's cloud WAN, called the CORE (AS8075) network, interconnected Microsoft's datacenters and Internet peering edges. However, as the cloud workloads evolved, inter-datacenter traffic began to dominate, shrinking the capacity available for carrying Internet traffic to peering edges. In response, Microsoft built a second WAN to offload inter-datacenter traffic. This WAN, called the software-defined WAN or SWAN (AS8074) used software-defined traffic engineering (TE) and bandwidth brokering to achieve higher network utilization [14] than RSVP-TE [2] in CORE. Deployment of two cloud WANs (Figure 1), one for Internet traffic and the other for inter-datacenter traffic is an industry-wide trend with Google [17] and Meta [9] operating similar split-WAN architectures.

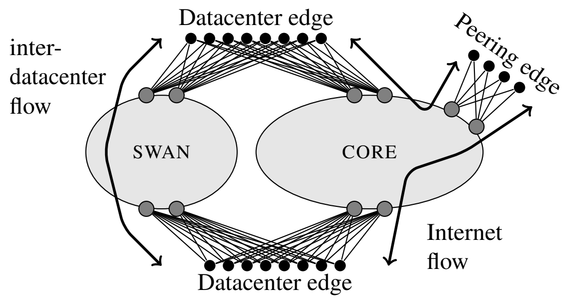


Figure 1: Before ONEWAN, there were two wide area networks, CORE (AS8075) and SWAN (AS8074). Datacenter edge connected to both networks, and peering edge only connected to CORE. Internet traffic was served by CORE and inter-datacenter traffic by SWAN. CORE used RSVP-TE and SWAN used SDN based traffic engineering.

Why is one WAN better than two? Maintaining two heterogeneous WANs specialized for inter-datacenter (SWAN) and Internet (CORE) traffic led to several operational challenges. First, the two-WAN architecture requires that datacenter edges connect to both SWAN and CORE routers (see Figure 1). The dual WAN connectivity from datacenter edges led to wasteful use of expensive network equipment and limited power supply. This problem was made worse by massive build outs of new datacenter regions and edge sites. Second, the split-WAN architecture makes capacity planning hard. At a given time, one WAN can be under-utilized while the other is over-utilized. Moreover, acquiring optical capacity for both WANs in every geographical region and building the required redundancy on each network, became prohibitively expensive. Finally, CORE and SWAN used routers with completely different protocol stacks. As a result, engineers were trained to configure, monitor and manage two distinct networks. Deploying new WAN sites took more time since different routers had to be deployed for the two networks.

In 2020, we observed a steady growth in Internet traffic from peering edges due to increased use of collaboration tools spurred by remote work (Figure 2). While the network capacity between Internet peering edges and the cloud WAN is scare and expensive, it was not in the purview of TE in the split-WAN architecture. Thus, it became important to engineer the growing traffic on WAN-facing links from peering edges but the key protocol of the CORE network, RSVP-TE, was not up to this task as it had reached scaling limits in our network.

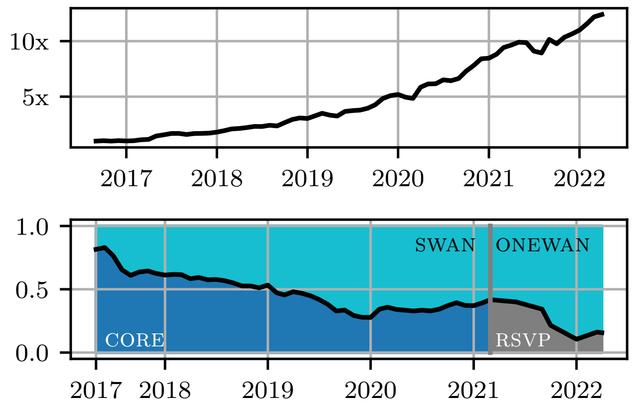


Figure 2: WAN traffic growth, the portions carried by CORE and SWAN when they were separate networks, and the portions of RSVP-TE and ONEWAN-TE traffic in the unified network.

Software-defined control in ONEWAN. Due to these operational challenges, we consolidated the split-WAN architecture into a unified ONEWAN. We decided to implement ONEWAN using SDN principles (like SWAN) instead of a standards-based approach (like CORE) for three main reasons. First, the key protocol of the CORE network, RSVP-TE, was reaching scale limits due the existing size of the CORE network topology. Second, RSVP-TE needed network-wide configuration changes which have a global blast radius. In contrast, we had deployed BLASTSHIELD to control the blast radius of faults in the SWAN network [22]. Finally, despite the earlier intention of using SWAN only for discretionary workloads, SWAN had evolved to carry mission-critical application traffic (*e.g.,* Azure, Bing, Office, and Teams) in addition to discretionary inter-datacenter workloads (*e.g.,* replication, backup). We measure our service level objectives (SLOs) as the daily average over percentage of successfully transmitted bytes in an hour. The SLOs of SWAN exceeded 99.999% for customer traffic, and 99.9% for discretionary traffic.

Challenges in evolving from SWAN to ONEWAN. In this work, we discuss the main technical challenges we overcame to unify the split-WAN architecture into ONEWAN (Table 1):

Increased routing table sizes. SWAN was responsible for routing datacenter prefixes only, which were few enough for all SWAN routers to run BGP and store the datacenter routing tables in the router memory. ONEWAN routers had to contend with Internet routing tables and it would be cost prohibitive to make every ONEWAN router hold the entire Internet routing table. Hence, ONEWAN assigned two roles to routers: (1) aggregation routers that hold full IP routing tables, and (2) backbone routers that act as forwarding only nodes that do not run BGP. We develop a new SDN function in ONEWAN called *traffic steering* on aggregation routers to encapsulate WAN packets with information needed by backbone routers to do TE without IP routing (§ 3).

Fast failure repair. One of RSVP-TE's strengths is *fast reroute*, which enables it to switch from primary to backup paths within milliseconds of a failure. This fast convergence is essential for performance-sensitive services like video streaming and virtual desktop over the WAN. *Local repair* is SWAN's equivalent function of detecting and repairing failed paths using agents that run on the routers. Improving convergence times in ONEWAN required significant enhancements and was an area of new learnings (§ 4).

Scaling TE optimization. ONEWAN has ten times the number devices of SWAN. The traffic engineering optimization techniques used in SWAN had to scale to a network size that is ten times larger. We developed scalable path and linear programming optimizations to deal with the increased scale of traffic engineering in ONEWAN (§ 5).

Estimating Internet traffic matrices. Accurate estimation of traffic matrices is crucial for engineering Internet traffic. Existing mechanisms for traffic matrix estimation fell short for ONEWAN since they did not contend with Internet-facing traffic. We developed a scalable pipeline for accurate traffic matrix measurement for ONEWAN (§ 6).

Hitless transition in the live network. Finally, the transition to ONEWAN was done in the live cloud network as it continued to carry user traffic. We devised techniques to enable both SWAN and CORE networks to undergo a hitless transition to ONEWAN (§ 7).

2 ONEWAN Architecture

The consolidation of SWAN and CORE networks into ONEWAN was a large undertaking that took years of engineering effort in planning, testing, implementing and verifying the new WAN architecture. In this section we motivate the design choices that led to the ONEWAN architecture.

Figure 3 shows a simplified view of ONEWAN with two datacenter regions and two edge sites. Datacenter regions are large campuses with routers at the root of the datacenter network connecting to aggregation routers in regional network gateways (RNGs). RNGs connect multiple datacenters in a geographical region with a maximum fiber distance under 100 kilometers. Backbone routers are present in all RNGs and additional transit gateways that are hubs for long-haul optical links. Peering edge routers at edge sites connect to aggregation routers in the same site. Aggregation routers connect to backbone routers in RNGs and gateways. ONEWAN traffic engineering (ONEWAN-TE) applies to inter-datacenter flows between datacenter regions, and Internet flows between edge sites and datacenter regions. The set of traffic engineered links consists of all backbone–backbone and aggregation–backbone links. The Clos interconnect between edge and aggregation

Challenge	**Techniques used**
Route scale increase	Only aggregation routers hold full IP routing tables. Controllers add routes for BGP next hops instead of BGP prefixes (§ 3).
Avoid costly new builds	Develop ONEWAN agents for four firmware versions to cover all existing routers. Interconnect CORE and SWAN with aggregation routers (§ 3 and § 4).
TE optimization scale increase	Traffic matrix-aware path computation. Optimize LP solvers (§ 5.1 and § 5.2).
Route convergence time	Fast local repair using diverse backup paths. Tunnel liveness probes that return to sender using controller routes (§ 4 and § 5.3).
Measuring real-time traffic matrix	Use IPFIX sampling with a high throughput pipeline. Anycast source-specific destinations determined from IPFIX flow records (§ 6).
Minimize risk of outages	Divide ONEWAN into geographies managed by separate controllers. Steering routes control what traffic is migrated ([22], § 7).

Table 1: Summary of ONEWAN challenges and our approaches for solving them.

routers is always intra-site or intra-campus, is built to high capacity, and hence is not a part of traffic engineering.

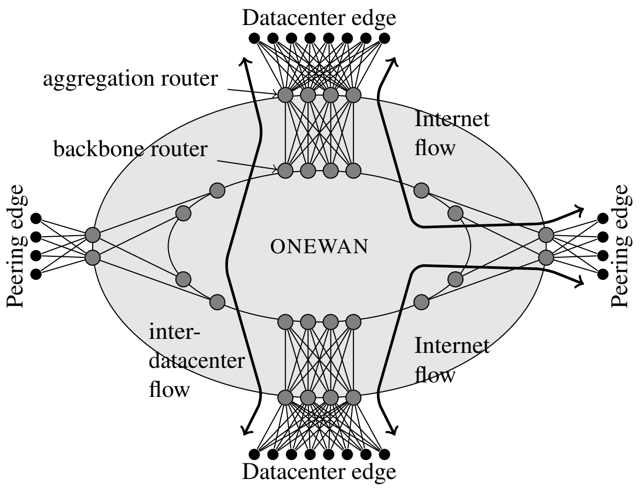
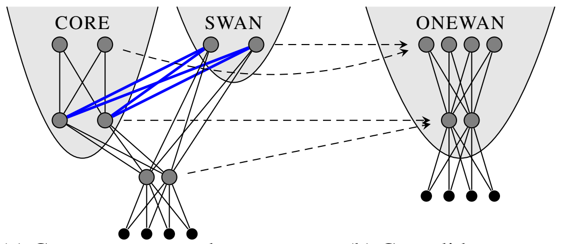


Figure 3: ONEWAN is a unified network that serves Internet and inter-datacenter flows using SDN TE. ONEWAN consists of aggregation routers at its boundary and backbone routers in its interior (gray dots). Datacenter and peering edge routers (black dots) connect to aggregation routers using a Clos interconnect. ONEWAN-TE applies to any inter-site flow between aggregation routers, be they inter-datacenter or between edge site and datacenter.

In-place conversion to ONEWAN. Since both CORE and SWAN networks carry mission-critical customer traffic, the unification of the two networks had to be done *in-place* with no visible impact to client performance. This goal necessitated incremental changes to both physical connectivity and network configuration to unify SWAN and CORE networks. The first step for the unification was to physically connect CORE and SWAN. Figure 4 shows new links that connect CORE aggregation routers to SWAN backbone routers. When CORE and SWAN were separate networks, they each had a set of aggregation routers. In the second step, we eliminated one set of aggregation routers and their links to save power. We merged the routing domains of the interior gateway protocol, which IS-IS [16] in CORE and SWAN, but did not merge the BGP autonomous systems of the two networks to allow each network to carry its original traffic.



(a) Connect CORE and SWAN (b) Consolidate aggregation

Figure 4: (a) Physically connect CORE and SWAN using aggregation routers (shown as thick lines). (b) Consolidate two sets of aggregation routers into one.

Leverage existing network hardware. Our goal was to repurpose the WAN routers in SWAN and CORE networks as opposed to building a clean-slate ONEWAN from the ground-up with an entirely new fleet of routing hardware to reduce the capital expenditure in the consolidation process. SWAN had $O(100)$ routers and $O(10^5)$ datacenter routes while CORE had $O(1000)$ routers and $O(10^6)$ Internet routes. To leverage existing hardware for the increased scale in ONEWAN, only aggregation routers run BGP with the full Internet and datacenter routing tables. This allows the remaining backbone routers to be simpler commodity switches with smaller routing table memory.

Choice of tunneling mechanism. ONEWAN uses Multiprotocol Label Switching (MPLS) [31] to transport IP packets in a tunnel across the backbone. We note that ONEWAN does not use any TE protocols like RSVP-TE or TE extensions in the interior gateway protocols like IS-IS. We use MPLS for its efficient implementation of label stack encapsulation on existing routers in SWAN and CORE. Our approach could be generalized to other tunneling abstractions though it is outside the scope of this work.

Router roles. Aggregation routers implement a key function in ONEWAN, called traffic steering, described in detail in § 3. Aggregation routers are also used as transit routers in ONEWAN-TE tunnels. This enables the large number of legacy devices in CORE and SWAN to make full use of available bandwidth in either network. Edge routers are sources or sinks for traffic. They are either datacenter routers or peering routers. These routers continue to perform the same function before and after the consolidation of SWAN and CORE.

3 ONEWAN Traffic Steering

Routing in ONEWAN occurs in three parts (see Figure 5). First, a *steering route* on the aggregation router encapsulates IP packets entering ONEWAN. Second, backbone routers forward packets received from aggregation routers along traffic engineered tunnels. In the final step, the egress backbone router uses a segment routed [10] path to route packets to the network destination. ONEWAN-TE controller computes and ONEWAN router agents program traffic steering and engineering routes on routers. IS-IS updates the segment routes.

Steering routes. BGP on aggregation routers receives routes announced by BGP route reflectors or its clients and chooses a set of one or more equal-cost BGP next hops for each prefix. The BGP next hops are typically aggregation routers at network egress sites though they could also be endpoints a few hops beyond the network egress site in legacy portions of ONEWAN. All ONEWAN sites are assigned a static identifier called the *site label*. When BGP looks up the route for the next hop, the highest preference route is the steering route added by the ONEWAN-TE controller. The steering route pushes a stack of two MPLS labels onto packets entering the backbone. The top label in the stack is the egress site label. The bottom label in the stack is the segment routing node segment identifier (node SID) of the BGP next hop learned from IS-IS.

The egress site label refers to the backbone exiting site on the shortest path to the packet's destination. It is the same site as the egress aggregation router, though it can be different in legacy portions of ONEWAN. Ingress backbone routers use the egress site label to determine the set of traffic engineered tunnels to use. IP flows are weighted load balanced to a specific tunnel based on their 5-tuple and traffic class. Once the tunnel is selected, the ingress backbone router swaps the egress site label with the traffic engineered tunnel label.

The bottom label serves two purposes. First, it provides forwarding from the egress backbone router towards the endpoint. Second, it provides a fallback if no traffic engineered tunnel for the egress site is up due to failures. When the ingress backbone router has no operationally up traffic engineered tunnels to a particular egress site, ONEWAN agent automatically adds a route to pop the egress site label and forward the packet using the segment route for the BGP next hop node SID. The advantage of this design is that failures in the network are quickly and transparently handled by the routers without immediate intervention of the controller.

Steering route weights. The steering route sprays packet flows to connected backbone routers using unequal load balancing. Although aggregation routers are directly connected with equal capacity to backbone routers, each backbone router is not an equal choice for ingress. For example, a backbone router may have a longer path to the endpoint or may have less available bandwidth to an egress site. Latency increases in the former and congestion can occur in the latter. The ONEWAN-TE controller excludes backbone routers with the shortest path latency from the ingress aggregation router to the egress site exceeding the best latency by a threshold. It then calculates weights using single commodity maximum flow from the ingress aggregation router to the egress site. Weights are recalculated whenever the topology changes. Figure 6 illustrates the weight calculation for flows from aggregation router a to endpoint f. Both backbone routers b and c are used to spread the load because they have similar shortest path latency to the egress site. The weight to b is 33% because the maximum flow bandwidth of $a-b-T$ is proportionately less.

Why have two stages of traffic splits? ONEWAN calculates traffic steering splits using single commodity max-flow and calculates traffic engineering splits using priority max-min fairness optimization (§ 5.2) for the following reasons. We were concerned that the TE optimization and path computation algorithms may not scale to the full size of ONEWAN and operating on a subgraph of backbone routers would ease the scaling challenges. Moreover, aggregation routers are connected with high capacity links to backbone routers and so do not need traffic engineering. Finally, we wanted steering routes to be updated quickly in case of failures and did not want the updates to be slowed down by an optimization phase.

In hindsight, the two traffic splitting mechanisms in ONEWAN can be unified since our improvements to the TE optimization (§ 5) enable it to handle the full ONEWAN topology of backbone and aggregation routers. Aggregation routers are transits for ONEWAN-TE tunnels between CORE and SWAN devices and so have to be part of TE optimization. ONEWAN agents in aggregation routers react to network topology changes faster than the controller.

Segment routing at the egress backbone router. TE tunnels terminate at the backbone router instead of the aggregation router for a separate reason. Segment routing implementations on vendor routers only allow penultimate hop popping, meaning that the penultimate router must remove the node SID before delivering the packet to the intended node. Routers do not easily support popping a label stack. This necessitated at least one segment routed hop and is why the TE tunnel is between backbone routers. Support for ultimate hop popping would eliminate the last segment routed hop.

4 ONEWAN Agent and Local Repair

ONEWAN agents are responsible for installing routes provided by ONEWAN-TE controllers. ONEWAN agents also perform *local repair*. The local repair mechanism detects

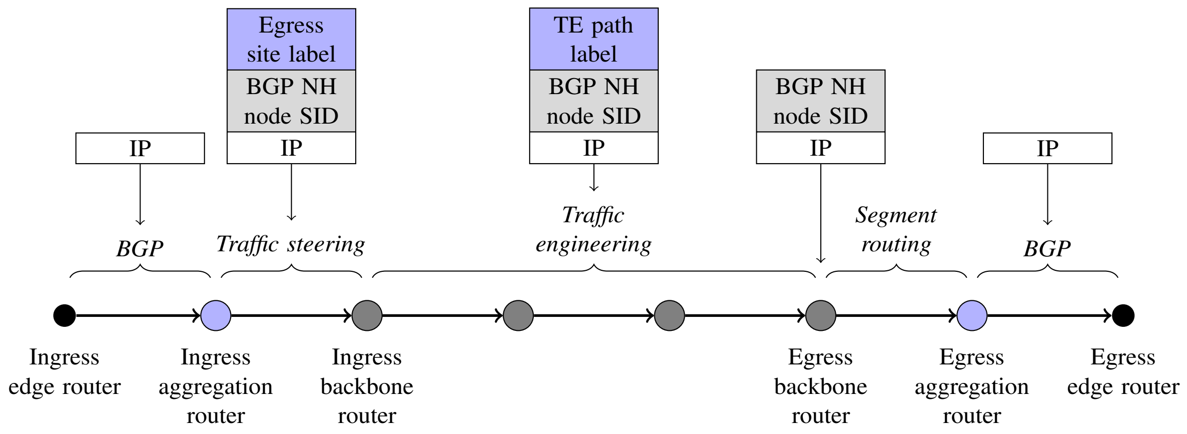
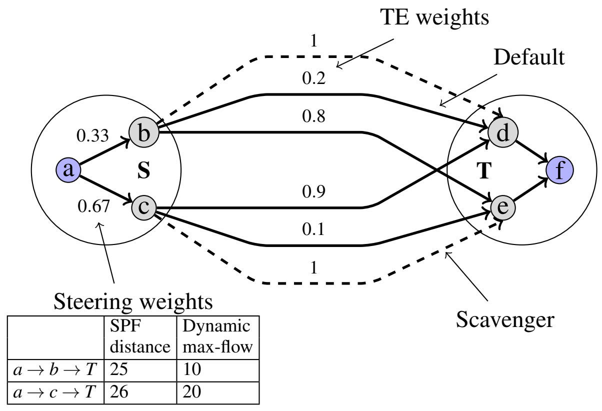


Figure 5: ONEWAN routing occurs in three parts. The first part is the steering route in the aggregation router, the second is the traffic engineered tunnel between backbone routers, and the third is the segment routed path from the egress backbone router to the network egress point. The traffic steering and engineering routes are added by the ONEWAN-TE controller.



	SPF distance	Dynamic max-flow
$a \rightarrow b \rightarrow T$	25	10
$a \rightarrow c \rightarrow T$	26	20

Figure 6: ONEWAN-TE example of flows from aggregation router a to endpoint f. Steering route load balances flows to selected backbone routers based on shortest-path distance and maximum flow from the backbone router to the egress site. The backbone routers perform full traffic engineering optimization.

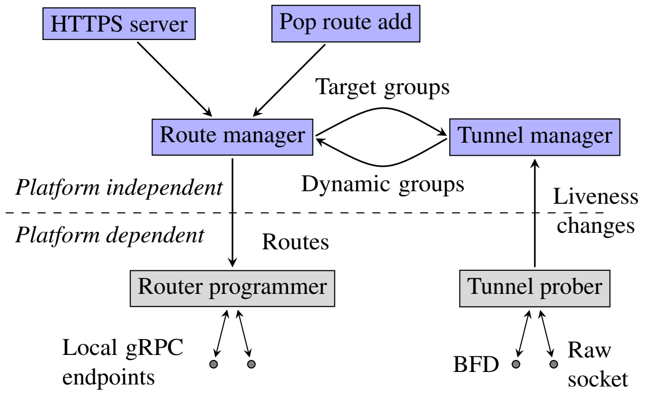


Figure 7: ONEWAN agent has platform dependent and independent components. The agent installs routes provided by ONEWAN-TE controller and switches off tunnels experiencing forwarding faults.

the forwarding state of tunnels and reprograms actions to send traffic on surviving tunnels thereby minimizing transient packet loss due to route blackholes or congestion.

The ONEWAN agent runs as a process on all aggregation and backbone routers, optionally inside a Docker container. It is supported on four different firmware operating systems. To support the heterogeneity of firmware, the agent is structured as separate platform-independent and platform-dependent components, with well-defined APIs between them. The platform-independent portion has the bulk of the complexity in the agent. Figure 7 shows the agent organization.

Route programming. ONEWAN agents communicate with ONEWAN-TE controllers using an HTTPS server; no routing protocol is required. We use OpenFlow [28] match actions and groups to represent routes. Groups represent the set of traffic steering and engineering tunnels originating at ingress routers, tunnel weights for unequal load balancing, traffic class to indicate what type of traffic the tunnel is meant for, whether the tunnel is primary or backup, and attributes for probing tunnel liveness. Transit routes use unary actions without groups.

The *route programmer* implements dynamic route operations through an internal gRPC or equivalent connection provided by the router firmware. It converts groups to weighted cost multipath (WCMP) next hops with 32 members and duplicates each tunnel in proportion to its weight.

The *target group* is the set of tunnels received from the controller and the *dynamic group* is the set of tunnels that are alive, with original weights redistributed among them. When a group has no primary tunnels alive, the backup tunnels are elevated to primary. A tunnel is associated with a traffic class, default matching any traffic class. When a group has no tunnels of default traffic class alive, the tunnels of the next traffic class are modified to be the default traffic class. In Figure 8, the target group has six tunnels A-F. The backup tunnels E and F become primary when A and B fail. If A comes back up, it is the sole primary default tunnel. We explain how ONEWAN-TE calculates diverse backup and class-aware tunnels in § 5.2 and § 5.3.

If no tunnels are alive, the agent replaces the dependent egress site label routes with pop-and-forward (on a backbone router) or removes the dependent BGP next hop steering routes (on an aggregation router). Part of agent initialization is the creation of pop-and-forward routes for the entire allocated egress site label space, to cover any stray traffic received.

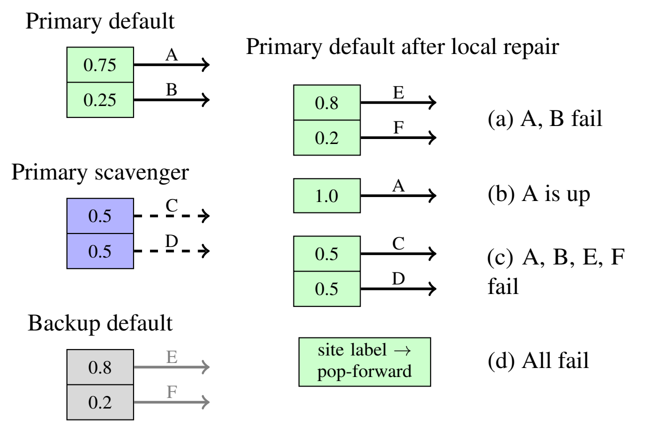


Figure 8: Local repair. The ONEWAN agent automatically adjusts the composition and weights of tunnels based on liveness. (a) If primary default tunnels A and B fail, then backup tunnels E and F become primary. (b) If A comes back up, it is the sole primary default tunnel. (c) If only scavenger tunnels C and D survive, they become primary. (d) If all tunnels fail, the site label route is replaced with a pop-and-forward action.

Tunnel probing. Aggregation routers use single-hop tunnel probing using Bidirectional Forwarding Detection [21] to check the links to backbone routers. Ingress backbone routers perform end-to-end tunnel probing using a labeled self-ping mechanism via a raw socket. Tunnel probes use the same routes as data packets in the forward direction.

Tunnel probes in SWAN relied on IS-IS in the return path. This caused backup tunnels to fail even though they did not use failed links in the forward path because the probe return path was affected by IS-IS route convergence. Without primary and backup tunnels, the agent removed the controller route and traffic reverted to IS-IS. Client traffic experienced IS-IS convergence times and congestion losses as long as the tunnels were still down. This significantly degraded user experience. Therefore, probes in ONEWAN return from the tunnel destination to the tunnel source using controller routes. The return hops are the reverse of the forward hops and thus do not share fate with links unrelated to the data tunnel. In Figure 9, the probe packets return on $d-b-a$, not $d-c-a$ even though the latter is shorter. The ONEWAN-TE controller reuses available data tunnels in the reverse direction when possible, and creates new tunnels otherwise.

Multiple probe packets can be in flight and the probing interval is independent of the path round-trip time. A loss of a configured number of probes marks a tunnel down, and a successful probe marks a tunnel up. We send probes at 100ms

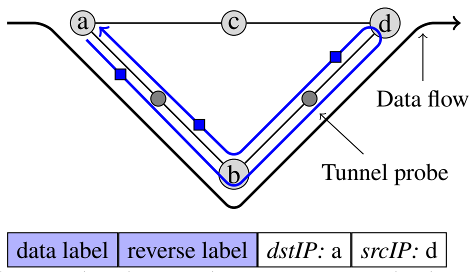


Figure 9: Tunnel probes use the same routes as the data packets in the forward direction and return from the destination using the same links to avoid false failures. The transit routes are programmed by the controller and agents of transit routers.

intervals and mark the tunnel down after loss of 3 probes. Fault detection time is thus 300ms plus the distance to the fault which in the worst case is the tunnel latency.

Local repair. In the split-WAN architecture, Internet traffic was handled by RSVP-TE in the CORE network. Standards-based RSVP-TE in the CORE network implements a fast reroute (FRR) [29] mechanism that allows it to recover from link failures in $O(10)$ ms. Fault detection time is the distance to the fault, which in the worst case is the link latency, plus a small delay for the optical transponder to notify the router. FRR switches to precomputed bypass tunnels at the point of fault and runs in or near the line card network processing unit. Switching times are a few milliseconds.

In contrast with FRR, ONEWAN's local repair happens at the ingress router of the tunnel not the point of fault. Faults in the first hop link are detected by interface down events and faults in subsequent hops are detected by lost probes. Repair is initiated in the route processor and subject to greater inter-process communication and scheduling delays. The route programmer modifies the WCMP in place to decrease the number hardware writes performed at the time of repair. As a result, ONEWAN convergence time is under one second. Although slower than FRR, ONEWAN meets the convergence time requirements of video streaming, video conferencing and other interactive applications currently served by the network. ONEWAN tunnel probes also validate forwarding because they exercise the routes used by data packets. The backup tunnels are chosen based on diversity and residual bandwidth and hence experience less transient packet loss due to congestion (described in § 5.3).

Route programming. ONEWAN is divided into geographical regions and regional ONEWAN-TE controllers program routes on devices in their region [22]. All routers program steering routes in parallel, in a single phase. The pop-and-forward routes on the backbone routers eliminate the need to synchronize programming steering and TE routes.

TE routes are also updated in parallel, using three phases with a barrier between each. A *make-before-break* sequence ensures that no route blackholes or loops form during programming. The set of routes for all routers is logically divided into two sets: the transit and tunnel egress routes T, and the

Metric	ONEWAN-TE
Per-device steering + TE routes	$O(10^3)$
Network level tunnels	$O(10^4)$
Per-device FIB update (p95)	2.0 sec
Network level FIB update (p95)	3.7 sec

Table 2: ONEWAN-TE scale in terms of routes, tunnels, and update times per device and for the entire network.

tunnel ingress routes G. Since the G routes depend on the T routes, make-before-break ensures that traffic cannot enter a tunnel until each subsequent hop has been programmed. An important property of ONEWAN-TE route generation is that the label spaces between successive iterations do not overlap, except where the paths they represent are identical. The label range is large enough to allocate unique labels in the worst case.

The phases of replacing the routes of iteration n, $T_n \cup G_n$, with routes of iteration $n+1$, $T_{n+1} \cup G_{n+1}$, are as follows:

- The initial state in all routers is $T_n \cup G_n$.
- $T_n \cup T_{n+1} \cup G_n$ is sent to agents which perform their route update and report success or error.
- $T_n \cup T_{n+1} \cup G_{n+1}$ is programmed. During this phase there may be a mix of G_n and G_{n+1} routes in the network, but all the T routes they depend on will be present.
- $T_{n+1} \cup G_{n+1}$ is programmed.

Traffic shifts in the second phase. Since a phase completes in under 4 seconds, a moderate scratch capacity avoids transient congestion. We reserve 15% scratch capacity to handle transients from programming and traffic microbursts. If an agent reports an error or connection is lost, the controller rolls back to the initial state in a single phase. Any inconsistencies after the rollback are corrected by the ONEWAN agent local repair.

4.1 Evaluation

Route scale. The number of traffic engineering tunnels was a significant scaling issue with RSVP-TE in CORE. Large number of RSVP-TE tunnels increased network convergence time after link failures and exceeded hardware resources in older aggregation routers. Table 2 shows that ONEWAN-TE uses 10 times fewer tunnels than RSVP-TE. When ONEWAN-TE optimizes, it simply reserves more bandwidth in existing tunnels, or creates new and destroys unused tunnels. On the other hand, RSVP-TE signals new tunnels with incremental bandwidth reservation, which it combines less frequently. Second, RSVP-TE requires a full mesh of label switched paths between nodes, but ONEWAN-TE only creates tunnels for nodes within a geography, and inter-geography flows reuse the intra-geography tunnels.

Steering routes scale with number of endpoints, and traffic engineering routes scale with number of backbone nodes. Both forwarding information bases of routes (FIB) have small sizes even for a large network. Time to update the FIB is affected by the round-trip distance between the controller and router, and the number of routes in the FIB (see Figure 10).

Class based forwarding. An objective of ONEWAN-TE is to use underutilized links on longer paths for replication or backup traffic which are marked as scavenger traffic class but use diverse shortest paths for best-effort and higher traffic classes. ONEWAN agent installs the egress site label route with an intermediate policy lookup that is indexed by traffic class to change from default class WCMP to a class specific WCMP. Figure 11 shows that ONEWAN-TE assigns 55% of scavenger traffic to longer paths. Differentiated paths decrease scavenger drops due to microbursts in higher traffic classes. When best-effort and scavenger queues use weighted round-robin queue scheduling, differentiated paths also decrease best-effort transient drops due to scavenger microbursts. Figure 12 shows the successful transmission rate SLO for best-effort and scavenger traffic classes for a 3-month period in 2022.

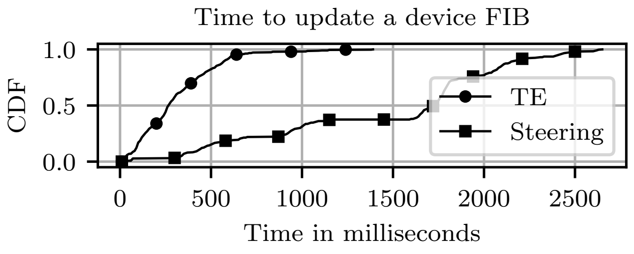


Figure 10: Per-device FIB update times for steering and TE routes.

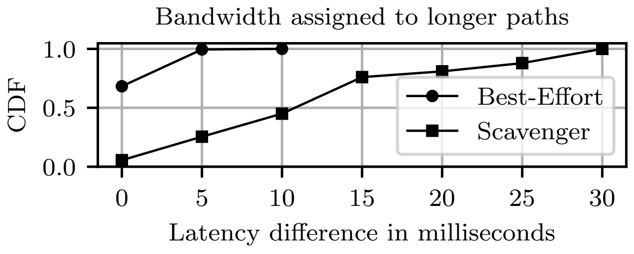


Figure 11: Class based forwarding uses paths for scavenger traffic class that are different from best-effort and higher classes. Longer paths carry 55% of scavenger bandwidth.

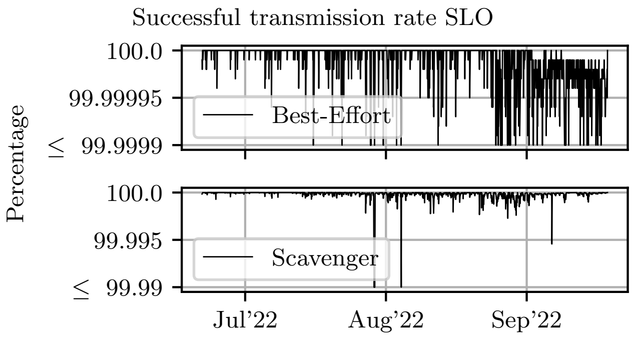


Figure 12: Successful transmission rate SLO for best-effort and scavenger traffic classes for a 3-month period in 2022.

5 Traffic Engineering Optimization

The ONEWAN TE optimizer has two inputs – predicted traffic matrix (TM) (described in § 6) and dynamic topology. A traffic trunk is an aggregate traffic flow from a source backbone router to a destination site for a specific traffic class, and the traffic matrix is a collection of predicted bandwidths for traffic trunks. The dynamic topology consists of sites, nodes and links. Nodes and links have tens of different attributes, including interface addresses, device role, link operational bandwidth, bandwidth reserved for RSVP-TE, link metric, whether a link or node should be avoided due to maintenance activity, and link reliability information. Each node is associated with a site, and all nodes in a site are equivalent destinations for a traffic trunk destined to the site. The TE optimizer operates in two phases: path computation phase and optimization phase.

In the path computation phase, we perform online computation of paths on the dynamic topology for all traffic trunks using efficient path finders (§ 5.1). We compute enough paths to enable the optimization phase to have adequate choices when allocating traffic.

In the optimization phase, the priority fairness optimization solver (§ 5.2) allocates traffic trunks to paths using the path formulation of multi-commodity flow problem. The TM is divided based on the traffic class of trunks and each traffic class is optimized differently. The priority fairness solver allocates demands of high priority trunks (best-effort and higher traffic classes) with the objective of minimizing the cost-bandwidth product. In contrast, it allocates low priority scavenger traffic trunks with the goal of minimizing maximum link utilization. The rationale to minimize maximum link utilization objective for low priority traffic is to decrease congestion drops in the scavenger class from microbursts in higher traffic classes, and to allow bandwidth broker [14] to serve more requested bandwidth using under-utilized links. High priority users expect the best latency the network can offer.

The priority fairness solver chains four solvers, max-min fairness, minimize cost, minimize maximum utilization, and diverse path, in different combinations based on traffic classes. The inputs to all solvers are paths computed in the first phase, TM, and upstream solver constraints.

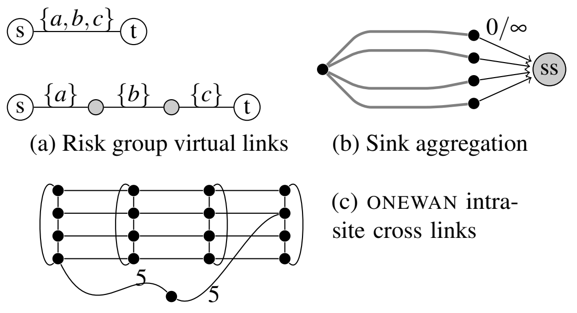
5.1 Path computation

Path finders in ONEWAN implement techniques for exploring paths in the network topology. Over the years, we have developed many path finders. Today, we accumulate paths from two paths finders in ONEWAN— penalizing and maximum flow path finders. The union of paths from the two finders has a mix of diverse shortest and maximum flow paths.

The penalizing path finder returns risk diverse paths with policies to never reuse a *risk group* for paths between the same source–sink pair or reuse only if necessary. Risk group is an identifier for shared optical infrastructure used by two or more links. A link can have zero or more risk groups. The finder uses Dijkstra's algorithm by setting link weights to penalties for risk groups used in previous shortest paths. Penalties are either infinity or a large value like the sum of all edge weights. The first path returned by the solver is the shortest path. Each subsequent path is the shortest path in the graph with modified weights. The finder operates on a graph where the risks groups have been expanded into virtual links (see Figure 13 (a)).

The maximum flow path finder uses maximum flow algorithms [11] and converts the augmenting paths into network paths. Link bandwidth is set to the reservable bandwidth for ONEWAN-TE. Link distance is not used in this finder.

Recall from § 3 that traffic engineered routes are not to individual routers, but to a group of routers in the destination site. Therefore, ONEWAN path finders compute paths from source nodes to destination sites using the technique of *sink aggregation*. In sink aggregation, we add a super sink node (ss) to the graph and connect sink nodes to it using directed edges of zero weight and infinite bandwidth (see Figure 13 (b)). Sink aggregation reduces the number of paths that ONEWAN-TE needs to allocate traffic on. Work is further reduced by computing paths only for source-sink pairs in the traffic matrix, instead of all pairs of nodes. This is called *TM-aware source sinks*. Figure 14 shows that these techniques reduce the path counts by a factor of 30. The resulting speed-ups extend into the optimization phase because fewer paths mean fewer columns in the linear programming constraint matrix. The average time to compute paths in ONEWAN is only 5 seconds.



(a) Risk group virtual links (b) Sink aggregation (c) ONEWAN intra-site cross links

Figure 13: Path computation: (a) penalizing path finder expands risk groups to virtual links, (b) sink aggregation finds paths to all nodes in a site in the same exploration using a super sink, (c) cross-links within a site complicate *k*-shortest path exploration.

Why does ONEWAN not use k-shortest paths? Path computation using maximum flow algorithms works better than *k*-shortest path algorithms on the ONEWAN topology. This is because ONEWAN sites have a Clos or cross-link structure that requires *k* to increase exponentially with the number of inter-site hops (see Figures 3 and 13 (c)). There is dissimilar link bandwidth and cost for nodes in a site that make optimizations difficult. Switching from *k*-shortest paths to maximum flow algorithms gave a significant boost in path choice and speed.

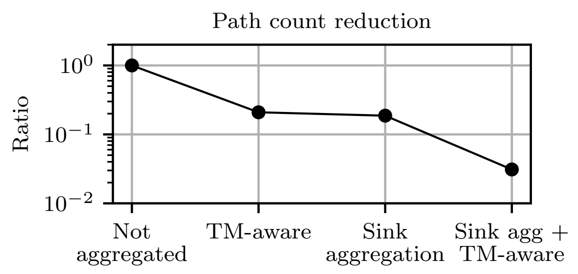


Figure 14: Path count reduction when paths are computed for traffic matrix aware source sinks and aggregated by sink nodes in a site.

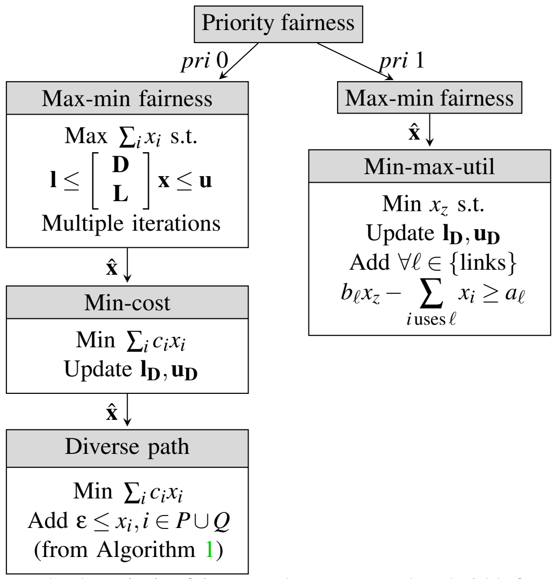
5.2 Priority fairness solver

The priority fairness solver assigns priorities to traffic trunks based on their traffic class. Best-effort and higher traffic classes map to priority 0 (highest) and scavenger traffic class trunks are priority 1. Cloud network operators set a *committed data rate* for each priority. Committed data rate is the minimum percentage of link bandwidths guaranteed to a priority. If excess network bandwidth is available for use, traffic trunks receive more than the guaranteed allocation in priority order. By pre-committing link bandwidths based on priority, we ensure that lower priority trunks do not starve in the network.

The priority fairness solver runs in two stages. In the first stage, starting with the highest priority, it sets link bandwidths in proportion to the committed rate of the priority. This excludes the bandwidth committed to lower priorities, guaranteeing that lower priority demands are not starved for bandwidth. The fairness solver then allocates bandwidth for demands in this priority. It repeats this process for each lower priority demand set and accumulates all the downstream solver results into a single priority-aware solver result.

During the second stage, the solver walks through each set of demands in a priority and identifies fully satisfied demands. These demands are marked as frozen. For unsatisfied demands, it adds the credit allocated in the first stage back to the available link bandwidth. Adding the credits back to the links enables these demands to be run again as if none were allocated. In the first stage the demands were limited to available bandwidth after excluding committed rates of lower priorities. In this stage, unused committed rates are available and can be used for unsatisfied or partially allocated demands.

Solver chaining. ONEWAN-TE has multiple objectives. It allocates priority 0 traffic to achieve max-min fairness and minimum cost using a diverse set of network paths. It allocates priority 1 traffic to achieve max-min fairness and minimize the maximum link utilization. The priority fairness solver achieves these TE objectives by chaining multiple solvers to compute traffic allocations. Traffic allocations from upstream solvers in the chain constrain the solution space of subsequent solvers, achieving one TE objective per link of the solver chain. Solver chaining breaks ONEWAN's TE problem into reconfigurable linear programming (LP) steps. The priority fairness solver (see Figure 15) uses four solvers. For brevity,


Priority fairness
pri 0
pri 1
Max-min fairness
Max $\sum_i x_i$ s.t.
$\mathbf{l} \leq \begin{bmatrix} \mathbf{D} \\ \mathbf{L} \end{bmatrix} \mathbf{x} \leq \mathbf{u}$
Multiple iterations
$\hat{\mathbf{x}}$
Min-cost
Min $\sum_i c_i x_i$
Update $\mathbf{l_D}, \mathbf{u_D}$
$\hat{\mathbf{x}}$
Diverse path
Min $\sum_i c_i x_i$
Add $\varepsilon \leq x_i, i \in P \cup Q$
(from Algorithm 1)
Max-min fairness
$\hat{\mathbf{x}}$
Min-max-util
Min x_z s.t.
Update $\mathbf{l_D}, \mathbf{u_D}$
Add $\forall \ell \in \{\text{links}\}$
$b_\ell x_z - \sum_{i \text{ uses } \ell} x_i \geq a_\ell$
Figure 15: The priority fairness solver reserves bandwidth for traffic classes based on their priorities. It then invokes the chain of max-min fairness, min-cost and diverse path solvers for high priority traffic. High priority traffic does not consume bandwidth reserved for low priority traffic. After allocating high priority traffic, it invokes the max-min and min-max util solvers for lower priority traffic. Unused bandwidth after both high and low priority chains is made available for unmet demands in the second round of priority fairness solver.

we omit a detailed discussion of the optimization problems in the solvers, previously described in [3, 14, 19, 27]. $\mathbf{x}$ has an element for each path and its solution $\hat{\mathbf{x}}$ is the allocation of bandwidths requested by trunks to paths. The solvers share a common core of demand and link constraints $(\mathbf{D}, \mathbf{L})$.

Chain of solvers. The *Max-min fairness solver* optimizes throughput using approximate max-min fairness. It uses multiple iterations of maximizing throughput with adjustments to demand constraint bounds. *Min-cost solver* uses the solution of max-min fairness to minimize the dot product of cost and allocations. For ONEWAN-TE, cost c_i is path metric, which is the sum of link metrics. It uses the solution of the max-min fairness to adjust the bounds in demand constraints. *Diverse path solver* solves the same objective but with diverse path constraints described in § 5.3. Priority 1 trunks are optimized with max-min fairness using the residual link capacity after deducting the allocations of priority 0. *Min-max-utilization solver* uses the upstream solution to adjust the bounds in demand constraints. To minimize maximum utilization with a linear objective, it adds a new variable x_z that represents maximum utilization, and utilization constraints where a_ℓ is previously allocated link bandwidth and b_ℓ is link capacity.

LP solvers. ONEWAN integrates two LP solvers, CLP [5] and GLOP [12]. They provide a 5× speedup over the original solver used in SWAN. We achieve additional speedup by reducing the constraints sent to the solvers: the use of destination sites reduces the number of paths, and therefore columns in the constraint matrix are reduced 7-fold (§ 5.1). The biggest gain in constraint reduction was achieved by pruning small traffic trunks that are under 500 Mbps. This decreased the constraint matrix rows and columns by factors of 5 and 7 respectively. The small trunks represent only 2.5% of network traffic (described in § 6.2) and get optimized within five minutes of growing large. Practicality of using LP solvers also depends on settings: dual simplex is superior to primal simplex for our LP models. Furthermore, pure cycling is possible as the max-min fairness models have high degeneracy — many columns are the same because the non-zero elements in the constraint matrix and the cost vector coefficients are 1. Cost perturbation is used to jiggle the values out of cycles.

5.3 Diverse path solver

A collection of paths is *risk diverse* if the intersection of risk groups of all paths in the collection is an empty set. ONEWAN applies diverse path protection to best-effort and higher traffic classes. In the ONEWAN-TE solver chain, the diverse path solver computes diverse paths using the primary paths from the min-cost solver. The goal of the diverse path solver is to decrease transient congestion in the interval between local and global repairs. ONEWAN agents perform local repair at the ingress router and ONEWAN controllers perform global repair based on the new topology. Spreading traffic on multiple paths without considering risk diversity can result in transient packet loss due to route blackholes or congestion since local repair is forced to use IS-IS routes.

The diverse path solver uses a greedy weighted set cover to find the minimum weight set of diverse paths that protects the shared risks in primary paths. Algorithm 1 outlines diverse path constraint generation. The WEIGHT function is configurable. Diverse paths can be configured to not exceed the primary path latency by a configurable jitter threshold, or diverse paths with more residual bandwidth can be preferred. We define jitter as a normalized ratio of path latencies, $(max - min)/\sqrt{min}$. Since high priority users expect the best latency in non-failure conditions, diverse paths with excess jitter are excluded. The diverse path solver re-solves the minimum cost objective with diversity constraints (see Figure 15). Diverse paths usually get the smallest possible weight since using them pulls the solution away from the minimum cost solution.

Figure 16 shows the percentage of total traffic on risk diverse paths. Without diverse path solver, any diversity is purely accidental, and was measured at 10%. When the agent only supported primary tunnels, jitter threshold of 5 was used to not adversely impact the latency of flows using the diverse paths. This constrained the choice of diverse paths and risk diverse traffic percentage was 75%. Once the agent implemented primary-backup tunnels, the jitter threshold was not required, and the protected traffic increased to 99%. The remaining 1% is due physical constraints on diversity of optical circuits.

Algorithm 1 Adds diverse path constraints

```
1:  procedure DIVERSITYCONSTRAINTS($P$, $U$)
        $P$ is the set of primary paths.
        $U$ is the set of computed paths.
2:      $risks \leftarrow$ SHAREDRISKS($P$)
3:      $ss \leftarrow$ SOURCESINKS($P$)
4:      $candidates \leftarrow$ PATHS($U, ss$) $\setminus P$
5:      for all $q$ in $candidates$ do
6:          $protect[q] \leftarrow risks \setminus$ SHAREDRISKS($q$)
7:          $residual \leftarrow$ residual bandwidth of $q$
8:          $jitter \leftarrow$ latencies of $q$ and $P$
9:          $w[q] \leftarrow$ WEIGHT($residual, jitter$)
10:     end for
11:     $Q \leftarrow$ WEIGHTEDSETCOVER($protect, w$)
12:     for all $i$ in $P \cup Q$ do
13:         ADDCONSTRAINT($\varepsilon \leq x_i$)
14:     end for
15: end procedure
```

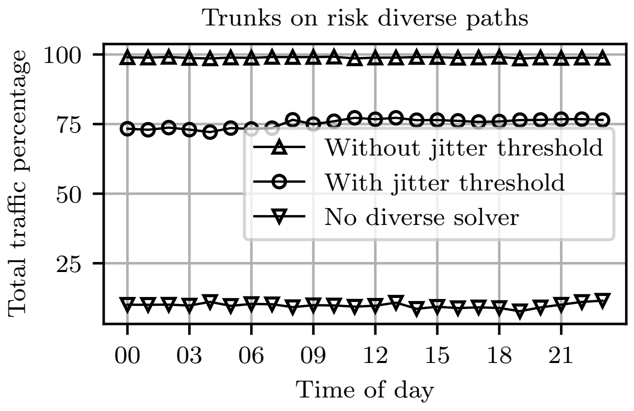


Figure 16: Percentage of total traffic on risk diverse paths in a typical day. 10% of the traffic is protected without diverse path solver, 75% with jitter threshold, and 99% without jitter threshold.

6 Measuring WAN Traffic Matrices

The traffic matrix (TM) is a key input to the traffic engineering optimizer. A traffic trunk is an aggregate traffic flow from a source backbone router to a destination site for a specific traffic class. There are four primary traffic classes in ONEWAN: voice, interactive, best-effort, and scavenger. The WAN TM is a collection of traffic trunks and bandwidths for each trunk. A trunk's bandwidth can be a requested value for discretionary traffic, a measured value for Internet or non-discretionary traffic, or a prediction based on measured values. Bandwidth broker is a service that measures discretionary traffic at sending hosts and aggregates it into trunk-level requested bandwidth. The input to the traffic engineering optimizer is the complete traffic matrix that is a combination of the requested TM and predictions based on the measured TM.

Measured TMs are computed by sampling packets as they enter the WAN. SWAN sampled traffic using sFlow [30] at backbone routers. On the other hand, ONEWAN sampled traffic using IPFIX [33] at aggregation routers. The sampling point and technique were dictated by which router exported the fields required for identifying traffic trunks. Flow record attributes such as sampling router address, input and output interfaces, BGP next hop, traffic class, and packet 5-tuple are used to identify the traffic trunk. ONEWAN required a high throughput data pipeline to process flow records from a larger number of devices at a faster sampling rate compared to SWAN. Since ONEWAN-TE re-optimizes traffic allocations every 5 minutes, TMs are measured at short, minute-level timescales.

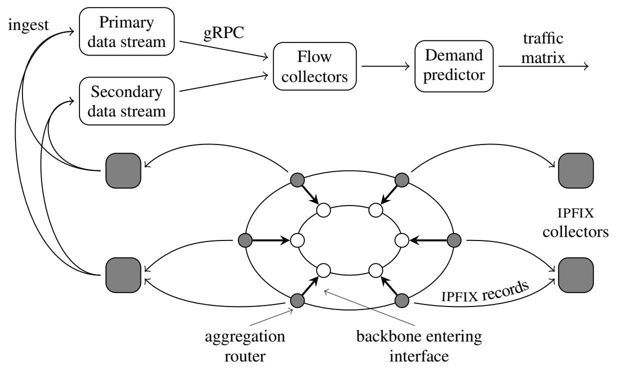


Figure 17: ONEWAN's high throughput data pipeline for TM measurement. Aggregation routers export flow information to IPFIX collectors which process and store them in redundant data streams. Flow collectors query the data stream to compute bandwidths of traffic trunks. The demand predictor predicts the complete TM.

Challenges in measuring anycast traffic volumes. A large fraction of the traffic in CORE is anycast — traffic towards destination prefixes advertised by multiple sites in the WAN. Anycast traffic is routed to the router nearest to the *source* router using standard BGP path selection rules. Correctly identifying the destination of an anycast traffic trunk requires knowledge of the BGP route lookup result, which is different for each source. One way of solving this in software is to acquire copies of the routing tables of all aggregation routers and replay the route lookup. To avoid the overhead of copying tables and replaying routes, ONEWAN leverages IPFIX sampling where routers store the route lookup result in the BGP next hop attribute of IPFIX flow records.

IPFIX data pipeline to measure traffic matrices. In Figure 17, aggregation routers sample one in 4,096 IP packets, and export flow records using anycast to the nearest IPFIX collector cluster. The IPFIX collectors write the statistics to redundant data streams. Flow collectors query the data stream for flow records from aggregation routers to the connected backbone routers and build a traffic matrix. They also identify and tag discretionary traffic in the measured TM to help in combining with the requested TM. The demand predictor aggregates traffic matrices from all collectors and uses linear regression and autoregressive moving average models on measured TMs to make predictions.

6.1 Error correction in measured TMs

We calculate the accuracy of traffic matrices measured using aggregations of IPFIX data against packet counters like interface counters, output queue counters, and RSVP-TE used bandwidth counters. Early versions of the IPFIX data pipeline had issues like invalid or missing attributes in flow records *e.g.,* missing BGP next hop attribute for IPv6 flows, or incorrect egress interface in certain flows. We detected and fixed such issues over time.

Flow record exporters use UDP, even though it is unreliable, because it needs less router resources. Hence, it is important that the data pipeline does not lose flow records due to traffic surges, unequal load distribution, or systemic or fault induced capacity crunch. The first generation of the data pipeline was lossy. The second generation of the pipeline used gRPC streaming from the data stream to the flow collectors (see Figure 17) and was not lossy. While operating with the first generation, we developed an error correction that gave similar results as the second generation. Since there is potential of failures in any pipeline, we outline the error correction technique used to improve reliability of our TM estimation.

We define the *interface error rate* as the ratio of input interface bit rate on the backbone entering interface measured by SNMP and calculated by IPFIX. Values greater than one indicate underestimation, and less than one indicate overestimation by IPFIX. The flow collector continuously calculates interface error rate, and scales the IPFIX measured bandwidths of individual traffic trunks in proportion to the interface error rate. The flow collector scales the bandwidth of each trunk separately based on its input interface error.

6.2 Traffic matrix characteristics

Figure 18 (a) shows typical diurnal and weekly traffic patterns seen in service provider networks. Interestingly, the number of traffic trunks in ONEWAN is 8× the trunks in SWAN due to the doubling of source backbone routers, and increase in destination sites in ONEWAN.

In Figure 18 (b), 82% of trunks are under 100 Mbps and represent 1.3% of total traffic in WAN. The small trunks are uniformly distributed across the network, consist of many flow types, and do not pathologically contend for specific links. We take advantage of this distribution to speed up traffic engineering optimization by letting the smallest trunks go unengineered.

In Figure 18 (c), trunk distance is calculated based on fiber distance of the shortest path from source to destination. The largest traffic trunks have source and destination close to each other. 25th-percentile by distance is 95% of total traffic. We use this to define ONEWAN geographies such that 75% of the WAN traffic is intra-geography.

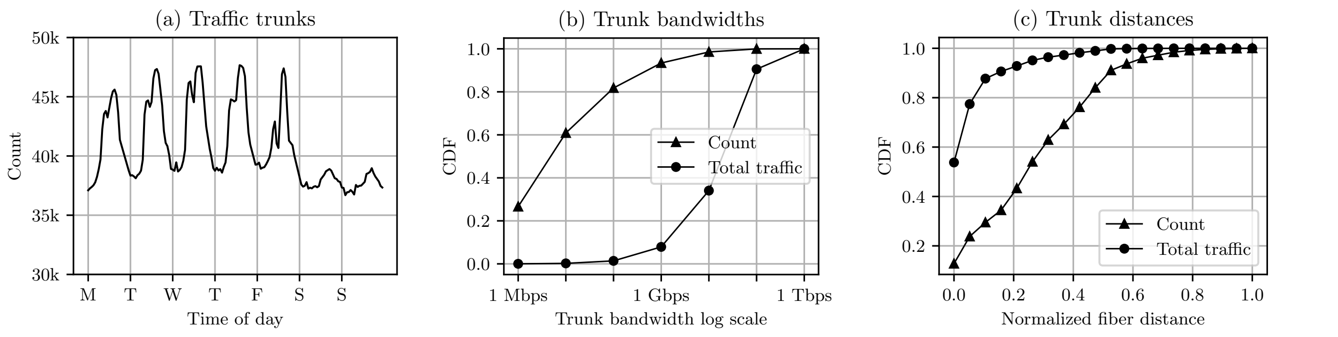


Figure 18: Traffic matrix characteristics for a typical week: (a) count of trunks by time of day, (b) distribution of trunks by bandwidth, (c) distribution of trunks by normalized fiber distance from source to destination.

7 Operational Experience

In this section we share lessons learned from the unification of SWAN and CORE networks into ONEWAN.

Traffic migration. We migrated traffic from RSVP-TE to ONEWAN-TE in the production network. So, it was important to perform this migration in a hitless manner. We used steering routes to migrate traffic from a specific set of source routers to another specific set of endpoints. We began by migrating traffic that was sourced and destined within the same geography because the configuration changes required for this are local to that geography. We used RSVP-TE maximum reservable bandwidth configuration on routers to control the percentage of link bandwidth made available to ONEWAN-TE. Initially, ONEWAN-TE had a small fraction of link bandwidth, sufficient to initiate traffic migration. We gradually increased ONEWAN-TE traffic and reduced the RSVP-TE maximum reservable bandwidth until RSVP-TE configuration could be entirely removed from the router.

Merging IS-IS routing domains. Merging the SWAN and CORE IS-IS domains posed a high risk for SWAN routers, whose IS-IS would observe a 7-fold increase in link state advertisements after the merge. IS-IS uses backoff timers to pace the shortest path first (SPF) execution. The purpose of backoff timers is to react quickly to the first few events but under constant churn, slow down to prevent the router from collapsing. The interactions of TE extensions still using IS-IS with interoperability issues caused IS-IS SPF to be persistently in backoff state slowing convergence. This prompted the design change to make ONEWAN-TE tunnel probing completely independent of IS-IS by using controller routes to return from the tunnel destination to the tunnel source (§ 4).

Cost savings with ONEWAN. ONEWAN brings the benefits of SDN TE to Internet traffic which was previously managed by RSVP-TE in the CORE network. SDN TE is known to make efficient use of network capacity, thereby reducing the need for capacity augmentation in the network. In Figure 19, we compare projected capacity augments needed with ONEWAN vs. SWAN + CORE to meet the organic traffic growth. We expect to reduce capacity augments by 10% of current installed capacity in the next few years, which is of significant value for the size of our network. This does not include savings on inorganic growth like building new regions.

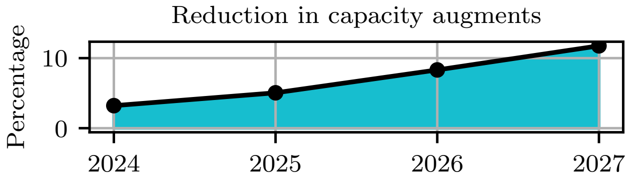


Figure 19: Reduction in capacity augments with ONEWAN compared to SWAN + CORE as a percentage of current installed capacity.

8 Related Work

We are the first to apply SDN techniques to replace RSVP-TE in an Internet backbone. SDN based traffic engineering was first used in inter-datacenter networks [9,14,15,17,20,24,25]. An inter-datacenter network has fewer points of presence, simpler scaling requirements, and easily built from scratch. We show that the challenges of scalability, reliability, feature parity, and migration can be overcome and replace RSVP-TE in an Internet backbone. This leads to unification of the cloud network with a single SDN controlled backbone.

SDN controllers for Internet peering have been discussed in [4, 7, 13, 34, 35, 37]. They tackle an important adjacent problem of Internet traffic engineering at the peering edge. These controllers enable performance-aware egress peer selection and inbound traffic engineering between autonomous systems. Our work focuses on the dynamic path selection and load balancing of this traffic between the peering edge and the end host in a datacenter, as it transits the backbone within the autonomous system. Microsoft peering edge uses a similar SDN controller that is outside the scope of this paper. ONEWAN controllers measure peering traffic and adapt the backbone to the needs of the peering edge traffic.

Traffic matrix estimation through models and link data have been studied in [8,26,32,38]. In our experience, direct measurement of traffic matrices is, unfortunately, necessary for online TE in operational networks. We extend prior work with an evaluation of our IPFIX sampling based measurement system, and present data on real world traffic matrices.

Prior work [14, 18, 23] states that programming end-to-end paths in WANs takes minutes, and the size of tables to store traffic engineering rules is a constraint. Our work shows that network updates take seconds, even in very large networks. A key reason for this difference is that prior work used TCAM-based policy engines that are flexible but limited resources, and our work uses IP and MPLS lookup tables, which are optimized and large even in commodity switches, and programming in parallel using make-before-break semantics. Path selection is usually done offline [6] and load balancing over selected paths is online. In ONEWAN-TE, both are done online using the dynamic topology and traffic matrix. Traffic engineering is studied as the optimization of one objective like minimizing link utilization [36]. ONEWAN-TE optimizes each traffic class with different objectives, and uses class based forwarding to achieve the intent in the data plane. [1, 25] study the TE problem with faults. ONEWAN-TE formulation is tuned for a larger network. It protects faults at the granularity of optical risk groups and balances the opposing requirements of proactive fault protection without increasing latency in non-failure conditions.

9 Conclusion

Our journey with using SDN for traffic engineering has completed a full circle. SDN-TE technology matured in our network while managing inter-datacenter traffic, and has now replaced legacy TE in the Internet backbone. ONEWAN represents a 1000× increase in traffic volume and a 100× increase in network size compared to SWAN a decade ago. Some key elements of SWAN have withstood the test of time. For example, traffic engineering optimization retains the same structure and formulation. We still use router agents on WAN routers which run on multiple firmware operating systems, but have modified them to deal with greater scale and functionality. ONEWAN brings the benefits of smaller fault-domains from BLASTSHIELD to the Internet backbone, making it more reliable. We hope ONEWAN expands the scope of future work in wide-area TE, standardization of the controller-agent programming abstraction, and analysis of traffic matrix characteristics of cloud WANs. ONEWAN marks the progression of SDN into an Internet backbone. It is driven by scaling our backbone to serve today's users. We continue to expand the roles and functionality of ONEWAN as new opportunities emerge with the growth of network demands.

Acknowledgements. We thank our colleagues in Azure wide area networking team who made significant contributions to make ONEWAN successful, our shepherd Ram Durairajan, and NSDI reviewers for their comments that improved this paper.

References

[1] David Applegate, Lee Breslau, and Edith Cohen. Coping with network failures: Routing strategies for optimal demand oblivious restoration. In *Proceedings of the Joint International Conference on Measurement and Modeling of Computer Systems*, pages 270–281. 2004.

[2] Daniel O. Awduche, Lou Berger, Der-Hwa Gan, Tony Li, Vijay Srinivasan, and George Swallow. RSVP-TE: Extensions to RSVP for LSP tunnels, December 2001. RFC 3209.

[3] Jeremy Bogle, Nikhil Bhatia, Manya Ghobadi, Ishai Menache, Nikolaj Bjørner, Asaf Valadarsky, and Michael Schapira. TEAVAR: striking the right utilization-availability balance in WAN traffic engineering. In *Proceedings of ACM SIGCOMM*, pages 29–43. August 2019.

[4] Matthew Caesar, Donald Caldwell, Nick Feamster, Jennifer Rexford, Aman Shaikh, and Jacobus van der Merwe. Design and implementation of routing control platform. In *Proceedings of USENIX NSDI*, pages 15–28, May 2005.

[5] COIN-OR linear programming solver. https://github.com/coin-or/Clp.

[6] Anwar Elwalid, Cheng Jin, Steven H. Low, and Indra Widjaja. MATE: MPLS adaptive traffic engineering. In *Proceedings of IEEE INFOCOM*, volume 3, pages 1300–1309, 2001.

[7] Nick Feamster, Jay Borkenhagen, and Jennifer Rexford. Guidelines for interdomain traffic engineering. *ACM SIGCOMM Computer Communication Review*, 33(5):19–30, October 2003.

[8] Anja Feldmann, Albert Greenberg, Carsten Lund, Nick Reingold, Jennifer Rexford, and Fred D. True. Deriving traffic demands for operational IP networks: methodology and experience. *IEEE/ACM Transactions on Networking*, 9(3):265–279, 2001.

[9] Mikel Jimenez Fernandez and Henry Kwok. Building Express backbone: Facebook's new long-haul network, May 2017. https://engineering.fb.com/2017/05/01/data-center-engineering/building-express-backbone-facebook-s-new-long-haul-network/.

[10] Clarence Filsfils, Stefano Previdi, Les Ginsberg, Brune Decraene, Stephane Litkowski, and Rob Shakir. Segment routing architecture, July 2018. RFC 8402.

[11] Andrew V. Goldberg, Éva Tardos, and Robert E. Tarjan. Network flow algorithms. In Bernhard Korte, Lásló Lovász, Hans Jürgen Prömel, and Alexander Schrijver, editors, *Paths, Flows, and VLSI Layout (Algorithms and Combinatorics)*, volume 9, pages 101–164. Springer-Verlag, 1990.

[12] OR-Tools – Google optimization tools. https://github.com/google/or-tools.

[13] Arpit Gupta, Laurent Vanbever, Muhammad Shahbaz, Sean P. Donovan, Brandon Schlinker, Nick Feamster, Jennifer Rexford, Scott Shenker, Russ Clark, and Ethan Katz-Bassett. SDX: A software defined internet exchange. In *Proceedings of ACM SIGCOMM*, pages 551–562, 2014.

[14] Chi-Yao Hong, Srikanth Kandula, Ratul Mahajan, Ming Zhang, Vijay Gill, Mohan Nanduri, and Roger Wattenhofer. Achieving high utilization with software-driven WAN. In *Proceedings of ACM SIGCOMM*, pages 15–26, August 2013.

[15] Chi-Yao Hong, Subhasree Mandal, Mohammad Al-Fares, Min Zhu, Richard Alimi, Kondapa Naidu B., Chandan Bhagat, Sourabh Jain, Jay Kaimal, Shiyu Liang, Kirill Mendelev, Steve Padgett, Faro Rabe, Saikat Ray, Malveeka Tewari, Matt Tierney, Monika Zahn, Jonathan Zolla, Joon Ong, and Amin Vahdat. B4 and after: Managing hierarchy, partitioning, and asymmetry for availability and scale in Google's software-defined WAN. In *Proceedings of ACM SIGCOMM*, pages 74–87, August 2018.

[16] Intermediate System to Intermediate System intra-domain routeing information exchange protocol for use in conjunction with the protocol for providing the connectionless-mode network service (ISO 8473). ISO/IEC 10589:2002, November 2002. https://www.iso.org/standard/30932.html.

[17] Sushant Jain, Alok Kumar, Subhasree Mandal, Joon Ong, Leon Poutievski, Arjun Singh, Subbaiah Venkata, Jim Wanderer, Junlan Zhou, Min Zhu, Jonathan Zolla, Urs Hölzle, Stephen Stuart, and Amin Vahdat. B4: Experience with a globally-deployed software defined WAN. In *Proceedings of ACM SIGCOMM*, pages 3–14, August 2013.

[18] Xin Jin, Hongqiang Harry Liu, Rohan Gandhi, Srikanth Kandula, Ratul Mahajan, Ming Zhang, Jennifer Rexford, and Roger Wattenhofer. Dynamic scheduling of network updates. In *Proceedings of ACM SIGCOMM*, pages 539–550, August 2014.

[19] Srikanth Kandula, Dina Katabi, Bruce Davie, and Anna Charny. Walking the tightrope: Responsive yet stable traffic engineering. In *Proceedings of ACM SIGCOMM*, pages 253–264. 2005.

[20] Srikanth Kandula, Ishai Menache, Roy Schwartz, and Spandana Raj Babbula. Calendaring for wide area networks. In *Proceedings of ACM SIGCOMM*, pages 515–526, August 2014.

[21] Dave Katz and Dave Ward. Bidirectional Forwarding Detection, June 2010. RFC 5880.

[22] Umesh Krishnaswamy, Rachee Singh, Nikolaj Bjørner, and Himanshu Raj. Decentralized cloud wide-area network traffic engineering with BLASTSHIELD. In *Proceedings of USENIX NSDI*, pages 325–338, April 2022.

[23] Praveen Kumar, Yang Yuan, Chris Yu, Nate Foster, Robert Kleinberg, Petr Lapukhov, Chiun Lin Lim, and Robert Soulé. Semi-oblivious traffic engineering: The road not taken. In *Proceedings of USENIX NSDI*, pages 157–170. April 2018.

[24] Nikolaos Laoutaris, Michael Sirivianos, Xiaoyuan Yang, and Pablo Rodriguez. Inter-datacenter bulk transfers with NetStitcher. In *Proceedings of ACM SIGCOMM*, pages 74–85. 2011.

[25] Hongqiang Harry Liu, Srikanth Kandula, Ratul Mahajan, Ming Zhang, and David Gelernter. Traffic engineering with forward fault correction. In *Proceedings of ACM SIGCOMM*, pages 527–538, August 2014.

[26] Alberto Medina, Nina Taft, Kavé Salamatian, Supratik Bhattacharyya, and Christophe Diot. Traffic matrix estimation: Existing techniques and new directions. In *Proceedings of ACM SIGCOMM*, pages 161–174. 2002.

[27] Debasis Mitra and K.G. Ramakrishnan. A case study of multiservice, multipriority traffic engineering design for data networks. In *IEEE GLOBECOM*, volume 1B, pages 1077–1083, 1999.

[28] OpenFlow switch specification, March 2015.

[29] Ping Pan, George Swallow, and Alia Atlas. Fast reroute extensions to RSVP-TE for LSP tunnels, May 2005. RFC 4090.

[30] Peter Phaal and Marc Levine. sFlow version 5, July 2004.

[31] Eric C. Rosen, Arun Viswanathan, and Ross Callon. Multiprotocol label switching architecture, January 2001. RFC 3031.

[32] Matthew Roughan, Mikkel Thorup, and Yin Zhang. Traffic engineering with estimated traffic matrices. In *Proceedings of the 3rd ACM SIGCOMM Conference on Internet Measurement*, pages 248–258. 2003.

[33] Ganesh Sadasivan, Nevil Brownlee, and Benoit Claise. Architecture for IP flow information export, March 2009. RFC 5470.

[34] Brandon Schlinker, Hyojeong Kim, Timothy Cui, Ethan Katz-Bassett, Harsha V. Madhyastha, Italo Cunha, James Quinn, Saif Hasan, Petr Lapukhov, and Hongyi Zeng. Engineering egress with Edge Fabric: Steering oceans of content to the world. In *Proceedings of ACM SIGCOMM*, pages 418–431, 2017.

[35] Rachee Singh, Sharad Agarwal, Matt Calder, and Paramvir Bahl. Cost-effective cloud edge traffic engineering with Cascara. In *Proceedings of USENIX NSDI*, pages 201–216, April 2021.

[36] Hao Wang, Haiyong Xie, Lili Qiu, Yang Richard Yang, Yin Zhang, and Albert Greenberg. Cope: Traffic engineering in dynamic networks. In *Proceedings of ACM SIGCOMM*, pages 99–110. August 2006.

[37] Kok-Kiong Yap, Murtaza Motiwala, Jeremy Rahe, Steve Padgett, Matthew Holliman, Gary Baldus, Marcus Hines, Taeeun Kim, Ashok Narayanan, Ankur Jain, et al. Taking the edge off with Espresso: Scale, reliability and programmability for global internet peering. In *Proceedings of ACM SIGCOMM*, pages 432–445, 2017.

[38] Yin Zhang, Matthew Roughan, Nick Duffield, and Albert Greenberg. Fast accurate computation of large-scale IP traffic matrices from link loads. In *Proceedings of ACM SIGMETRICS*, pages 206–217, 2003.