MegaTE: Extending WAN Traffic Engineering to Millions of Endpoints in Virtualized Cloud

Congcong Miao[1*] Zhizhen Zhong[2*] Yunming Xiao[3*] Feng Yang[1*] Senkuo Zhang[1*]
Yinan Jiang[4] Zizhuo Bai[4] Chaodong Lu[1] Jingyi Geng[1] Zekun He[1]
Yachen Wang[1] Xianneng Zou[1] Chuanchuan Yang[4]

[1]Tencent [2]Massachusetts Institute of Technology [3]Northwestern University [4]Peking University

Abstract

In today's virtualized cloud, containers and virtual machines (VMs) are prevailing methods to deploy applications with different tenant requirements. However, these requirements are at odds with the resource allocation capabilities of conventional networking stacks in wide-area networks (WANs). In particular, existing WAN traffic engineering (TE) systems *at the granularity of aggregated traffic flows* are not designed to cater to each individual flow. In this paper, we advocate for a radical new approach to extend TE systems to involve *millions of virtual instance endpoints*. We propose and implement a first-of-its-kind system, called MegaTE, to satisfy the needs of each fine-grained traffic flow at the virtual instance level. At the core of the MegaTE system is the paradigm shift from the *top-down* centralized control to the *bottom-up* asynchronous query in the TE control loop, combined with eBPF-based segment routing on the data plane and TE optimization contraction on the control plane. We evaluate MegaTE using flow-level simulations with production traffic traces. Our results show that MegaTE supports 20× more endpoints with the similar algorithm run time compared to prior work. MegaTE has been adopted by large-scale public cloud providers. Notably, Tencent rolled out MegaTE in its cloud WAN since December 2022. Our production analysis shows that MegaTE reduces the packet latency of real-time applications by up to 51%.

CCS Concepts

• **Networks** → **Traffic engineering algorithms**; *Network performance evaluation*; **Wide area networks**; **Network architectures**.

Keywords

Wide-Area Networks, Traffic Engineering, Segment Routing, eBPF, Network Optimization, Virtualized Cloud

ACM Reference Format:
Congcong Miao, Zhizhen Zhong, Yunming Xiao, Feng Yang, Senkuo Zhang, Yinan Jiang, Zizhuo Bai, Chaodong Lu, Jingyi Geng, Zekun He, Yachen Wang, Xianneng Zou, Chuanchuan Yang. 2024. **MegaTE**: Extending WAN Traffic Engineering to Millions of Endpoints in Virtualized Cloud. In *ACM SIGCOMM 2024 Conference (ACM SIGCOMM '24), August 4–8, 2024, Sydney, NSW, Australia.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3651890.3672242

*Equal contribution.

ACM SIGCOMM '24, August 4–8, 2024, Sydney, NSW, Australia

ACM ISBN 979-8-4007-0614-1/24/08
https://doi.org/10.1145/3651890.3672242

1 Introduction

The advent of cloud computing has catalyzed a fundamental evolution in how applications are deployed and managed in modern computer systems. By allowing multiple tenants to provision and run *virtual instances* like containers, virtual machines (VMs), and virtual private networks (VPNs) on the same physical infrastructure with a "pay-as-you-go" model, significant cost-sharing benefits have been achieved [4, 38].

Currently, as cloud applications become more latency sensitive and geologically distributed, the requirements of cloud tenants have evolved from basic computing resources like CPU cores, memory, disk, etc. to networking demands like packet latency (e.g., round-trip time) and connection bandwidth. In today's networking stack, these requirements are primarily handled by the traffic engineering (TE) system on the IP layer. In particular, when traffic flows arrive at routers, the TE *control plane* decides the routing paths (i.e., TE tunnels) to forward the packet, and the TE *data plane* devices (e.g., routers and switches) split the *aggregated data flows* by forwarding each individual packet according to the five tuples of the IP packet header.[1] Through splitting traffic across multiple tunnels routed over the network, TE systems play a crucial role in optimizing network utilization and minimizing link congestion in WANs [19, 20, 31, 37, 46]. However, our measurement study on a large-scale public cloud shows that today's WAN running conventional TE systems cannot satisfy the networking requirements of tenants' virtual instances on the application layer (§2.1).

The root cause of why existing TE systems do not meet virtual instance service requirements is that conventional TE operates on the IP layer at network switches and routers that handle traffic *at the granularity of aggregated flows*. The dynamic provisioning and decommissioning of millions of virtual instances on the application layer result in the fact that multiple flows of the same tenant have different identifiers in five tuples of IP packets. Hence, after hashing on routers, these flows may be routed on different TE tunnels and experience different packet latencies. As a result, there is no guarantee that multiple flows of the same virtual instance have the same packet latency, harming application-level performance. This fundamental mismatch between the capabilities of traditional TE systems and the nuanced demands of modern virtual instances highlights

[1]The five-tuple in IP networking is a set of five different values that uniquely identify a network connection or session: source IP address, destination IP address, source port, destination port, and transport protocol.

the need for a new TE system that manages each individual flow at the virtual instance level (§2.2).

In this paper, we advocate for extending TE systems to involve millions of virtual instance endpoints in the context of a large-scale public cloud. We first identify the key challenges in designing such a TE system (§3.1). Then, we propose MegaTE, a novel TE system that accommodates the *fine-grained* and *instance-specific* requirements of virtual instances. To the best of our knowledge, MegaTE is the first TE system that guarantees virtual instance flow performance. To achieve this, MegaTE proposes to rearchitect the TE control loop from *top-down* centralized control to *bottom-up* asynchronous database query (§3.2). Specifically, in traditional TE control loops, decisions on traffic routing and resource allocation are dispatched from the control plane (e.g., a centralized controller [19] or multiple decentralized controllers [28]) through a large number of *persistent connections*, leading to scalability issues and a lack of responsiveness to real-time changes in network conditions when network size grows larger. This approach struggles to accommodate the dynamic requirements of a modern virtualized cloud, where the networking needs of individual instances vary significantly and change rapidly across *millions* of endpoints. Instead, our *bottom-up* approach allows each endpoint on the data plane to query the TE results asynchronously with the eventual consistency mechanisms [5] such that the controller does not need to maintain persistent connections with millions of endpoints.

To implement and deploy MegaTE's novel bottom-up control loop at the scale of millions of virtual instance endpoints, we present key implementation components on both control and data planes.

On the control plane, MegaTE formulates the TE optimization problem with network contraction [2] to reduce the size of the problem. In particular, we leverage the fact that millions of endpoints are connected in a hierarchy through orders of magnitude fewer routers to contract the network topology. We further formulate the problem into a subset sum problem (SSP) and propose a fast approximate algorithm to solve the TE optimization within the TE update interval (e.g., 5 min [19]) for millions of endpoints (§4).

On the data plane, we build MegaTE on the synergy between eBPF (i.e., extended Berkeley Packet Filter)-based end host networking stacks to achieve efficiency and adaptability. By integrating eBPF for efficient packet processing and programmable network devices for dynamic path selection, MegaTE dynamically adjusts to changing network conditions and application demands, ensuring optimal performance and resource utilization across the cloud infrastructure (§5).

We first evaluate MegaTE using large-scale flow-level simulations. Our results show that MegaTE supports 20× more endpoints with a similar algorithm run time compared to state-of-the-art TE systems. Meanwhile, MegaTE satisfies at most 8.2% more demand than state-of-the-art TE algorithms in failure scenarios (§6).

MegaTE has been rolled out at Tencent WAN (TWAN), a large-scale public cloud provider with millions of virtual instance endpoints, since December 2022. Production measurement results show that MegaTE reduces the packet latency for the time-sensitive applications by 51%, ensuring availability for high-priority applications with an average availability of 99.995%, and reduces the cost by 50% for the low-priority applications (§7).

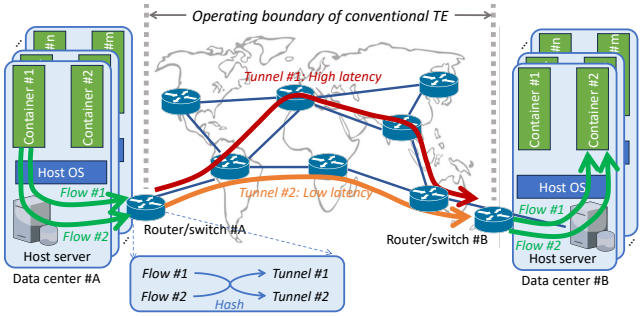


Figure 1: Conventional TE systems splits aggregated traffic flows on network routers/switches, orthogonal to host servers or virtual instance endpoints.

2 Background and Motivation

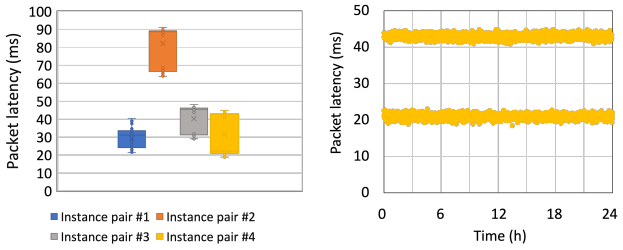
In the modern virtualized cloud, containers and VMs are primary methods for deploying, managing, and scaling applications across distributed computing resources. Containers, being lightweight and modular, package applications with their required dependencies on top of the operating system (OS), ensuring consistency across different platforms. Conversely, VMs offer strong isolation and security by simulating entire hardware systems, each with a complete OS stack. These virtualization technologies allow cloud operators to customize *computing* resources for each instance with unparalleled flexibility. Nevertheless, the *networking* stack, particularly the TE system, has yet to achieve the same level of efficiency.

In this section, we first perform a measurement study on a public cloud provider to quantitatively understand the limitations of existing TE systems in handling traffic flows of virtual instances (§2.1). We then identify the need to design new TE systems that meet the requirements of cloud tenants (§2.2).

2.1 Conventional TE Fails Cloud Tenants QoS

We begin by measuring the packet latency over time between four virtual instance pairs in geologically distributed data centers. Figure 1 is a graphical illustration of a cloud WAN that interconnects geologically distributed data center sites, where millions of virtual instances are dynamically provisioned and decommissioned. For conventional TE systems, they operate only on network routers/switches that split aggregated traffic flows onto different TE tunnels. In this case, multiple traffic flows initiated from the same virtual instance endpoint[2] enter the WAN through the edge router that aggregates and splits multiple flows. Due to dynamic provisioning and decommissioning of virtual instances, it is not practical to install flow entries for each individual flow on the routers. Therefore, the hash function of packet splitting cannot guarantee that all flows from the same virtual instances are routed on the same TE tunnel, resulting in possible violations of the packet latency requirements of cloud tenants. Figure 2(a) depicts the packet latency distributions using the box plot among four virtual instances measured for one day. It shows that the packet latency of these virtual instances has a large variance and cannot produce a stable packet

[2]The term *virtual instance endpoint* describes the source or destination of cloud tenants' traffic flows, such as VMs or containers.



(a) Packet latency distribution (b) Instance pair #4

Figure 2: Measured packet latency in a public cloud provider running conventional TE.

latency. If we zoom in on one of the virtual instance pairs (virtual instance pair #4) as shown in Figure 2(b), we find that packet latency is mostly clustered into two groups around 42 ms and 20 ms. The reason is that today's TE system cannot identify the flows at the instance level and hence naively hashed the flow onto two different tunnels dynamically, resulting in two different packet latencies over time. Ideally, tenants require the flow between *instance pair #1* and *instance pair #4* to be running on the low-latency path, while the other flow between *instance pair #2* and *instance pair #3* does not have latency requirements and therefore should be routed on the high-latency path.

2.2 The Need for TE to Manage Individual Flow

The traditional TE systems use hash functions to assign network flows to specific tunnels or paths based on parameters such as source and destination IP addresses, ports, and protocol types [11]. Although this method offers a simple and computationally efficient way to distribute traffic across available paths, it does not consider the unique characteristics and needs of individual flows. In particular, for a virtualized cloud environment, characterized by high traffic density and short-lived virtual instances, the static nature of five-tuple-based IP routing rules often falls short. This issue is particularly evident in microservice architectures for containers, where the complex and frequent communications among these microservices require more flexible and responsive routing solutions. Furthermore, the use of VPNs for robust security and isolation adds additional complexity to packet processing, which complicates the forwarding process. Consequently, it can result in suboptimal path selections, where latency sensitive or high priority traffic may be routed through longer or more congested paths, inadvertently increasing latency. Furthermore, hash-based mapping is not effective in real-time adaptation to dynamic network conditions, such as varying traffic loads or changing network topologies, which can further exacerbate latency issues. Therefore, there is a growing need for more intelligent and adaptive TE systems to handle each individual traffic flow and ensure optimal tunnel selection.

3 Supporting Millions of Endpoints in TE

In this section, we first identify several challenges in designing MegaTE to manage each individual flow of millions of virtual instance endpoints in the cloud (§3.1). Then, we explain how the key idea of this work, rearchitecting the TE control loop, alleviates the scaling bottleneck of supporting millions of endpoints (§3.2).

3.1 The TE Granularity Challenge

MegaTE, as the first TE system that extends its control loop to millions of endpoints to provide virtual-instance-level performance guarantees, has faced a set of challenges. The millions of endpoints, i.e., the virtual instances of tenants in the cloud that generates traffic flows, serve as a key factor that differentiates MegaTE from traditional WAN TE methodologies, which only operate on network routers/switches with aggregated traffic flows. Next, we elaborate on these key challenges.

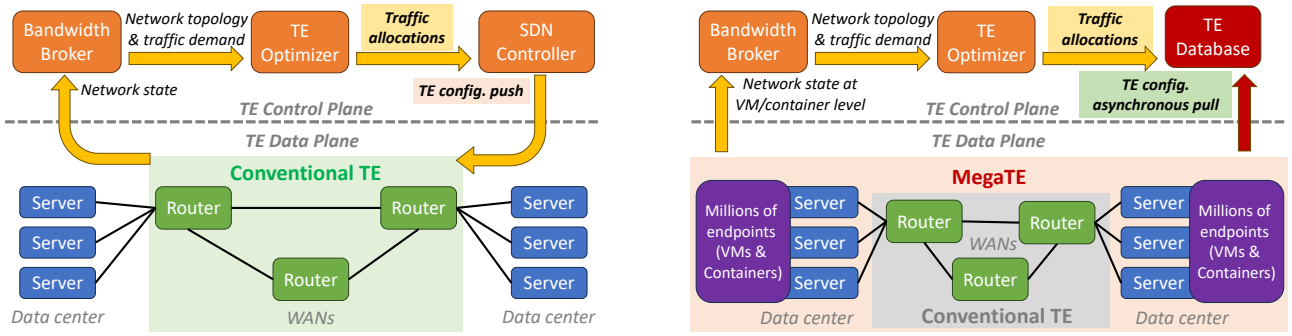
Handling individual flow on the data plane. Conventional TE systems do not support TE in the granularity of individual traffic flows between virtual instance endpoints due to the limited number of flow tables in the router. Therefore, in MegaTE, segment routing (SR) is used to allow WAN routers to identify and adhere to the specified routes. This SR information should be inserted at the end host servers hosting the virtual instance endpoints, since identifying the flow's required QoS is not possible when the packets leave the server due to containerization. Specifically, we need to be able to (*i*) collect flow level metrics and send them to the TE control plane; (*ii*) acquire TE results from the TE control plane and label outgoing packets with SR details based on TE decisions. However, these desired functions introduce unique challenges to today's TE systems. In particular, the first task requires accurate flow identification at the level of containers or VMs. The second task involves end host servers adding TE results to packet headers such that the subsequent routers/switches are able to forward the packets as designated.

Combating TE optimization complexity on the control plane. Another core challenge in MegaTE is the computational complexity involved in formulating optimal TE decisions at the control plane. Upon receiving data from the endpoints, the controller is tasked with determining traffic allocations among all pairs of nodes, which may be virtual machines or containers. Traditional TE systems manage aggregated traffic on only hundreds or up to a few thousand nodes. In contrast, MegaTE must manage the traffic between all pairs of endpoints. The presence of millions of endpoints magnifies the problem scale by three orders of magnitude. Additionally, the indivisible nature of endpoint traffic renders the TE optimization problem NP-hard. To tackle this issue without compromising accuracy, a novel algorithm that accommodates the new TE formulation and effectively reduces the problem's scale is imperative.

3.2 Rearchitecting the TE Control Loop

In conventional TE systems (as shown in Figure 3(a)), at every TE interval, the routers in the TE data plane collect the network states at the granularity of aggregated flows and send them to the bandwidth broker on the TE control plane. After that, the bandwidth broker forwards the network topology and traffic demand to the TE optimizer, which performs the optimization computation to obtain traffic flow allocations. The TE controller receives the optimized flow allocations and pushes them back to the data plane through persistent connections.

Extending conventional TE's control loop. As MegaTE extends the WAN TE to involve millions of endpoints to enable the management of flows from millions of virtual instances on the cloud, an intuitive and simple approach is to directly extend the conventional



(a) Conventional TE runs only on routers with a top-down control loop.

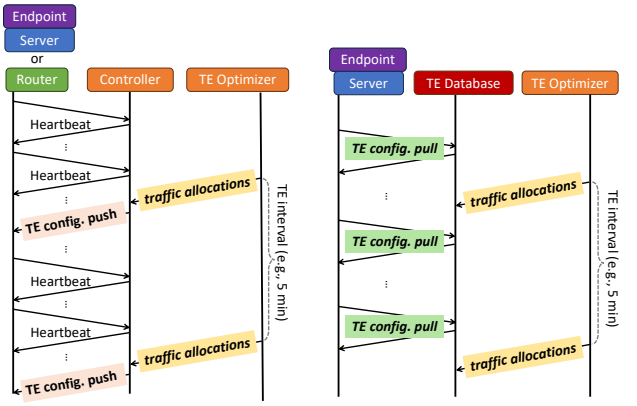
(b) MegaTE involves millions of endpoints with a bottom-up control loop.

Figure 3: MegaTE rearchitects TE control loop with asynchronous database query and endpoints involvement.

TE's control loop to endpoints. Specifically, as shown in Figure 4(a), each endpoint will initialize a persistent connection, establishing communications between the TE controller and the endpoints. The TE controller then pushes the TE configurations to the endpoints whenever necessary, e.g., during the regular TE configuration intervals or in response to a network failure. To ensure immediate synchronization of the TE configurations, the connections between the TE controller and the endpoints should be persistent. Therefore, a heartbeat packet is periodically generated by the endpoint and sent to the TE controller to keep the connection alive. This mechanism ensures that the connection remains active even if there are no TE configurations to be transferred. However, millions of endpoints require the controller to maintain millions of persistent connections to synchronize the TE configurations. This approach consumes a lot of resources and overwhelms the control plane.

Trade data plane consistency for control plane scalability. The conventional TE's top-down control loop that maintains a real-time synchronization between millions of endpoints and the control software will greatly hinder the control plane scalability. In contrast, a more attractive approach is to leverage distributed cloud databases to offload the millions of controller connections. As shown in Figure 3(b), MegaTE uses a bottom-up approach in which the TE controller is replaced by a TE database. The calculated TE configurations are queried by the endpoints in an asynchronous manner, rather than being pushed to the data plane via persistent connections. Therefore, the asynchronous query relaxes the requirement that all endpoints must update their TE configurations at exactly the same time, significantly improving the control plane scalability.

Towards eventual consistency. Different from conventional TE's control loop, MegaTE leverages distributed cloud databases and employs a bottom-up approach to achieve the eventual consistency [5] of TE configurations such that the controller does not need to maintain persistent connections with millions of endpoints. Specifically, as shown in Figure 4(b), the centralized controller updates the TE configurations periodically or whenever there is a network failure, and stores the new updates into the cloud databases with an incrementing version number. Each endpoint periodically queries the cloud databases to check for the latest TE configuration version using a short connection. Upon detecting a discrepancy between



(a) The top-down control loop needs to maintain a huge number of persistent connections to push TE configurations.

(b) The bottom-up control loop allows endpoints to pull up-to-date TE configurations in a *connectionless* manner.

Figure 4: TE configuration synchronization.

the version number of its current TE configurations and that provided by the controller, the endpoint initiates a connection to the databases to pull the new TE configurations and proceeds to update its settings accordingly.

Highly concurrent key-value store as TE database. Given that MegaTE needs to handle frequent queries from millions of endpoints, we build an in-memory key-value database based on Redis [10] to support high-performance data read and write, particularly suitable for our scenario that requires high concurrency. Here, our customized database supports up to 160,000 concurrent queries per second using two shards. The performance of the database could be linearly scaled with more shard resources. In order not to introduce additional resources, but to use two shards to handle millions of queries, we divide all endpoints into several parts, and each part initiates queries asynchronously during a specific time period (e.g., 10 seconds). This approach will reduce the query loads at a specific time by equally spreading them over the timeline. As all TE configurations are loaded at the end of the time period, the endpoints update the configurations in the packet header. Despite the lack of immediate synchronization, all nodes in the network will converge on a consistent view of the network state over time.

Input	$G(V, E)$	Site topology with node set V and link set E.
	$k \in K$	k-th index of site pair set K.
	$i \in I_k$	i-th index of endpoint pair set I_k connecting k-th site pair.
	c_e	Bandwidth capacity of link $e \in E$.
	T_k	Set of pre-established tunnels for site pair k.
	d_k^i	Bandwidth demand of i-th endpoint pair connecting k-th site pair in a TE interval.
	$L(t, e)$	1 if tunnel t uses link e and 0 otherwise.
	w_t	Weight of tunnel $t \in T_k$.
Output	$f_{k,t}^i$	1 if the traffic d_k^i is routed on tunnel t, 0 otherwise.

Table 1: MegaTE notations.

4 MegaTE Control Plane

In this section, we first provide the formulation of our TE problem. Unlike a conventional TE formulation that is based on a multi-commodity flow (MCF) problem, the traffic flow at the virtual instance level has a binary state of accept or reject. This turns our problem into an NP-Hard decision problem which is difficult to solve (§4.1). We then propose a two-stage optimization algorithm to contract the network topology to make sure that the run time of our algorithm is within the TE update intervals (§4.2).

4.1 Problem Formulation

Different from prior TE systems [2, 15, 16, 19, 25, 35] where traffic flows are allowed to be split at the router sites over multiple paths, however, in MegaTE, we address the TE problem at the granularity of the virtual instance endpoint, where the traffic flows from the virtual instances cannot be arbitrarily divided. Since there will always be multiple flows between an endpoint pair, routing these flows from the same tenant over multiple paths would lead to unstable latency, degrading the application quality of experience. The introduction of the binary state of the traffic flow at the endpoint changes the traditional TE formulation into a new paradigm. We will describe it below in detail.

TE input. As listed in Table 1, we define network topology to be a graph $G = (V, E)$, where V and E represent the sets of nodes and edges, respectively. Let c_e represent the capacity of the edges. Here, as the capacity of the edges between the endpoint and the site is sufficient, we focus on the capacity of the edges between router site pairs. We then sequentially list and order all possible pairs of sites, using $k \in K$ as the index for each pair. For each site pair k, we pre-establish a set of paths (or TE tunnels) T_k to route traffic, where each tunnel t is assigned weight w_t. Here, w_t can be determined by the network latency where the higher value means larger network latency. Let $L(t, e)$ denote if the tunnel t uses the link e. For a specific site pair k, we enumerate and order all pairs of endpoints, one from each network site, assigning an index $i \in I_k$ to each of these endpoint pairs. There is a set of pairs of source-destination endpoints (or "flows") that connect the site pair k, where each pair of endpoints is associated with a demand d_k^i.

TE Output. Our goal is to determine the paths for flows from virtual instance endpoints to provide virtual-instance-level performance guarantees. As the flow from the instance cannot be split, we use a binary variable $f_{k,t}^i$ to indicate the path of the flow where 1 represents the flow d_k^i is routed on the tunnel t.

Optimization goal and constraints. When calculating bandwidth allocation for endpoint-to-endpoint traffic, our goal is to maximize the total satisfied network traffic across all demand pairs (which we refer to as `MaxAllFlow` problem). We present the formulation of our TE problem which is shown below.

$$\max_{f} : \sum_{k,i,t} d_k^i f_{k,t}^i - \epsilon \sum_{k,i,t} w_t d_k^i f_{k,t}^i \tag{1}$$

$$s.t.$$

$$\forall e : \sum_{k,i,t} d_k^i f_{k,t}^i L(t, e) \le c_e \tag{1a}$$

$$\forall k, i : \sum_{t} f_{k,t}^i \le 1 \tag{1b}$$

$$\forall k, i, t : f_{k,t}^i \in \{0, 1\} \tag{1c}$$

Notably, our objective function (1) is to maximize overall throughput while preferring shorter paths. ϵ is a small constant. Constraint (1a) states that no link should be overloaded. Constraint (1b) guarantees that each flow can only be allocated on at most one tunnel, ensuring that endpoint-to-endpoint flow is indivisible over paths. Constraint (1c) states that $f_{k,t}^i$ is binary and 1 represents the flow d_k^i is routed on the tunnel t. The introduction of integer variable renders `MaxAllFlow` problem as an NP-Hard problem. Readers can refer to Appendix A.1 for more details.

Although each endpoint flow connecting to the same router site pair will be routed to only one path, all flows connecting to the same site pair will most likely be assigned to different paths. The aggregated flows at the router site will appear to be split among paths, showing consistency with previous work [15, 16, 25].

TE among multiple QoS classes. We classify the traffic into three service classes. We refer to them as QoS classes 1 to 3, where QoS class 1 is the highest priority, which contains essential network control traffic and a few critical services such as cloud gaming. These applications are always time-sensitive; QoS class 2 is for most of user application traffic and internal application traffic; and QoS class 3 is for heavy and bulk data transfer, such as logs. Since combining all QoS traffic together will make the problem size prohibitively large, similar to [19], we resort to separating the traffic to reduce the solving time of the optimization problem. Specifically, we determine bandwidth allocation rate by invoking `MaxAllFlow` separately for QoS classes in priority order. Once a higher QoS class is allocated, the remaining capacity of link e is updated by $c_e = c_e - \sum_{k,i,t} d_k^i f_{k,t}^i L(t, e)$, which is then used for the lower QoS class to calculate the optimal bandwidth allocation.

4.2 Two-Stage Optimizaiontion Algorithm

Despite recent advances such as NCflow [2] and TEAL [44] accelerated TE calculation on the large network topology, these methods

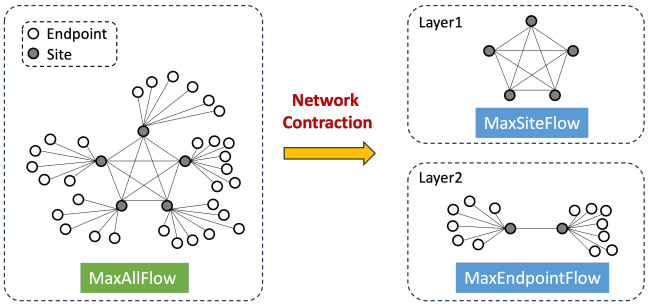


Figure 5: Traditional `MaxAllFlow` problem is hard to solve on the hyper-scale network topology as well as its NP-Hard nature. We contract the network in two layers and propose a two-stage optimization algorithm where we solve the `MaxSiteFlow` problem in the first layer and the `MaxEndpointFlow` in the second layer.

can only effectively address graphs featuring up to several thousand network nodes. MegaTE significantly increases the size of the network topology by three orders of magnitude to millions of endpoints, making existing TE solutions impractical to address such a hyper-scale network. Meanwhile, the NP-Hard nature of MaxAllFlow problem further hinders existing TE approaches to provide the TE allocations in a short time.

Contracting the Problem Size. To address this challenge, a more attractive approach is to contract the network topology to reduce the scale of the problem and facilitate parallel computation. This approach has been proposed in previous work such as POP [36] and NCFlow [2]. However, POP [36] does not fit our scenario since these traffic flows whose originated endpoints connect to the same sites should be split into the same sub-problem and the random partitioning in POP could drop these flows into different sub-problems. We embrace an approach akin to that of NCFlow [2]. However, different from NCFlow, a better way can be applied to contract the network topology. As shown in Figure 5, an important observation in MegaTE network topology reveals that it consists of two distinct layers of connectivity, where the first layer represents a highly meshed topology among the network sites and the second layer includes the network sites, each forming a hub that connects to multiple endpoints. These connections are singular and direct, with each endpoint linked to only one network site. Based on this observation, we simply separate the graph into two layers following the above description and propose a two-stage optimization algorithm to solve the `MaxAllFlow` problem on each layer separately where we solve the `MaxSiteFlow` problem on the first layer and the `MaxEndpointFlow` on the second layer.

- `MaxSiteFlow`: On the first layer, we first aggregate the traffic flows of endpoint pairs connecting to the same site pair to generate the traffic demands D_k at the site level (line 1-6). In this way, the flow allocation for the first layer, referred to as `MaxSiteFlow`, then aligns with the optimization objectives found in existing TE studies, where it becomes an MCF (multi-commodity flow) problem and the number of nodes is a few hundred or fewer. It can thus be easily solved using existing TE approaches [2, 19] to obtain the bandwidth allocation $F_{k,t}$ between site pair k and on the tunnel t (line 7-9).

Algorithm 1: Two-stage optimization

Input: $\{d_k^i\}, \{c_e\}, \{T_k\}, \{L(t,e)\}, \{w_t\}$ ▷ An endpoint decision problem
Output: $\{f_{k,t}^i\}$ ▷ Solution

1 **Function** `SiteMerge`($\{d_k^i\}$):
2 **foreach** $k \in K$ **do**
3 $D_k \leftarrow \sum_i d_k^i$ ▷ Aggregate the demands at site level
4 **end**
5 **return** $\{D_k\}$
6 **End Function**
7 **Function** `MaxSiteFlow`($\{D_k\}, \{c_e\}, \{T_k\}, \{L(t,e)\}, \{w_t\}$):
8 Solve LP:

$$\begin{aligned} \arg\max_{F} \quad & \sum_{k,t} F_{k,t} - \epsilon \sum_{k,t} w_t F_{k,t} \\ s.t. \quad & \sum_{t} F_{k,t} \le D_k, && \forall k \\ & \sum_{k,t} F_{k,t} L(t,e) \le c_e, && \forall e \\ & F_{k,t} \ge 0, && \forall k,t \end{aligned}$$

9 **return** $\{F_{k,t}\}$
10 **End Function**
11 **Function** `MaxEndpointFlow`($F_{k,t}, \{d_k^i\}$): ▷ Parallelizable
12 $m_{k,t} \leftarrow$ `FastSSP`($F_{k,t}$, $\{d_k^i\}$)
13 **Transform** $\{f_{k,t}^i\} \leftarrow m_{k,t}$
14 **return** $\{f_{k,t}^i\}$
15 **End Function**

- `MaxEndpointFlow`: In the second layer, we need to solve the traffic allocation for the endpoint pairs. Specifically, for a specific site pair k, given the bandwidth allocation $F_{k,t}$ and bandwidth demand set $\{d_k^i\}$, the `MaxEndpointFlow` problem is to select a subset of the bandwidth demands whose total traffic is closet to, without exceeding, $F_{k,t}$. The `MaxEndpointFlow` problem is a typical subset sum problem (SSP), which is a special scenario of the Knapsack problem. Note, the `MaxEndpointFlow` problem with different site pairs can be solved in parallel to accelerate the TE optimization. Due to its NP-Hardness, various algorithms have been introduced to tackle this problem in pseudopolynomial time, with dynamic programming (DP) [6] being a typical method of choice. However, the DP approach is not ideal due to its high complexity when dealing with small values of endpoint pair demands d_k^i against a large number of endpoint pairs $|I_k|$ as well as a large value of site-pair bandwidth allocation $F_{k,t}$. [3] Therefore, we propose a novel semi-DP technique that significantly reduces complexity while allowing for controllable precision in the solution. We refer to this algorithm as `FastSSP`, which is an approximation of the optimal solution.

The `FastSSP` algorithm (line 12) operates through a structured four-step process. Step 1 (*Clustering*): Initially, it groups all endpoint

[3] The time complexity of the DP is $O(|I_k|F_{k,t})$.

flows into m large traffic demands where each traffic demand meets or exceeds a threshold M, setting the stage for subsequent normalization. Here, m is a small integer, determined by M. Step 2 (*Normalization*): Next, the aggregated demands are normalized with a factor δ, simplifying the problem and reducing the computational complexity by δ faced by the DP methods. Step 3 (*DP solving*): Following this, we employ a traditional DP strategy [6] to address the SSP challenge to obtain the optimal solution. Here, the time complexity of DP reduces from $O(|I_k|F_{k,t})$ to $O(m\lfloor\frac{F_{k,t}}{\delta}\rfloor)$. Step 4 (*Sorted-based greedy algorithm*): The final step involves managing the minor flows excluded after the DP computation phase. Although this remains an SSP issue, the aggregate demand of these residual flows is relatively minor, meaning any suboptimal allocations will not significantly impact the overall solution. Consequently, we implement a sorting-based greedy algorithm for the efficient allocation of these residual flows. The time complexity is $O(|I_k|log|I_k|)$. Please refer to Appendix A.2 for more details.

In this way, we obtain a set of endpoint pairs allocated with network bandwidth, represented as $m_{k,t}$ by solving the `FastSSP`. We finally transform the set $m_{k,t}$ into a set of $f^i_{k,t}$ to indicate if traffic d^i_k is routed on tunnel t (line 13).

Overall Procedure. The overall procedure for the two-stage optimization algorithm is outlined in Algorithm 1. Initially, we contract the network topology into two distinct layers, leveraging our observation of topology's inherent characteristics. The first layer deals with flow allocations at the site level, i.e., `MaxSiteFlow`, mirroring conventional TE strategies. The solution of the first layer is adopted as the input for the subproblem `MaxEndpointFlow` at the second layer, which is a series of SSP problems that we introduce an innovative approximation algorithm `FastSSP` to address.

5 MegaTE Data Plane

In this section, we present the data plane implementation of MegaTE. In particular, we first describe the eBPF-based host stack for instance identification and instance-level flow collection (§5.1). Then, we introduce segment routing to route packets from the endpoints (§5.2).

5.1 eBPF-Based Host Networking Stack

In the virtualized cloud, the dynamic provisioning and decommissioning of millions of virtual instances on the application layer result in the fact that multiple flows of the same tenant have different identifiers in five tuples <`src_ip`, `dst_ip`, `proto`, `src_port`, `dst_port`> of IP packets. These flows from the same tenant can not be distinguished on the router sites, limiting the conventional TE systems [12, 20, 25] to provide network resources in a way that aligns with the specific needs of each virtual instance. Meeting the QoS demand for each virtual instance requires the insertion of SR information into the packet header in end hosts, enabling WAN routers to identify and adhere to the specified routes.

eBPF [14, 34] enables user-space applications to customize complex operations inside the kernel, such as packet tracing and packet processing [18, 45]. Each eBPF program should be attached to a hook, such as kernel probes, kernel tracepoints, and traffic control (TC) [33], which will be triggered by a kernel event. The eBPF maps are generic key-value stores used to store eBPF program states,

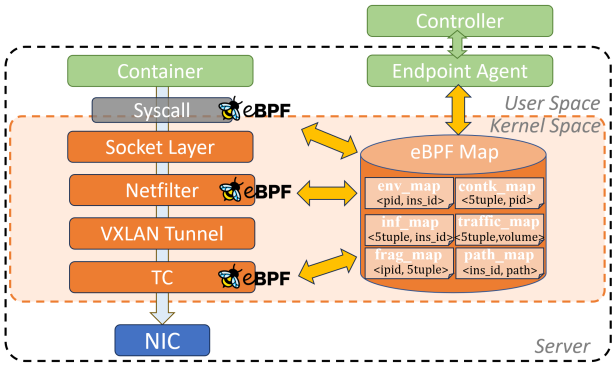
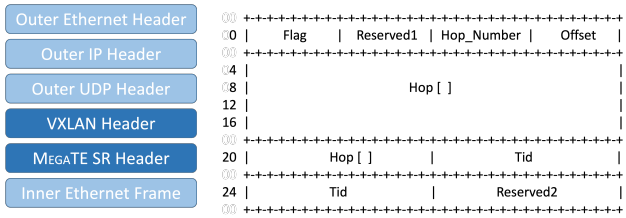


Figure 6: eBPF implementation on MegaTE.

enabling communications among various eBPF programs and between eBPF programs and user-space processes. In each end host, there is an endpoint agent, which is used for the interaction between the controller and endpoint. The powerful functionality of eBPF provides a promising way to achieve instance-level segment routing in end hosts. We will describe the implementation of eBPF in detail below.

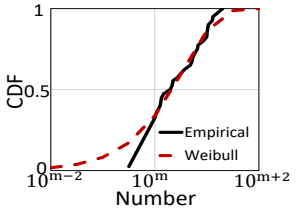
Instance identification. Mapping the packet in the TC layer to its originated virtual instance is critical for achieving the instant-level flow collection and segment routing at the virtual instance. However, it is hard to directly identify the packet using five tuples at the TC hook to its originated instance. To address this, as shown in Figure 6, we attach the eBPF programs at different kernel hooks to obtain the related information. Specifically, the eBPF program is attached at `syscalls/sys_enter_execve` tracepoint hook to collect the process ID (*pid*) and instance ID (*ins_id*) once the instance runs a process, and then stores them into the eBPF map named `env_map`. When the process creates a socket connection for packet transmission, it will trigger another eBPF program attached at `kprobe/ctnetlink_conntrack_event` to collect five tuples (*5tuple*) and process ID (*pid*) of the connection and store them into the eBPF map named `contk_map`. Then this eBPF program combines the `env_map` and `contk_map` to obtain `inf_map` with the key and value being *5tuple* and *ins_id* respectively. Therefore, each five-tuple item of a packet will be accurately mapped to its originated virtual instance.

Instance-level flow collection. The instance-level flow collection module consists of two parts: (*i*) a kernel-space eBPF program for profiling packets and storing flow statistics into the `traffic_map`, and (*ii*) a user-space process executed periodically to read the data from the eBPF map. Specifically, as the TC layer is the closest layer to the NIC that has access to the entire Ethernet frame, we attach the eBPF program at the TC layer to precisely capture the packet information. Once a packet arrives at the TC layer, the eBPF program is triggered to process the packet to obtain the packet statistics (e.g., five tuples, packet bytes). If the five-tuple field of the packet is not found in the `traffic_map`, the eBPF program will add a new item in the map with the key and value being the *5tuple* and traffic *volume* respectively. Otherwise, the eBPF program updates the traffic volume on the map by counting the packet size. For a large packet with a size exceeding the maximum transmission unit (MTU), it



(a) Header hierarchy. (b) SR Header.

Figure 7: Segment routing header design.



Topology	Sites	Endpoints
B4*	12	120,000
Deltacom*	113	1,130,000
Cogentco*	197	1,970,000
TWAN	O(100)	O(1,000,000)

Figure 8: Endpoint number. Table 2: Network Topology.

should be fragmented into several fragments and these fragments share the same identification (*ipid*) [39]. The eBPF program stores the traffic volume in the `traffic_map` and *ipid* and *5tuple* in the `frag_map` when it receives the first fragmentation. Upon receiving subsequent fragmentations, the eBPF program obtains the *5tuple* of the fragmentations by mapping the *ipid* in the `frag_map`, and then updates the traffic volume of that *5tuple* in the `traffic_map`. The endpoint agent will periodically (e.g., a TE period [19]) initiate a user-space process to read the instance-level flow data containing a tuple of *ins_id* and *volume* by combining the `inf_map` and `traffic_map` and store them into the backend server.

5.2 Segment Routing

Segment routing information insertion. In the cloud environment, Virtual Extensible LAN (VXLAN) [32] technology is adopted for communications among different containers. As shown in Figure 7(a), each Ethernet frame has a VXLAN header added and is then encapsulated in a UDP-IP packet. Once the centralized controller calculates the optimal path decision for each instance after the TE optimization, it will notify the endpoint agent to initiate a process to store the instance ID and packet path in the eBPF maps, namely `path_map` (see Figure 6). Upon receiving a packet, the eBPF program attached at the TC layer firstly captures the five tuples to obtain the optimal path of the packet from the eBPF maps by operating on the `inf_map` and `path_map`. The path of the packet is then inserted into the packet header, represented as the MegaTE SR header. Here, the SR header is inserted subsequent to the VXLAN header. Figure 7(b) shows the specifications of the SR header designed for packet routing. The "Hop_Number" field represents the total number of hops; the "Hop[ ]" array is a sequence of next hops, specifying the detailed path of the packet traversing the WANs; the "Offset" field is the current offset of the hop in the hop[ ]. Meanwhile, the eBPF program will also insert a flag in the "Reserved" field of the VXLAN header to indicate whether the packet is inserted with the MegaTE SR information.

Router implementation. The router site profiles the packet and analyzes the VXLAN header to identify if the packet uses MegaTE SR information. If it is identified as a MegaTE SR header, the router obtains the hop information from the SR header and forwards the packet to the specified path.

6 Evaluations

In this section, we first describe the experiment setup (§6.1). Next, we compare MegaTE with state-of-the-art TE schemes to show that MegaTE supports a larger number of endpoints with near-optimal flow allocations (§6.2), and demonstrate the robustness of MegaTE to handle link failures (§6.3). Finally, we show that MegaTE introduces negligible overhead when connecting millions of endpoints (§6.4).

6.1 Experimental Setup

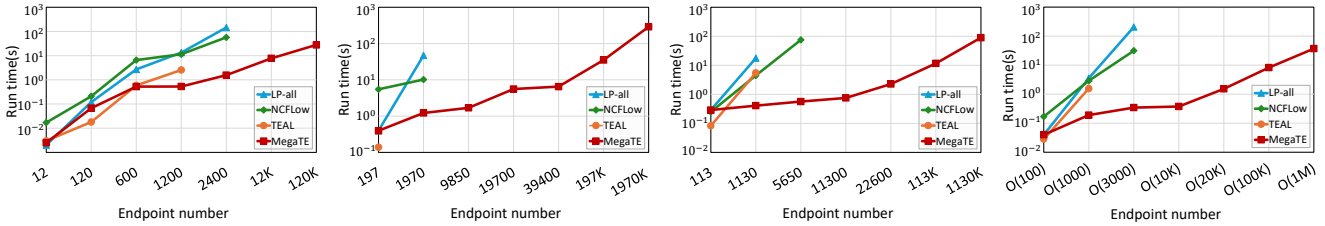
Network topologies at the granularity of endpoints. We study the distribution of the endpoint number in the production WAN that a router site connects. Figure 8 plots the CDF for the endpoint number across the router sites, derived by the empirical traces of the TWAN network. The x-axis represents the endpoint number connecting the router site, parameterized by m.[4] An important observation is that the number of endpoints that a router site connects varies significantly in orders of magnitude. We use a Weibull probability distribution to fit the empirical data collected from TWAN. We then consider four topologies ranging from tens to hundreds of router sites: our production TWAN, B4* [25], Deltacom* [1] and Cogentco* [1]. Here, * indicates that we have made the modification to the original topology by adding endpoints to sites. The number of sites and the maximum number of endpoints of these topologies are summarized in Table 2. Then we change the scale parameter of our Weibull distribution to study the impact of the endpoint number that a site connects on the network performance.

Traffic matrices at the granularity of endpoints. We collected instance-level flow data from endpoints for a typical day from TWAN. The flow data observed during each TE period (e.g., 5 minutes) between each source-destination endpoint pair is regarded as their traffic demand. To generate instance-level traffic demand on other network topologies, i.e., B4*, Deltacom*, and Cogentco*, we firstly map each new site pair to a random site pair in TWAN, and then map the instance-level traffic demand in each new endpoint pair connecting to the site pair to a random endpoint pair connecting the mapped site pair in TWAN. For different topology scales (i.e., different numbers of endpoints connecting to a router site), we randomly select the traffic demands from endpoint pairs connecting to the same site pair.

TE Benchmark schemes. We compare our proposed MegaTE against several state-of-the-art TE schemes that aim at solving TE problems with large topologies quickly.

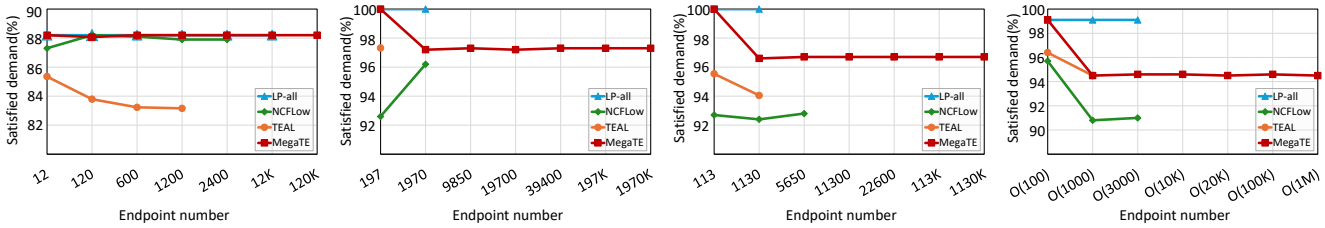
- *NCFlow* [2]: NCFlow divides the network topology into multiple disjoint clusters and solves the TE optimization subproblem in

[4]The exact value of m is omitted for confidentiality.



(a) B4* (b) Cogentco* (c) Deltacom* (d) TWAN

Figure 9: [Simulation] TE algorithm run time on four different network topologies.



(a) B4* (b) Cogentco* (c) Deltacom* (d) TWAN

Figure 10: [Simulation] Satisfied demand on four different network topologies.

each cluster in parallel, and the results from these clusters are merged to obtain a global allocation.

- *TEAL* [44]: TEAL is a learning-based TE algorithm that utilizes the graph neural network (NN) and Alternating Direction Method of Multipliers (ADMM) to generate a valid global allocation.
- *LP-all*: LP-all scheme is a linear programming (LP) algorithm that solves the multi-commodity flow (MCF) problem for the demands between endpoints.

Metrics. We consider the following performance metrics for TE.

- *Computation time*: We measure the average elapsed time required by each approach to compute the flow allocation among all traffic matrices. The measurement of TEAL is carried out using an additional GPU (Nvidia A30) following the setup in [44] while the rest is carried out on the server with 24 CPU threads (Intel Xeon Gold 5317) and 128GB of memory. We use the Gurobi to solve the LP-all, NCFlow, and MegaTE. The computation time of TEAL is the pure GPU run time, and LP-all, NCFlow, and MegaTE is the Gurobi solver run time.
- *Satisfied demand*: We use the satisfied demand to evaluate these TE schemes that optimize the total flow which is represented as the ratio of the total guaranteed demand to the total traffic demand across all traffic matrices.
- *Packet latency*: To measure the packet latency experienced by virtual instances' traffic flows, we sum the measured latency for each hop along the path for TWAN. For other topologies, i.e., B4*, Deltacom*, and Cogentco*, we simplify the packet latency as the number of hops in the network.

6.2 MegaTE vs. the state-of-the-art

We compare MegaTE with the state-of-the-art TE schemes on four network topologies. MegaTE achieves more than 20× larger network topology with a similar optimization run time compared to state-of-the-art TE systems (Figure 9). Meanwhile, MegaTE does not lose the optimality to achieve high satisfied demand even if the number of endpoints increases to millions (Figure 10). Furthermore, MegaTE achieves a fine-grained traffic allocation to reduce the packet latency for time-sensitive services (Figure 11).

TE computation time. Figure 9 presents the TE computation time as the network scale increases among four typical networks. On the small topology with about 100 endpoints, almost all TE schemes obtain the optimal TE decisions within one second. TEAL performs the best since it only needs to perform one forward pass on the NN model and several iterations of ADMM to obtain TE decisions, while the rest of TE schemes spend a certain amount of time by (integer) linear optimization-based solvers. As the network topology scales to thousands of endpoints, the performance varies among these TE schemes. For example, as for Deltacom*, the LP-all still completes the TE computation for the network with 1130 endpoints and it takes 18 seconds to complete the TE computation. This is attributable to the special structure of our network topology which greatly reduces the solution time. Both NCFlow and TEAL reduce the computation time to 5 seconds for the network topology with 1130 endpoints since NCFlow contracts the network into clusters and solves these clusters in parallel, and TEAL leverages NN and ADMM to accelerate the TE computation. In contrast, MegaTE completes the flow allocation among 22,600 endpoints with a lower optimization run time (i.e., 2 seconds) compared to NCFlow and TEAL, achieving more than 20× larger network topology. As for hyper-scale network topologies with tens of thousands to millions of endpoints, traditional TE schemes are not practical for performing TE computation since they will encounter out-of-memory issues due to a lot of memory consumption during the computation. MegaTE contracts the network topology to reduce

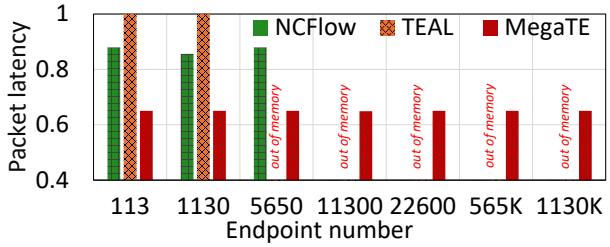


Figure 11: [Simulation] Packet latency in Delatcom*.

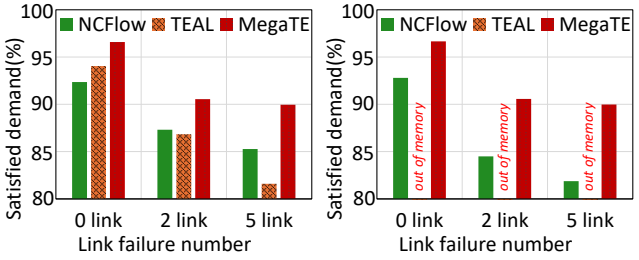
the scale of the problem and facilitate the parallel computation, and further introduces `FastSSP` algorithm to solve the problem. MegaTE completes the flow allocation with tens of seconds even if the network scales up to O(1,000,000) endpoints in TWAN.

Satisfied demand. Figure 10 presents the satisfied demand as the network scale increases among four typical networks. We observe that MegaTE retains a demand that is close to the optimal. For example, with 120 endpoints in B4*, MegaTE satisfies a demand of 88.1%, which is slightly lower than the optimal (vs. 88.2% for LP-all). As the network topology scales up to thousands of endpoints, such as 1130 endpoints in Deltacom*, both NCFlow and TEAL only satisfy 92.4% and 94.0% of the demand, respectively. The main reason is that these schemes overemphasize accelerating the TE computation and they always lose the optimality to support more satisfied demand. In contrast, MegaTE maintains 96.8% satisfied demand. Furthermore, we observe that MegaTE does not lose optimality even if the network topology scales to millions of endpoints, showing less degradation compared to the optimal results of LP-all. MegaTE contracts the network topology where the first layer leverages LP to compute the optimal aggregated traffic distribution and then allocates the bandwidth within the cluster using `FastSSP`. The `FastSSP` is an approximation of the optimal solution. Therefore, MegaTE is always optimal even if the network becomes larger with a larger number of connected endpoints.

Packet latency. Figure 11 shows the normalized packet latency of QoS classes 1 (i.e., time-intensive services) of a typical site pair in Deltacom. We observe that both NCFlow and TEAL perform poorly on these high-priority and time-intensive services. MegaTE achieves low latency, reducing the time by 25% and 33% compared to NCFlow and TEAL, respectively. The main reason is that previous approaches make TE decisions between router sites based on aggregated traffic. Once the aggregated traffic contains the flow with multiple classes, the higher class will be mistakenly allocated to the path with larger network latency. In contrast, MegaTE enables a finer-grained scheduling of network traffic at the endpoint level. Each flow could be accurately assigned to the appropriate path to meet its demand. For example, the flow from these time-sensitive applications will be allocated to the shortest path.

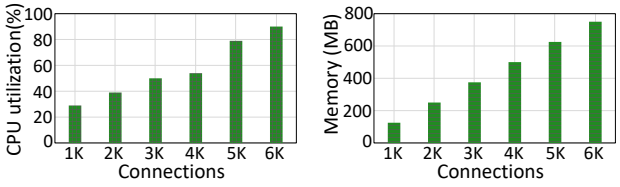
6.3 MegaTE Under Failures

MegaTE can obtain the flow allocation in seconds, even for hyper-scale topologies with millions of endpoints. The real-time computation allows MegaTE to efficiently address link failures [2, 31, 44], as MegaTE can quickly recompute flow allocation on the altered topology. Figure 12(a) shows the satisfied demand of TE schemes in the presence of different numbers of link failures (e.g., 2 and 5



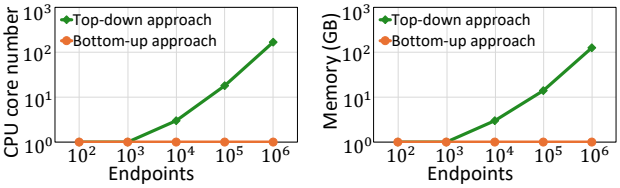
(a) Endpoint=1130 (b) Endpoint=5650

Figure 12: [Simulation] Satisfied demand under failures in Delatcom*.



(a) CPU core utilization (b) Memory usage

Figure 13: [Simulation] CPU utilization and memory usage.



(a) CPU core (b) Memory usage

Figure 14: [Simulation] CPU core counts and memory usage.

link failures) in Delatcom*. For the network with 1130 endpoints, we observe a stable gap (about 4%) of the satisfied demand among NCFlow and MegaTE since they perform TE recomputation once the link fails, and the new flow allocation is computed within seconds to avoid packet loss during the recomputation period. As the network topology becomes larger with a total of 5650 endpoints in Figure 12(b), we observe that the gap between the satisfied demand by NCFlow and MegaTE increases, reaching a value of 8.2%. The main reason is that NCFlow takes about 100 seconds to recompute the new traffic allocation. A large portion of network flows that have traversed the failed links will be dropped during the TE recomputation period. In contrast, MegaTE takes less than one second to recompute new traffic allocation upon link failures, satisfying more traffic demand than NCFlow. As for the hyper-scale topology with millions of nodes, previous TE approaches such as NCFlow and TEAL require a large memory size and a long time to execute the new TE policies, and are not practical. In contrast, MegaTE can still react to these failures in a timely manner (i.e., tens of seconds) to maintain a high satisfied demand.

6.4 MegaTE Synchronization Overhead

In the traditional TE system, [2, 8, 19, 20, 29, 31, 37, 40, 42, 46], the TE's control loop is top-down that the SDN controller only needs

to keep a persistent connection with each router to synchronize the TE policies. In such a system, synchronizing TE configuration is less of a concern because the controller only needs a manageable number of routers or switches, ranging from tens to a few thousand. However, the problem becomes challenging if we extend the top-down control loop to millions of endpoints. We evaluate the overhead of the conventional top-down control loop and MegaTE's bottom-up control loop.

As the control software is deployed in the cloud for robust consideration, we use the same cloud environment to faithfully evaluate the overhead in a real-world setting. We conducted a pressure test on CPU utilization and memory usage using a virtual machine with 1 CPU core and 1 GB memory, while varying the number of persistent connections. Figure 13 shows the CPU and memory overhead as the number of persistent connections increases. When the number of connections is 6,000, the CPU utilization reaches 90% and memory usage amounts to 750 MB. Based on the feedback from network operators, consistent and sustained high CPU utilization (e.g., 90%) will cause the risk of failures.

Figure 14 shows the number of CPU cores and memory usage as the number of endpoints increases. We observe that more resources are required as the number of endpoints increases. When the number of endpoints is 1,000, we can adopt the conventional top-down approach which maintains persistent connections between the controller and endpoints to synchronize the TE policies since this approach only consumes little resources. However, as the endpoint number increases to one million, maintaining persistent connections between the controller and all endpoints is quite resource-consuming, requiring at least 167 CPU cores running at high usage and 125 GB of memory. A more attractive approach is the bottom-up approach which offloads the millions of controller connections into database queries by leveraging the distributed cloud databases. We only need to use 1 CPU core and 1 GB of memory to synchronize the TE configurations to the database, which greatly reduces the load on the controller.

7 Production Deployment at Tencent WAN

MegaTE has been deployed in the production WAN of Tencent to serve millions of tenants since December 2022. We compare our MegaTE with our traditional TE approach, which does not differentiate the QoS class of endpoint flows, and directly distributes the traffic in the backbone using Multi-Commodity Flow (MCF). We use several real-world cases to evaluate MegaTE in production. Compared to the traditional approach, MegaTE reduces the latency for time-sensitive applications at most by 51%, ensuring availability for high-priority applications with an average of 99.995%, and reduces the cost by 50% for the low-priority applications.

Latency reduction for time-sensitive applications. Figure 15 shows five typical time-sensitive network applications with video streaming (App 1), live streaming (App 2), real-time message (App 3), financial payment (App 4), and online gaming (App 5). The y-axis represents the latency which is not disclosed for confidentiality reasons. We observe that MegaTE can reduce the network latency for all of these time-sensitive applications. Especially for APP1, the latency is reduced by more than 51%. As the traditional approach always allocates the aggregated network traffic at each site, some

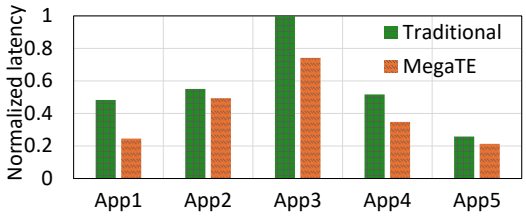


Figure 15: [Production] Packet latency reductions.

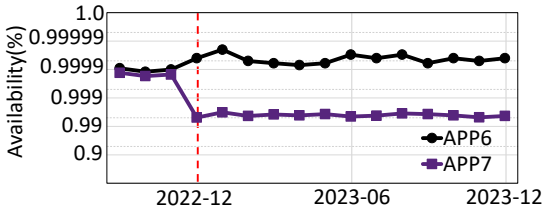


Figure 16: [Production] Customized service availability.

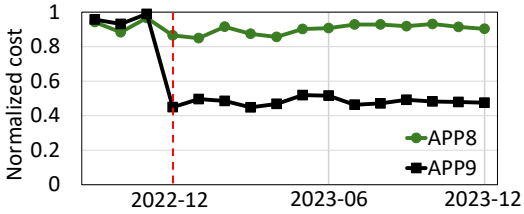


Figure 17: [Production] Cost reductions.

of the time-sensitive network traffic will be allocated to the long paths. In contrast, our proposed MegaTE enables to manage the network traffic at the endpoint, and thus meets the requirements of these time-sensitive applications.

Availability guarantee for high-priority traffic. Figure 16 shows the availability of two applications with App 6 belonging to QoS class 1 (99.99%) and App 7 belonging to QoS class 3 (99%). Upon the deployment of MegaTE in December, 2022, we observed that there is a large gap in their availability. But neither of them breaks the availability requirements. Specifically, we observe that the traditional approach may break up the availability requirements for App 6 with the availability of 99.988% in October, 2022. After the deployment of MegaTE, the average availability of App 6 maintains more than 99.995%, greatly exceeding the availability requirements. The main reason is that the flows of App 6 are precisely allocated to the high availability path. As for App 7, the generated flows are dispatched to a lower availability path which still meets the availability requirements.

Low cost for low-priority application. Figure 17 shows the cost of two typical applications. App 8 is an online gaming application (QoS class 1) and App 9 is a bulk transfer application (QoS class 3). Since the initial system cannot differentiate traffic with multiple priorities, all flows will be routed to the high-availability path to ensure the availability of high-priority applications. However, this operation will introduce additional costs for these low-priority applications. Upon the deployment of MegaTE, we observe that the cost of App 9 is reduced by 50%. The main reason is that the traffic

generated by the bulk transfer application is accurately dispatched to the low-cost path and the cost of the traffic per Gbps is reduced.

8 Discussions

TE with application-level statistics. MegaTE operates under a model of weak coupling with applications, where our scheduler makes decisions based solely on the observed ongoing traffic bandwidth. However, recent studies [3] have suggested considering TE with strong application coupling, where the flow sizes for a significant portion of the traffic are known in advance. These flow sizes can also be predicted through various methods [7]. Having such knowledge about flows presents an opportunity to make more informed TE decisions [26], though it also introduces complexity to the modeling process [3]. We plan to explore the incorporation of additional flow characteristics into our TE model in future work.

Parallelism in SSP. The number of SSPs that need to be solved in MegaTE is $O(N^2)$, where N represents the number of site nodes. In large-scale networks, the total computation time is significantly increased due to the limitation of CPU threads. Improving the parallelism of SSP, using a similar approach as Teal [44], to reduce processing latency with GPU acceleration will be our future work.

Accelerating MaxSiteFlow solving. Although MegaTE efficiently handles special network topologies with millions of leaf nodes, the solving time for MaxSiteFlow increases significantly when there are a large number of site nodes. A synergy between NCFlow or TEAL and SSP to accelerate the solving of MaxSiteFlow is also worth further investigation.

Hybrid approach on TE configuration synchronization. The eventual consistency approach greatly reduces resource consumption compared to the conventional approach. However, it takes several seconds to synchronize the TE configurations among all endpoints in failure scenarios, causing a part of the traffic loss during the synchronization period. Our measurements in TWAN show that a small part of the flows account for most of the network traffic. A hybrid approach that maintains persistent connections for these heavy-traffic endpoints and performs eventual consistency for the rest of the endpoints will be our future work.

9 Related Work

Traffic engineering. TE, as one of the core networking problems, has been widely studied in both industry and academia with the goal of maximizing network throughput, guaranteeing load balance among links, and failure resilience [2, 8, 19, 20, 26, 29, 31, 37, 40, 42, 46]. As network scale and traffic matrices increase, the time required to obtain TE decisions has become a bottleneck in the control loop. Some works focused on accelerating the TE computation time on large network topology [2, 36, 44]. However, these works only addressed the network topology with thousands of nodes and will suffer from long computation time and excessive use of network resources (e.g., tens of thousands of GPUs) when being applied to a million-node network. In contrast, MegaTE contracts the network topology where the first layer leverages the LP to compute the optimal aggregated traffic distribution and then allocates bandwidth within the cluster using FastSSP. Therefore, MegaTE only uses tens of seconds to calculate the flow allocation with millions of endpoints.

eBPF-based host networking stack. eBPF [14] is a general kernel technology evolved from BPF [34], which enables kernel programs in a secure, high-performance and scalable manner. It has opened up many new application scenarios in telemetry [17, 18], networking [45], security [43] and other fields. In terms of telemetry, Millisampler [17] accurately collected burst traffic data on the host with high precision, resolving the contradiction between significant performance overhead and collection accuracy brought about by traditional collection methods. PowerTCP [18] not only supported the collection of TCP INT signaling, but also achieved more accurate congestion control through eBPF without modifying network devices and invasive impact on host services. In networking, the load balancer by Katran [45] has been widely deployed in production network environments. In contrast, MegaTE uses eBPF to collect flow statistics at the instance level and add TE results to packet headers to route traffic flows at the instance level.

Knapsack problem and SSP. Both fall in the scope of combinatorial optimization, which consists of finding an optimal object from a finite set of objects. Early research includes the renowned dynamic programming algorithm [6], partition methods that accelerate dynamic programming algorithms [22], and quantization approximation methods with controllable errors [30]. All of these methods provide the final solution. Currently, mainstream research on SSP focuses on answering whether a solution exists [9, 13, 27], in near-linear pseudopolynomial time. In contrast, MegaTE utilizes the `FastSSP` to significantly reduce the time complexity.

10 Conclusion

We propose and implement a first-of-its-kind TE system, called MegaTE, to adapt to the requirements of each fine-grained flow between virtual instances. At the core of MegaTE is the paradigm shift from the *top-down* centralized control to *bottom-up* asynchronous database query in TE's control loop, combined with eBPF-based segment routing in endpoints and network contraction in TE optimization, to scale to support millions of endpoints. We evaluate MegaTE using large-scale simulations with production traffic traces. Our simulation results show that MegaTE supports 20× larger network topology with a similar optimization run time compared to prior work. MegaTE has been rolled out in a global-scale public cloud provider since December 2022. Our production results show that MegaTE reduces the packet latency for time-sensitive applications by 51% and reduces the cost by 50% for low-priority applications.

This work does not raise ethical issues.

Acknowledgment

We sincerely thank our shepherd Behnaz Arzani and anonymous SIGCOMM reviewers for their insightful comments. We are grateful to Sirui Li, Jaichen Dong, and Zhihao Wang for their help with the simulation evaluations. We also thank the software engineers, network architects, and senior management of the engineering teams at Tencent for their support and effort to deploy MegaTE to the production WAN. Chuanchuan Yang is the corresponding author.

References

[1] 2021. The Internet Topology Zoo. http://www.topology-zoo.org/.

[2] Firas Abuzaid, Srikanth Kandula, Behnaz Arzani, Ishai Menache, Matei Zaharia, and Peter Bailis. 2021. Contracting Wide-area Network Topologies to Solve Flow Problems Quickly. In *18th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 21)*. 175–200.

[3] Behnaz Arzani, Siva Kesava Reddy Kakarla, Miguel Castro, Srikanth Kandula, Saeed Maleki, and Luke Marshall. 2023. Rethinking Machine Learning Collective Communication as a Multi-Commodity Flow Problem. *CoRR* abs/2305.13479 (2023). https://doi.org/10.48550/ARXIV.2305.13479 arXiv:2305.13479

[4] Siamak Azodolmolky, Philipp Wieder, and Ramin Yahyapour. 2013. Cloud computing networking: Challenges and opportunities for innovations. *IEEE Communications Magazine* 51, 7 (2013), 54–62.

[5] Valter Balegas, Sérgio Duarte, Carla Ferreira, Rodrigo Rodrigues, Nuno Preguiça, Mahsa Najafzadeh, and Marc Shapiro. 2015. Putting consistency back into eventual consistency. In *Proceedings of the Tenth European Conference on Computer Systems*. 1–16.

[6] Richard Bellman. 1957. *Dynamic Programming*. Princeton University Press.

[7] Theophilus Benson, Ashok Anand, Aditya Akella, and Ming Zhang. 2011. MicroTE: fine grained traffic engineering for data centers. In *Proceedings of the 2011 Conference on Emerging Networking Experiments and Technologies, Co-NEXT '11, Tokyo, Japan, December 6-9, 2011*. ACM, 8. https://doi.org/10.1145/2079296.2079304

[8] Jeremy Bogle, Nikhil Bhatia, Manya Ghobadi, Ishai Menache, Nikolaj Bjørner, Asaf Valadarsky, and Michael Schapira. 2019. TEAVAR: striking the right utilization-availability balance in WAN traffic engineering. In *Proceedings of the ACM Special Interest Group on Data Communication*. 29–43.

[9] Karl Bringmann. 2017. A near-linear pseudopolynomial time algorithm for subset sum. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms* (Barcelona, Spain) *(SODA '17)*. Society for Industrial and Applied Mathematics, USA, 1073–1084.

[10] Josiah Carlson. 2013. *Redis in action*. Simon and Schuster.

[11] Shawn Shuoshuo Chen, Keqiang He, Rui Wang, Srinivasan Seshan, and Peter Steenkiste. 2024. Precise Data Center Traffic Engineering with Constrained Hardware Resources. In *21st USENIX Symposium on Networked Systems Design and Implementation (NSDI 24)*. 669–690.

[12] Marek Denis, Yuanjun Yao, Ashley Hatch, Qin Zhang, Chiun Lin Lim, Shuqiang Zhang, Kyle Sugrue, Henry Kwok, Mikel Jimenez Fernandez, Petr Lapukhov, et al. 2023. Ebb: Reliable and evolvable express backbone network in meta. In *Proceedings of the ACM SIGCOMM 2023 Conference*. 346–359.

[13] Martin E. Dyer. 2003. Approximate counting by dynamic programming. In *Proceedings of the 35th Annual ACM Symposium on Theory of Computing, June 9-11, 2003, San Diego, CA, USA*.

[14] eBPF. [n. d.]. eBPF - Introduction, Tutorials & Community Resources. https://ebpf.io/.

[15] Bernard Fortz, Jennifer Rexford, and Mikkel Thorup. 2002. Traffic engineering with traditional IP routing protocols. *IEEE communications Magazine* 40, 10 (2002), 118–124.

[16] Bernard Fortz and Mikkel Thorup. 2000. Internet traffic engineering by optimizing OSPF weights. In *Proceedings IEEE INFOCOM 2000. conference on computer communications. Nineteenth annual joint conference of the IEEE computer and communications societies (Cat. No. 00CH37064)*, Vol. 2. IEEE, 519–528.

[17] Ehab Ghabashneh, Yimeng Zhao, Cristian Lumezanu, Neil Spring, Srikanth Sundaresan, and Sanjay Rao. 2022. A microscopic view of bursts, buffer contention, and loss in data centers. In *Proceedings of the 22nd ACM Internet Measurement Conference*. 567–580.

[18] Jörn-Thorben Hinz, Vamsi Addanki, Csaba Györgyi, Theo Jepsen, and Stefan Schmid. 2023. TCP's Third Eye: Leveraging eBPF for Telemetry-Powered Congestion Control. In *Proceedings of the 1st Workshop on eBPF and Kernel Extensions*. 1–7.

[19] Chi-Yao Hong, Srikanth Kandula, Ratul Mahajan, Ming Zhang, Vijay Gill, Mohan Nanduri, and Roger Wattenhofer. 2013. Achieving high utilization with software-driven WAN. In *ACM SIGCOMM 2013 Conference, SIGCOMM 2013, Hong Kong, August 12-16, 2013*. ACM, 15–26. https://doi.org/10.1145/2486001.2486012

[20] Chi-Yao Hong, Subhasree Mandal, Mohammad Al-Fares, Min Zhu, Richard Alimi, Chandan Bhagat, Sourabh Jain, Jay Kaimal, Shiyu Liang, Kirill Mendelev, et al. 2018. B4 and after: managing hierarchy, partitioning, and asymmetry for availability and scale in google's software-defined WAN. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*. 74–87.

[21] Ellis Horowitz and Sartaj Sahni. 1974. Computing partitions with applications to the knapsack problem. *Journal of the ACM (JACM)* 21, 2 (1974), 277–292.

[22] Ellis Horowitz and Sartaj Sahni. 1974. Sahni, S.: Computing partitions with applications to the knapsack problem. Journal of the ACM 21, 277-292. *Journal of the Acm* 21, 2 (1974), 277–292.

[23] Oscar H Ibarra and Chul E Kim. 1975. Fast approximation algorithms for the knapsack and sum of subset problems. *Journal of the ACM (JACM)* 22, 4 (1975), 463–468.

[24] Giorgio P Ingargiola and James F Korsh. 1973. Reduction algorithm for zero-one single knapsack problems. *Management science* 20, 4-part-i (1973), 460–463.

[25] Sushant Jain, Alok Kumar, Subhasree Mandal, Joon Ong, Leon Poutievski, Arjun Singh, Subbaiah Venkata, Jim Wanderer, Junlan Zhou, Min Zhu, et al. 2013. B4: Experience with a globally-deployed software defined WAN. *ACM SIGCOMM Computer Communication Review* 43, 4 (2013), 3–14.

[26] Srikanth Kandula, Ishai Menache, Roy Schwartz, and Spandana Raj Babbula. 2014. Calendaring for wide area networks. In *ACM SIGCOMM 2014 Conference, SIGCOMM'14, Chicago, IL, USA, August 17-22, 2014*. ACM, 515–526. https://doi.org/10.1145/2619239.2626336

[27] Konstantinos Koiliaris and Chao Xu. 2017. A faster pseudopolynomial time algorithm for subset sum. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms* (Barcelona, Spain) *(SODA '17)*. Society for Industrial and Applied Mathematics, USA, 1062–1072.

[28] Umesh Krishnaswamy, Rachee Singh, Nikolaj Bjørner, and Himanshu Raj. 2022. Decentralized cloud wide-area network traffic engineering with {BLASTSHIELD}. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*. 325–338.

[29] Praveen Kumar, Yang Yuan, Chris Yu, Nate Foster, Robert Kleinberg, Petr Lapukhov, Chiun Lin Lim, and Robert Soulé. 2018. Semi-oblivious traffic engineering: The road not taken. In *15th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 18)*. 157–170.

[30] Eugene L. Lawler. 1979. Fast Approximation Algorithms for Knapsack Problems. *Mathematics of Operations Research* 4, 4 (1979), 339–356.

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

[32] M. Mahalingam, D. Dutt, K. Duda, P. Agarwal, L. Kreeger, T. Sridhar, M. Bursell, and C. Wright. 2014. Virtual eXtensible Local Area Network (VXLAN): A Framework for Overlaying Virtualized Layer 2 Networks over Layer 3 Networks. (2014).

[33] Linux Programmer's Manual. [n. d.]. tc-bpf(8). https://man7.org/linux/man-pages/man8/tc-bpf.8.html

[34] Steven McCanne and Van Jacobson. 1993. The BSD Packet Filter: A New Architecture for User-level Packet Capture.. In *USENIX winter*, Vol. 46. 259–270.

[35] Pooria Namyar, Behnaz Arzani, Srikanth Kandula, Santiago Segarra, Daniel Crankshaw, Umesh Krishnaswamy, Ramesh Govindan, and Himanshu Raj. 2024. Solving Max-Min Fair Resource Allocations Quickly on Large Graphs. In *21st USENIX Symposium on Networked Systems Design and Implementation, NSDI 2024, Santa Clara, CA, April 15-17, 2024*. USENIX Association. https://www.usenix.org/conference/nsdi24/presentation/namyar-solving

[36] Deepak Narayanan, Fiodar Kazhamiaka, Firas Abuzaid, Peter Kraft, Akshay Agrawal, Srikanth Kandula, Stephen Boyd, and Matei Zaharia. 2021. Solving large-scale granular resource allocation problems efficiently with pop. In *Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles*. 521–537.

[37] Yarin Perry, Felipe Vieira Frujeri, Chaim Hoch, Srikanth Kandula, Ishai Menache, Michael Schapira, and Aviv Tamar. 2023. DOTE: Rethinking (Predictive) WAN Traffic Engineering. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*. 1557–1581.

[38] Ben Pfaff, Justin Pettit, Keith Amidon, Martin Casado, Teemu Koponen, and Scott Shenker. 2009. Extending networking into the virtualization layer.. In *Hotnets*.

[39] Jon Postel. 1981. *Internet protocol*. Technical Report.

[40] Sanjay Rao, Mohit Tawarmalani, et al. 2021. FloMore: Meeting bandwidth requirements of flows. *arXiv preprint arXiv:2108.03221* (2021).

[41] Sartaj Sahni. 1975. Approximate algorithms for the 0/1 knapsack problem. *Journal of the ACM (JACM)* 22, 1 (1975), 115–124.

[42] Martin Suchara, Dahai Xu, Robert Doverspike, David Johnson, and Jennifer Rexford. 2011. Network architecture for joint failure recovery and traffic engineering. *ACM SIGMETRICS Performance Evaluation Review* 39, 1 (2011), 97–108.

[43] Lars Wüstrich, Markus Schacherbauer, Markus Budeus, Dominik Freiherr von Künßberg, Sebastian Gallenmüller, Marc-Oliver Pahl, and Georg Carle. 2023. Network Profiles for Detecting Application-Characteristic Behavior Using Linux eBPF. In *Proceedings of the 1st Workshop on eBPF and Kernel Extensions*. 8–14.

[44] Zhiying Xu, Francis Y Yan, Rachee Singh, Justin T Chiu, Alexander M Rush, and Minlan Yu. 2023. Teal: Learning-Accelerated Optimization of WAN Traffic Engineering. In *Proceedings of the ACM SIGCOMM 2023 Conference*. 378–393.

[45] Rui Yang and Marios Kogias. 2023. HEELS: A Host-Enabled eBPF-Based Load Balancing Scheme. In *Proceedings of the 1st Workshop on eBPF and Kernel Extensions*. 77–83.

[46] Zhizhen Zhong, Manya Ghobadi, Alaa Khaddaj, Jonathan Leach, Yiting Xia, and Ying Zhang. 2021. ARROW: restoration-aware traffic engineering. In *Proceedings of the 2021 ACM SIGCOMM 2021 Conference*. 560–579.

Appendices are supporting material that has not been peer-reviewed.

A TE Formulation

A.1 Proof of NP-Hardness

Theorem A.1. `MaxAllFlow` *is an NP-Hard problem.*

Proof. 0-1 knapsack is a well-known combinatorial optimization problem which is NP-Hard [21, 24, 41]. We prove `MaxAllFlow` problem is NP-Hard by showing that 0-1 knapsack problem is a special case of `MaxAllFlow`. To reduce the 0-1 knapsack problem to an instance of our problem, we create a graph with only one site pair connected by a path, and a set of endpoint pairs I associated with the site pair. We then assume the bandwidth allocation C (backpack capacity) between the site pair and corresponding endpoint pairs, and traffic demands $d^i, \forall i \in I$ (items). This creates a problem of how to assign items (traffic demands between endpoint pairs) into the backpack (bandwidth allocation) to maximize the total value of the items transferred into the backpack. As a result, this is an instance of a 0-1 knapsack problem. Since the 0-1 knapsack problem is NP-hard, the `MaxAllFlow` problem is also NP-hard. □

A.2 Topology Contraction

As described in Section 4.2, we separate the graph into two layers.

First Layer. For the first layer, we aggregate the endpoint pairs connecting to the same network site pair, i.e., $\forall k \in K, F_k = \sum_t F_{k,t}$ where F_k represents the traffic allocation between network site pair k and $F_{k,t}$ specifies the bandwidth allocation on the tunnel t. Similarly, we also aggregate the network site pair demands, i.e., $\forall k \in K, D_k = \sum_i d_k^i$ where D_k represents the demand between network site pair k. It follows that the TE optimization problem for the first layer (`MaxSiteFlow`) is transformed to the following:

$$\arg\max_F \quad \sum_{k,t} F_{k,t} - \epsilon \sum_{k,t} w_t F_{k,t} \tag{2}$$

$$s.t. \quad \sum_t F_{k,t} \le D_k, \qquad \forall k \tag{2a}$$

$$\sum_{k,t} F_{k,t} L(t,e) \le c_e, \quad \forall e \tag{2b}$$

$$F_{k,t} \ge 0, \qquad \forall k, t \tag{2c}$$

The objective function (2) is to maximize the overall throughput while preferring shorter paths. ϵ is a small constant. Constraint (2a) states that the allocated bandwidth among all paths should be less than the traffic demand. Constraint (2b) states that no link should be overloaded. Constraint (2c) states the allocated bandwidth should be non-negative. The first layer alone, i.e., Equation 2, aligns with the optimization objectives found in existing TE studies [19], where the number of network sites is up to a few thousand. Thus it can be easily solved using existing methods [2, 19].

Second Layer. The TE problem at the second layer can be regarded as the traffic allocation on the path t of site pair k, referred to as `MaxEndpointFlow`$(F_{k,t}, \{d_k^i\})$. We assume that $t \in T_k$ is ordered by ascending w_t, meaning that paths with lower latency and higher reliability are assigned lower subscripts t. It is noteworthy that `MaxEndpointFlow`$(F_{k,t}, \{d_k^i\})$ must be tackled sequentially, with solutions for higher values (e.g., shorter paths) of t building upon the outcomes of lower ones. For instance, the resolution of `MaxEndpointFlow`$(F_{k,2}, \{d_k^i\})$ is contingent upon the results obtained from `MaxEndpointFlow`$(F_{k,1}, \{d_k^i\})$.

`MaxEndpointFlow`$(F_{k,t}, \{d_k^i\})$: The `MaxEndpointFlow` problem is a subset sum problem (SSP), a special scenario of the Knapsack problem which is classified as an NP-hard problem. This characteristic raises the challenge of solving the `MaxEndpointFlow` problem in a timely manner. Various pseudopolynomial algorithms have been devised to tackle this issue, with dynamic programming (DP) being the traditional method of choice. However, the DP approach is less than ideal due to its high complexity when dealing with small values of $\{d_k^i\}$ against the large values of $|I_k|$ and $F_{k,t}$. To mitigate this, we propose a novel semi-DP technique that significantly reduces complexity while allowing for controllable precision in the solution. We refer to this algorithm as `FastSSP`$(F_{k,t}, \{d_k^i\})$, which is an approximation of optimal solution of `MaxEndpointFlow`$(F_{k,t}, \{d_k^i\})$.

`FastSSP` works as the following procedure:

(1) **Clustering**. As the endpoint flow d_k^i is small, we aggregate these pair demands to meet or exceed a threshold M. Here, we set M to $\frac{1}{3}\epsilon' F_{k,t}$ where ϵ' is close to 0. The original endpoint pair demands can be classified into m sizeable demand clusters. Here, m is a small integer, determined by M. Let $\{c_k^m\}$ represent the aggregated demands, where $\forall x \in [1, m], c_k^m \ge M$ and $|\{c_k^x\}| = m$.

(2) **Normalization**. As the time complexity of DP is closely related to the value of $F_{k,t}$ and aggregated demand number m, we reduce the time complexity of dynamic programming (DP) with a factor δ. Specifically, the aggregated demand c_k^m is normalized to $\hat{c}_k^m = \lceil \frac{c_k^m}{\delta} \rceil$ and bandwidth allocation $F_{k,t}$ is normalized to $\hat{F}_{k,t} = \lfloor \frac{F_{k,t}}{\delta} \rfloor$. Here, similar to [23], we set δ to $\frac{\epsilon'}{3}M$ (i.e., $\frac{\epsilon'^2}{9}F_{k,t}$). Therefore, the time complexity of DP reduces from $O(mF_{k,t})$ to $O(m\lfloor \frac{F_{k,t}}{\delta} \rfloor)$.

(3) **Solving DP**. After normalization, we adopt the classic DP approach [6] to solve the SSP and derive the selected normalized demand set $\{s_k^j\} \subseteq \{\hat{c}_k^m\}$ with maximizing the sum of the subset $\sum_j s_k^j$ subject to the constraint $\hat{F}_{k,t}$. Note that we can obtain the optimal solution by solving the DP without introducing much time complexity since both m and $\hat{F}_{k,t}$ are small. We then map the selected normalized demand set $\{s_k^j\}$ into its original demand set, represented as $\{c_k^{*i}\}$. According to the $\{c_k^{*i}\}$, we obtain the selected pair demands $\{d_k^{*i}\}$.

(4) **Sorted-based greedy algorithm**. After solving DP and obtaining the selected demand set, there will be some residual bandwidth $R_{k,t} = F_{k,t} - \sum_i c_k^{*i}$ with the residual demand set $\{r_{k,t}\} = \{x | x \in d_k^i \, and \, x \notin d_k^{*i}\}$. The allocation for the residual flows is another SSP problem. Because the residual flows have a relatively small demand, they will not pose a large error to the final outcome. We thus adopt a sorted-based greedy algorithm to solve the residual flow allocation, where the solution is $\{r_k^{*i}\} \subset \{r_k^i\}$. Finally, we obtain the rest set without bandwidth allocation $\{r_{k,t}\} = \{x | x \in d_k^i \, and \, x \notin d_k^{*i} \, and \, x \notin r_k^{*i}\}$. The error rate is $\beta \le min\{r_{k,t}\}/F_{k,t}$.

The total time complexity of `FastSSP` is $O(m\lfloor \frac{F_{k,t}}{\delta} \rfloor) + |I_k| log |I_k|)$