EdgeCross: Cloud Scale Traffic Management at Peering Edges

XIAOLIANG WANG, State Key Laboratory for Novel Software Technology, Nanjing University, China
PENGHUI MI, YONG ZHU, BAOYI AN, YINHUA WANG, LIXIANG WANG, XUEZHI YU, QIONG XIE, XIANG HUANG, MINGLIANG YIN, CHAOYANG JI, WEI SUN, YIHANG LV, Huawei Cloud Computing Technologies Co., Ltd., China
YUHANG CHEN, CAM-TU NGUYEN, CHEN TIAN, State Key Laboratory for Novel Software Technology, Nanjing University, China
XIAOMING FU, Institute of Computer Science, University of Göttingen, Germany

Cloud providers deployed dozens of PoPs and data centers globally to serve billions of geo-distributed users. The traffic management at peering edges has become a key capability of cloud network operators to meet the diverse demands of users. With the rapid growth of cloud applications, users have recently announced new performance requirements, e.g., achieving latency as low as possible instead of maintaining a specified delay. The conventional inter-domain bandwidth allocation approach, which aims to reduce the high operating expenditures of bandwidth usage, fails to meet these new requirements. We further reveal that the flow scheduling among PoPs may fail due to the limited link capacity hidden by the cloud private backbone network controller. Therefore we seek a new traffic management at peering edges.

We propose a new controller framework, EdgeCross, that satisfies not only users' emerging demands but maintains low operating costs. The large number of fine-grain application-aware flows and the consideration of backbone links' capacity lead to very high complexity of routing computation and verification for the controller. EdgeCross introduces a two-phase operation that first achieves the low-expense bandwidth allocation according to the standard 95^{th} percentile billing model and then allocates specified flows to peering edges based on users' requirements. EdgeCross further reduces large memory consumption by proposing an effective routing table compression approach. The evaluation based on a production network with 16 PoPs has shown that EdgeCross can successfully process the routes of 1 billion flows in 10 seconds, reduce the average delay for performance-sensitive flows by 2 milliseconds compared to traditional BGP, and is able to save the bandwidth cost by 10-26% compared to the state-of-the-art Cascara[25].

CCS Concepts: • **Networks** → **Network management**.

Additional Key Words and Phrases: Cloud Network, Peering Edges, SDN Controller, Flow Scheduling

ACM Reference Format:
Xiaoliang Wang, Penghui Mi, Yong Zhu, Baoyi An, Yinhua Wang, Lixiang Wang, Xuezhi Yu, Qiong Xie, Xiang Huang, Mingliang Yin, Chaoyang Ji, Wei Sun,, Yihang Lv, Yuhang Chen, Cam-Tu Nguyen, Chen Tian, and Xiaoming Fu. 2024. EdgeCross: Cloud Scale Traffic Management at Peering Edges. *Proc. ACM Netw.* 2, CoNEXT4, Article 24 (December 2024), 23 pages. https://doi.org/10.1145/3696396

1 Introduction

The public clouds provide high-speed, reliable, low-latency networking services for geo-distributed users. The cloud wide-area networks (WANs) play an important role in connecting data centers and users through globally deployed Points of Presence (PoP). These PoPs are interconnected by the cloud private backbone network. Through the SDN-based centralized controller [15, 16], cloud

ACM 2834-5509/2024/12-ART24
https://doi.org/10.1145/3696396

network operators can establish optimal routes, which meet users' demands and minimize operating expenditure of bandwidth costs[3, 23, 32]. Prior efforts mainly focus on reducing the inter-domain bandwidth cost by leveraging multiple latency-equivalent peering links of PoPs [25, 33].

Recently, with the rapid growth of diverse cloud services, like metaverse-gaming, quantitative finance, and short videos, the cloud provider receives various and stringent requirements on networking quality. For example, finance institutions demand ultra-low latency as much as possible. The prior cost-oriented bandwidth allocation approach can not satisfy the requirements of users. The operators have to manually select the peering link with the lowest latency from remote PoPs, which potentially increases the operating cost of cloud networks. Therefore we need a new unified model to re-define the objective functions for both users and cloud providers.

Our operation experience further reveals another shortcoming of the current inter-domain bandwidth allocation approach. It does not consider the capacity constraints of links in the cloud private backbone network, which connects PoPs in different places. When scheduling network flows to the remote low latency PoP egress, the operation may fail and can not satisfy the performance requirement of users in practice due to congestion in the backbone network. Therefore, we seek to improve the operation effectiveness by taking into account the link capacity of the backbone network when scheduling the egress traffic to peering edges.

In this paper, we propose EdgeCross, a software-defined centralized controller for inter-domain traffic routing at peering edges. EdgeCross first formulates traffic management as a multi-objective optimization to balance the bandwidth costs and the specified requirements of users. The main challenges are the high computation complexity caused by the fine-grained application-aware flows, the capacity constraints of links in the backbone network, and the storage complexity of the large cloud WAN routing tables in the controller's memory. When formulating the optimization problem, the variables and constraints have reached as high as $O(10^{10})$ and $O(10^{8})$ respectively, which can not be solved by the commodity optimization solvers [11].

To complete the route decision in each 5-minute execution interval in the product network, we propose a two-phase acceleration approach. It consists of a coarse-grained offline optimization for the pre-allocation of bandwidth usage at each peering edge and a fine-grained online optimization to schedule critical flows to the peering links. A parallel implementation is introduced to further accelerate the routing computation. EdgeCross realizes routing table compression through a hierarchical forest-based index structure, which allows storage and fast routing verification inside controllers. In summary, the contributions of this paper are:

- EdgeCross explains a key problem of traffic management at peering edges in modern cloud WAN networks. As far as we know, EdgeCross is the first work explaining the new requirements of cloud customers, such as low latency as possible, and as low cost as possible, instead of deterministic latency, and bandwidth. We further reveal the design and implementation challenges to meet the user demands, while reducing the operating expenditure of cloud providers.
- EdgeCross overcomes the challenge of high computation complexity for scheduling application-aware flows. We introduce a series of effective optimization approaches to accelerate the route computation, including joint-optimization of both peering edges and backbone networks, and the routing table compression at the control plane to verify the generated routes.
- The effectiveness of EdgeCross has been verified through the product network traffic. EdgeCross is able to process over 1 billion flows within 10 seconds, reduces the average delay for performance-sensitive flows by 2 milliseconds compared to traditional BGP, and saves the bandwidth cost by 10-26% compared to the state-of-the-art Cascara[25].

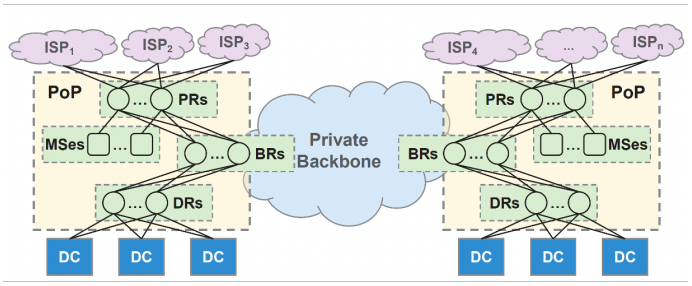


Fig. 1. DCs are connected to BRs and PRs through Data center Routers (DRs). Outbound traffic can be routed to peering link at any PoP.

2 EdgeCross Overview

2.1 Background

We briefly introduce the architecture of the cloud Wide-Area Networks (WANs). To serve geo-distributed users, the cloud providers deployed multiple Point-of-Presences (PoPs) and Data Centers (DCs) globally. As shown in Figure 1, each PoP hosts multiple Peering Routers (PRs) to connect with the Internet Service Providers (ISPs) through **peering links**. Each ISP also serves as a transit provider. We focus on the egress traffic management at Peering Routers, i.e. scheduling traffic from DCs to the Internet through carefully selected peering links. Notice that, through Backbone Routers (BRs), PoPs in different places are interconnected by the cloud private backbone network, the outbound traffic from DCs can go to the Internet through either directly connected PoPs or remote PoPs across the private backbone. Each peering router is equipped with a monitoring server (MS) to monitor the link quality to the ISP [30].

The operation experience is billed by the bandwidth usage of peering links. The widely used billing model is **the** 95^{th} **percentile billing model** which has evolved as an industry standard when cloud providers sign contracts with ISPs [15, 25]. Over the monthly billing cycle, bandwidth usage in the peering link is recorded for every time slot (usually 5 mins). By sorting the utilization of megabits in every time slot, the billing model charges the 95^{th} percentile of the link usage. The top 5% of **bursty bandwidth usage** over the billing period will not be counted. Notice that in the contract each peering link is assigned a committed bandwidth to guarantee the benefit of ISPs. The cloud provider is obligated to pay the committed bandwidth even if the total used traffic is less than the committed bandwidth value. Since outbound traffic is significantly higher than inbound traffic, the cloud operators, e.g. Microsoft [25], focus on minimizing the total outbound bandwidth cost at peering links. The following trends lead to new challenges for modern cloud network operators.

2.2 Motivation

Differentiated Requirements of Cloud Users. With the rapid growth of newly emerging cloud applications, users have differentiated requirements for networking services. We take the applications in Table 1 as examples. For gaming and finance, rather than low prices, users prefer the Premium Services to achieve latency as low as possible. On the contrary, for CDN and storage, users mainly apply for the routing path with low price by using the Cost-centric Service. For the short videos deployed in multiple clouds of a specified region, users distribute the workloads to different cloud networks based on the latency measured in real time.

Congestion of Workflows in Cloud Private Backbone. Traffic scheduling focuses on balancing workloads among PoPs in different domains to save cost, and enhance the availability and performance of applications workloads. Generally, traffic scheduling for the Internet and traffic

Service	Scenario	Demands
Premium	Game, Finance	Global Performance-aware
Latency-centric	Short Video	Specific Region Performance-aware
Bandwidth-centric	IT business	Global Access, Bandwidth-aware
Cost-centric	CDN, Storage Service	Low Cost

Table 1. Recommended choices of services

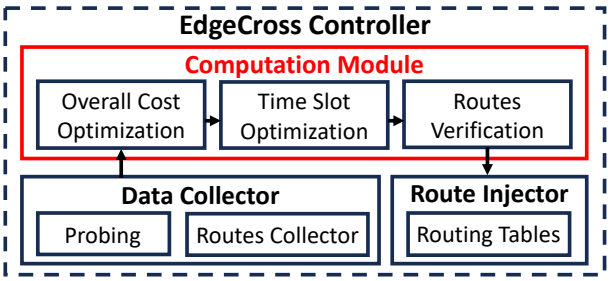


Fig. 2. The EdgeCross controller.

engineering (TE) for the private backbone is operated independently through different controllers [19]. Therefore the cloud WAN controller usually takes the assumption that the backbone network has sufficient bandwidth resources for routing all traffic. However, since the virtual paths of different pairs of PoPs may share the same physical backbone links, the assigned workload can exceed the capacity of a certain backbone link, and lead to congestion, unexpected delay, and even packet loss. We verify the problem by using the state-of-the-art inter-domain bandwidth allocation algorithm, Cascara [25], which assumes that the link capacity of the backbone is not limited. We conduct experiments by using the traffic data in our product networks without considering the constraints of backbone link capacity (see §6). Surprisingly, we observed that ~10-30% of the backbone links are overloaded. And in the worst case, the maximum link usage can be as high as 800%.

2.3 EdgeCross Controller

To address the above issues, we introduce the EdgeCross controller which revisits the outbound traffic scheduling approach at the peering edge to satisfy the new demand of users and save the bandwidth costs simultaneously. Similar to the sample given in Table 1, we provide the user interface for customers to explicitly demonstrate their requirements. The core of EdgeCross is shown in Figure 2, EdgeCross controller is designed to optimize and control traffic for cloud peering edges, which consists of a data collector, a route computation module, and a route injector. The data collector gathers route announcements from each PR device, the status of backbone links and peering links. These collected data are forwarded to the computation module, which computes and verifies the routes to be advertised.

We employ the triplet <PoP, service class, destination prefix> to identify flows. Here, the *service class* is aggregated to the source virtual machine's Elastic IP (EIP) address, serving to differentiate the performance metrics of customers. Finally, the route injector installs the routes into the routing tables of PRs in the network [1]. In this paper, we focus on the computation module and need to overcome the following key challenges:

Traffic Scheduling with Multi-objective. We need to carefully define the problems to simultaneously meet the differentiated requirements of users and applications while maintaining low operating costs based on the 95^{th} percentile billing model (§3).

[1]The data plane operation of overriding BGP's path selection and identifying the application-aware traffic is out of the scope of the paper, please refer to the operational experience works [23, 30, 32].

High Computation Complexity. Concerning the application-aware fine-grain traffic flow identification as well as the link capacity associated with both peer routers and private backbone, the variables and constraints have reached as high as $O(10^{10})$ and $O(10^{8})$ respectively in the formalized problem. We need an effective scheduling approach to solve the multi-objective problem with high computation complexity. The objective is to compute 1 billion routes within 10 seconds (§4.2).
Large Routing Table Size. The controller needs to collect all routing tables of PRs to verify the generated routes before installing new routes to PRs. Each PR has a routing table of ~8 million routes. Given around 200 PRs in our WAN, the size of the complete routing tables can be as large as 1600GB. It is hard to store the large table in the memory of controllers. We need an efficient routing table compression approach (§5).

3 Traffic Management at EdgeCross

We define the optimization problem (§3.1) and propose a two-phase scheme to solve the problem (§3.2), the details of which are introduced (§3.3 and §3.4) respectively.

3.1 Objective

We explain the goal of cloud peering edge traffic engineering. EdgeCross has two objectives concerning both operating cost and performance-sensitive traffic.
Ojective 1: Minimizing the total 95^{th} percentile bandwidth cost. Our allocation scheme aims to determine a traffic assignment to peering links over the entire billing period while minimizing the cumulative egress bandwidth cost and considering the capacity constraints of backbone links and committed bandwidth of peering links.

Objective 2: Minimizing the sum of weighted performance factors in each time slot. The performance factors encompass variables such as latency, packet loss, and delay jitter, which affect the quality of users' experiences when accessing the cloud. We compute the weighted sum of these performance factors by aggregating the factors associated with all performance-sensitive flows.

3.2 Problem Decomposition

Previous solutions on multi-objective optimization problems [5, 13, 18] identify a single decision vector to balance the trade-offs between many objectives. An optimal trade-off to achieve Pareto Optimal. We notice that the above two optimization objectives are in different time granularity. The first objective takes the overall bandwidth variance of the cloud as input and generates the minimum 95^{th} percentile billable bandwidth for each peering link. Therefore, solving the 95^{th} percentile charging problem requires considering the entire charging period of 1 month, which is not sensitive to the specific flow scheduling in each 5-minute slot. The second objective, minimizing the weighted sum of performance factors, focuses solely on assigning each performance-sensitive flow within each time slot. Therefore, we can decompose the optimization problem based on the granularity of the optimization period and adopt a two-phase model for the flow scheduling:

Phase 1: Estimation of 95^{th} Percentile Billable Bandwidth (§3.3). In the first phase, we focus on accurately estimating the optimal 95^{th} percentile billable bandwidth for each peering link based on the historical traffic. **The estimation is triggered periodically, e.g. each month.** However, when the traffic pattern is significantly changed, such as network device failures, we need to recompute the billable bandwidth. In our product network, the execution interval is about 10 hours.

Phase 2: Flow Scheduling (§3.4). The second phase takes the estimated 95^{th} percentile billable bandwidth given in the first phase as input, and schedules flow to PoPs at each time slot. It dynamically deals with performance-sensitive flows. **The flow scheduling is executed online**

Symbol	Meaning
E	Set of backbone links
V	Set of PoPs
T	Set of time slots
L	Set of peering links
F	Set of flows
F_S	Set of performance-sensitive flows
m	Number of peering links
n	Number of five-minute time slots in a month
l_i	Peering link i
C_i	Capacity of peering link l_i
$C_{(u,v)}$	Capacity of backbone link from PoP u to PoP v
L_u	Set of peering link of PoP u
ub_i	Minimal billable bandwidth of peering link l_i
E_u^S	Set of outgoing backbone links from PoP u
E_u^D	Set of incoming backbone links to PoP u
c_i	Peering rate (USD/Gbps) for peering link l_i
$perf(i, f)$	Performance function of flow f and peering link l_i

Table 2. Symbols used in both overall and the time slot optimization

every 5 minutes, and thus the execution time of the optimization is generally confined to 10 seconds with regard to the time consumption used to collect data.

3.3 Estimation of 95^{th} Percentile Billable Bandwidth

We first formalize the task of estimating the 95^{th} percentile billable bandwidth. Different from the previous cost-optimal allocation approach in Cascara[25], EdgeCross further considers the impact of the limitation of the backbone links. The notation given in Table 2 comprises symbols related to the backbone and peering edges. The formulation detail is introduced in Appendix A.1.

Objective Function. The objective is to minimize the total outbound bandwidth cost incurred across all peering links, i.e. min $\sum_{l_i \in L} c_i p_i$, where p_i is the 95^{th} percentile billable bandwidth of peering link l_i.

Input Parameters. In addition to the common input parameters, we predict the traffic demand $\{b_u^1, b_u^2, ..., b_u^i, ..., b_u^n\}$ at each slot i for all peering links associated with PoP $u \in V$, which are achieved based on the traffic of the last three months as well as the same month of last year.

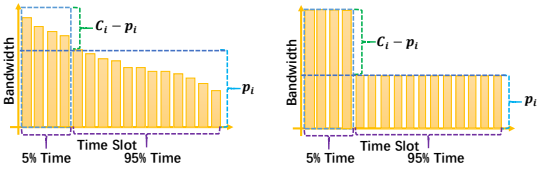
Output Parameters. The predicted billing bandwidth p_i for each peering link l_i.

Key Insights. We draw insights from the status quo of the cloud provider's network, which motivates the estimation of the 95^{th} percentile billable bandwidth:

- **Be aware of the impact of the backbone network.** Notice that efficient utilization of the backbone network can avoid congestion and failed scheduling. The 95^{th} bandwidth estimation in the overall optimization should take into account the constraints of backbone link capacity.
- **Minimizing 95^{th} percentile bandwidth cost.** The problem of 95^{th} percentile bandwidth allocation is not new [15, 25]. To ease understanding, a toy example is shown in Figure 3. Given the traffic flows of peering link l_i, cloud providers carefully schedule traffic to the 5% free time slots, and evenly distribute the traffic to the remaining peering link, to achieve a low billable bandwidth cost. This operation is called **bursting of the peering link**. The scheduler can then

determine p_i, the 95^{th} percentile billable bandwidth of peering link l_i. It guarantees that no cost increases if the bandwidth usage of peering link l_i follows the distribution given in Figure 3(b). However, in practice, the operation of bursting the peering link can change the traffic pattern and lead to network congestion. To mitigate the impact of this problem, EdgeCross employs an incremental bursting strategy by gradually increasing the traffic allocation at the peering links to ensure a more stable traffic growth there.

Complexity Analysis. Regarding the capacity constraints of the egress ports and backbone networks for all time slots in a month, both the number of variables and constraints is $O(n(|E|+|L|+|V|))$, which is approximately 15 million variables and 30 million constraints in our network topology for $n = 8640$. We can not solve the problem in time with a commodity optimization solver like Google OR-Tools [11], which takes more than one month. We provide the corresponding solution in §4.1.



(a) Before optimization (b) After optimization

Fig. 3. Bandwidth usage distribution of peering link l_i.

3.4 Flow Scheduling per Time Slot

The task of flow scheduling is performed in each slot. i.e. assigning flows to the peering links based on the 95^{th} percentile billable bandwidth given in Phase 1. The problem is a Generalized Assignment Problem (GAP) [24], which is NP-hard, and formulated in Appendix A.2.

Objective Function. The goal of our time slot optimization scheme is to find a flow assignment to edge links in the current time slot such that the current weighted sum of the performance factor of all flows and residual capacities of peering links are minimized. i.e.

$$\min \frac{\beta}{|F_s|} \sum_{f \in F_s} \sum_{l_i \in L} w_1(f) x_i^f perf(i,f) + (1-\beta) \frac{\sum_{l_i \in L} y_i}{\sum_{l_i \in L} C_i}$$

Each performance-sensitive flow $f \in F_S$ is associated with a function $perf(i,f)$, which indicates the achieved performance at peering link l_i based on the service class in the current time slot. For example, $perf(i,f)$ can be a discretized weighted functions of latency and packet loss, consisting of both a latency function and a packet loss function. The latency term assigns a value based on pre-determined thresholds: if the latency exceeds $100ms$, the latency term is set to 1; if it exceeds $50ms$ but is below $100ms$, the value is 0.8, and so on. The packet loss term is similarly defined, where different packet loss rates correspond to specific weight values. Thus, lower values of $perf(i,f)$ correspond to better performance. The binary variable x_i^f indicates whether flow f is scheduled to peering link l_i, where $x_i^f = 1$ if flow f is routed through l_i, and $x_i^f = 0$ otherwise. The weight $w_1(f)$ identifies the priority of flow f. The first function in the formulation defines the aggregate performance across all applications. β is a hyperparameter used to adjust the trade-off between cost and performance. Furthermore, we incorporate the selection of bursting egresses. The variable y_i represents the available capacity at egress l_i. If selecting as the slot of 5% bursting peering link, y_i is set to the maximum capacity C_i; otherwise, y_i is set to the estimated 95^{th} percentile bandwidth p_i.

Constraints. The constraints introduced in our formalization pertain to peering links, backbone links, and PoPs. The bandwidth allocation of a PoP depends on its outgoing, incoming, and total

traffic. The assignment of a flow during the current time slot depends on the available capacity of peering links and backbone links.

Input Parameters. In addition to the common input parameters of WAN and edge of the cloud. The data input is all related to the current time slot, including b_u and a set of flows F.

Key Insights. In the process of time slot optimization, we conduct a comprehensive analysis of prior research findings. Based on this analysis, we summarize several valuable insights:

- The billable bandwidth ensures the predicted traffic can be forwarded to the peering link with a low operating expenditure. However, the 95^{th} billable bandwidth is estimated offline based on the predicted traffic. During the online flow scheduling, the real traffic usually varies from the predicted traffic. Therefore, we need to solve two problems at each time slot: 1) Determining which peering link we can fully utilize its bandwidth, i.e. bursting of the peering link; 2) Maintaining the bandwidth usage of other links close to the billable bandwidth p_i as much as possible, which can avoid high unexpected bandwidth cost.
- We need to avoid the waste of peering links bandwidth resource. For example, if the peering links of remote PoPs have sufficient resources, we need to avoid the backbone network becoming the bottleneck and affecting the scheduling decision. Thus we still need to consider the limited capacity of backbone links in real-time during the flow scheduling at each time slot.

These insights further motivate us to decompose the time slot optimization problem into two subproblems: selecting bursting peering links and assigning traffic flows, the detail of which is introduced in §4.2.2. Ultimately, we achieve a ratio of outbound bandwidth cost optimization of approximately 32%, which reveals that deliberating the selection of bursting peering links is much more critical than the estimation of billable bandwidth.

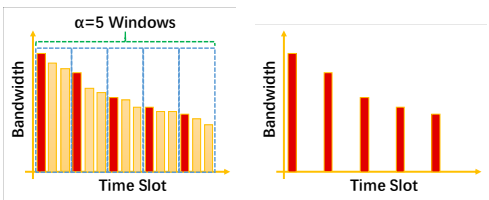
Complexity Analysis. The total number of constraints amounts to $O(|F|(|E| + m))$, where $|F|$ represents the number of flows. In our testing topology, there are 2 million flows per time slot, i.e., the time slot algorithm is projected to involve approximately 200 million constraints and 2 billion decision variables. The current optimization solver can schedule 10,000 flows in ~1 minute. Hence, it is unable to schedule 200 million flows per time slot. We seek for acceleration approach in §4.2.

4 Optimization

4.1 Accelerating the Estimation of 95^{th} Percentile Billable Bandwidth

In order to estimate the 95^{th} percentile billable bandwidth in a few hours (10 hours in our network), we introduce an approximation approach to estimate the bandwidth demand but significantly reduce the computation complexity. It is reasonable because the traffic workload varies in practice and we can not achieve the precise bandwidth demand through prediction. We can mitigate the impact of the approximation approach through the realistic flow scheduling in each time slot (§4.2).

Minimizing the number of time slots to be considered while retaining the characteristics of the bandwidth distribution. Notice that the number of variables and constraints are both closely related to the number of time slots, we aim to reduce the number of time slots to be considered through a sampling scheme. We design an optimization scheme through the idea of sorting and



(a) Before sampling (b) After sampling

Fig. 4. A toy example of sampling when $\alpha = 5$.

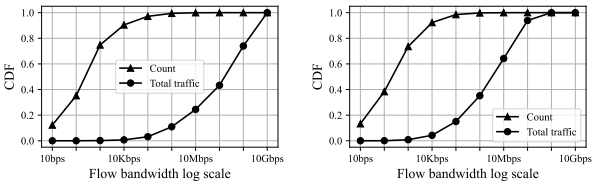
sampling that greatly reduces the problem-solving time while ensuring a certain degree of optimality. For each peering link, we first sort the total predicted bandwidth demands of the whole time slots in a month. Figure 4 shows a toy example of the sampling approach. We equally divide all time slots into α parts, each of which is referred to as a window. Within each window, we retain the time slot with the maximum bandwidth demand and remove the others. We select the maximum one to ensure that any other demand in the same window can be scheduled to the peering link without violating the link capacity constraint in practice. Since we sort the bandwidth demands of all slots, given a carefully selected window it ensures that the difference in bandwidth requirements is small enough. Finally, we take all sampled bandwidth demands into consideration which can represent the distribution of bandwidth demands in a PoP.

As long as we choose the proper size α, we can maintain the accuracy while gaining a huge acceleration. α is determined by the traffic in practice. Generally, given $n = 8640$ slots in a month, we chose $\alpha = 288$ and successfully reduced the number of variables and constraints by 96.6%. Since there appears to be a tradeoff between time and accuracy of the optimization problem, we can solve the estimation of 95^{th} percentile billable bandwidth problem within 20 minutes, but with an error of less than 5% compared to the optimal solution.

4.2 Accelerating Flow Scheduling

To ensure the flow scheduling algorithm execution time is under 10 seconds, we analyze the traffic characteristics and introduce feasible solutions.

4.2.1 Minimize the number of scheduled flows. We analyze the distribution of the traffic flows based on the traffic of one week, i.e. from May 22 to May 28, 2023, in the product cloud network. As shown in Figure 5, the flows using 94.21% of the total bandwidth comprise only 0.91% of the overall flow count. In addition, we find that approximately 4.5% of the total flows are performance-sensitive flows, accounting for approximately 0.8% of the total bandwidth of all flows. The cost is mainly determined by the total bandwidth instead of the number of flows. A small number of large flows use the majority of the bandwidth.



(a) All flows (b) Performance-sensitive flows

Fig. 5. Bandwidth distribution of outbound traffic

By observing the distribution of the outbound traffic, we notice that a few large flows dominate the bandwidth usage. We do not need to schedule all the flows in each slot. Instead, we can focus on those large flows for cost-saving and scheduling performance-sensitive flows to satisfy the service requirements. Specifically, this optimization avoids frequent traffic scheduling and mitigates the impact of packet loss and delay jitter. We introduce a hyperparameter FT to identify the threshold. Those flows demanding for bandwidth less than FT are directly forwarded to the default peering links. After setting a proper FT, we can drastically reduce the number of flows that need to be scheduled by 94%.

4.2.2 First allocating bandwidth and then scheduling flows. To reduce the computation complexity of the flow scheduling in each time slot, we decompose the optimization per slot into two steps. Step 1: Allocating the total demanded bandwidth of users in the current time slot to peering link of all PoPs with regard to both the constraint of the backbone link capacity and the estimated 95^{th} percentile billable bandwidth values. In this step, we only consider the total traffic bandwidth demands of each PoP in the current time slot, and allocate bandwidth demands on the backbone

links, and determine whether we can use the full capacity of peering link l_i without considering the specified flow scheduling. The bandwidth allocation problem can be formalized as a MILP. Step 2: Scheduling flows to the peering links based on the bandwidth allocation given by Step 1 while ensuring optimal performance for performance-sensitive flows. The flow scheduling problem can be formulated as a bin packing problem, and we apply a heuristic algorithm to solve this problem.

Bandwidth Allocation. We model the bandwidth allocation problem as a MILP. The model uses the methodology of estimating 95^{th} percentile billable bandwidth algorithm to statistically compute the flow bandwidth within a PoP while considering the goal of minimizing the cost.

Objective function. To minimize the cost, the objective function is set to minimize the sum of unused capacity at each peering link.

Input parameters. In addition to the common inputs listed in Table 2, the algorithm requires only the current time slot and the total traffic in each PoP in the current time slot.

Variables. We set two variables. $B_{(u,v)}$ denotes the usage of the backbone link from PoP u to PoP v in the current time slot. k_i is a binary variable indicating if peering link i needs to burst in the current time slot, i.e. use all available capacity.

The optimization formulation is explained in Appendix A.3. To address the constraint imposed by the limited backbone network capacity, we need to determine the bandwidth that can be scheduled between each pair of PoPs. To this end, we leverage the intermediate results of the above optimization problem: the expected utilization of each backbone link, i.e. the bandwidth should be used for each link of the private backbone network such that we can achieve the optimal objective results. We denote this bandwidth usage of the physical backbone network by "recommended bandwidth". We then allocate the workload between PoPs based on the recommended bandwidth. To this end, we define the virtual link between PoP u and PoP v, $B_{(u,v)}$, the capacity of which represents the sum of the bandwidth of flows to be scheduled. Therefore, the goal of this step is to determine the capacity of the virtual links between PoPs based on the recommended bandwidth usage of the physical backbone network links.

We need to find some virtual links to fully use the recommended bandwidth, i.e. the sum of allocated virtual links capacity is equal to the recommended bandwidth. This problem can be regarded as a specialized maximum flow problem, which can be solved using a greedy algorithm with the time complexity of $O(|V|^2|E|)$ where $|V|$ stands for the number of PoPs and $|E|$ is the number of physical links in the backbone network. Finally, we can obtain the virtual link capacity $B_{(u,v)}$ for each pair of PoP u and PoP v, which means allocating $B_{(u,v)}$ bandwidth for flows from PoP u to PoP v in the scheduling process.

Flow Scheduling. Flow scheduling requires the use of the capacity of the virtual links solved in the previous step. The principle is to make the actual used capacity of the virtual links as much as possible. Due to the small number of performance-sensitive flows and total bandwidth, we can first schedule those performance-sensitive flows to the peering link with the best performance, i.e. finding the peering link l_i that can minimize the $perf$ function defined in §3.4). The scheduling process needs to avoid violating the capacity limit of each peering link and not exceeding the capacity limit of the inter-PoP virtual links. Traffic characterization analysis in §4.2.1 indicates that performance-sensitive flows constitute only a small portion of the total bandwidth. Therefore, almost all performance-sensitive flows can be allocated to the optimal peering link.

For the non-performance-sensitive flows, the scheduling problem can be regarded as a bin-packing problem, assigning flows to fill up the peering link capacity of the virtual links as much as possible. Algorithm 4 in Appendix A.4 presents the pseudo-code of the heuristic scheduling

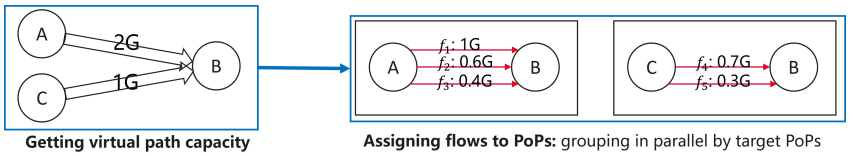


Fig. 6. An example of parallelization. A, B, and C represent distinct PoPs, where the traffic from A to B, and traffic from C to B can be calculated in parallel at A and C.

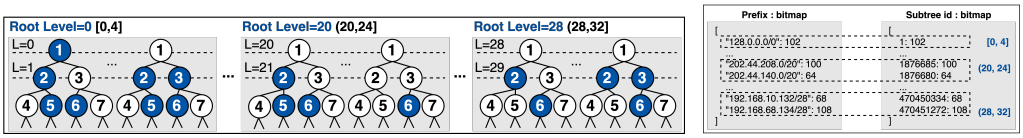
algorithm. Briefly, flows are first sorted based on the traffic bandwidth demands, and then assigned to the predetermined virtual links greedily. Specifically, the priority is used to ensure that virtual links across the PoP are filled before scheduling flows within the PoP.

4.2.3 Parallel Flow Scheduling at Each PoP. The flow scheduling algorithm can be executed in parallel at each PoP. We use a local controller in each PoP for scheduling flows from that PoP to other PoPs by assigning flows to the virtual link in parallel. The local controller within each PoP runs the algorithm individually and uploads these results to the central controller. It is notable that the capacity of the virtual link is determined based on the recommended bandwidth of the backbone network. The traffic workload will not exceed the recommended bandwidth, which will lead to no congestion. As shown in Figure 6, the central controller makes the peering link bursting decision and allocates the virtual link bandwidth between PoPs, i.e. 2Gbps from PoP A to PoP B and 1Gbps from PoP C to PoP B. After getting the bandwidth of the virtual links between PoPs, the local controller can then assign flows to a virtual link based on its capacity. Here, PoP assigned three flows: f_1 of 1Gbps, f_2 of 0.6Gbps, and f_3 of 0.4Gbps, the sum of which is no larger than the capacity of the virtual link from PoP A to PoP B. The first step is to conduct inter-PoP scheduling to ensure optimal utilization of the capacity across all virtual links. Scheduling of flows within a PoP can be regarded as a bin packing problem which can be solved in parallel [6]. This approach enables each PoP to autonomously handle data collection, optimal scheduling computation, and scheduling result dissemination.

5 Routing Table Compression

EdgeCross uses the routing computation module (§3) to determine the traffic routes based on the complete routing tables collected from all PRs. Besides, we need the complete routing table of cloud WAN to verify the generated routes before installing the new routes to PR devices. The storage of routing tables introduces a big challenge for the limited memory of the controller's server. Specifically, each entry in the routing table consists of a prefix and a series of routing information, e.g., next hop, origin, AS path, community_list, origin_as, local_pref, etc. A 64GB memory can only store ~15 million routing entries.

Therefore, we target compressing the routing table in the EdgeCross controller. Generally, the tree-based index structures are used to compress the routing table, such as Binary tree[29], Patricia trie[27], Radix tree[26], TreeBitmap[7], Poptrie[1], etc. For example, the routing information can be compressed by dictionary encoding, and an original table with a total of 918768 entries can be compressed to 119856 entries, with a compression ratio of about 7.6. However, in practice, the EdgeCross controller's memory still can not accommodate the compressed index of routing table entries for the entire network, the routing computations of which require a large number of data exchanges between memory and disk and slow down the actions of the longest matching, full matching, subnet matching, field filtering, etc. We notice that compressing all routes onto a single tree causes the depth of the tree to be too large, which affects the efficiency of routing index construction and computations. Updating a large tree frequently also incurs a significant computation overhead. Therefore, we consider how to efficiently compress routing entry indexes



(a) Logical structure (b) Memory structure

Fig. 7. Logical and Memory Data Structure: Forest

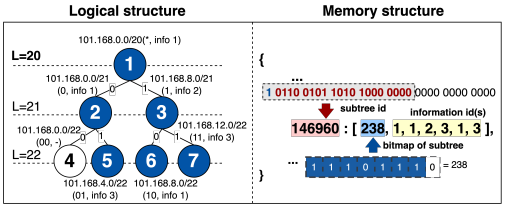


Fig. 8. Logical and Memory Data Structure: Subtree

Fig. 9. The network topology

while ensuring the functionality and performance of dynamic addition, deletion, modification, and querying.

Index Structure. To improve efficiency, it is crucial to effectively manage the scope of route computations and prioritize localized updates. We propose a hierarchical forest-based index structure, as illustrated in Figure 7(a), which decomposes a large tree into multiple flattened subtrees according to the prefix hierarchy (e.g., [/16, /20], [/20, /24]). This structure can reduce the path length of route computations and control the scope of routing updates within a smaller subtree.

However, the space required to maintain the hierarchical forest directly in memory remains non-trivial. We further propose a more compact routing index structure to compress every subtree, i.e., ForestBitmap[2]. By uniquely representing each subtree with two integers, i.e., subtree id and bitmap value, and mapping it to a memory entry, we can significantly reduce the memory overhead. As shown in Figure 7(b), the prefix "192.168.10.132/28" is encoded as a logical subtree with the root node "192.168.10.128/28", which has the id 470450334 as the subtree id. The bitmap value of the logical subtree, e.g., 68, indicates the non-null nodes with valid routes by setting the corresponding bits to "1" in the order of level traversal.

In addition, routing computations are primarily executed through efficient bit operations, while avoiding extensive search and backtracking on the tree structure, ensuring a better performance. This compact and flexible indexing structure can achieve $O(1)$ online update performance and is easily expandable.

Subtree Encoding. Irrespective of the pure prefix index, we need a more comprehensive approach to storing and indexing the entire routing tables with attribute information. To illustrate the process of subtree compression encoding, we use the 3-layer subtree as an example. As shown in Figure 8, the logical subtree stores the routing table entries, each of which consists of a prefix and its associated routing information. These entries can be encoded in a memory Map entry, where the Map key is the subtree id (e.g., 1464960) and the Map value is an array that contains the bitmap (e.g., 238) and a series of attribute id(s).

We obtain the subtree index id from the common binary prefix resulting from subnet aggregation, represented as 1 0110 0101 1010 1000 0000. We pad the most significant bit of 1 to acquire the unique subtree id of 146960. This id enables us to locate the subtree containing the route. A 3-layer

[2]ForestBitMap is implemented by JAVA similar to the general-purpose radix-tree implementation.

Days	1	2	3	4	5	6	7
BGP	1080.09	1106.51	1137.02	1162.89	1173.52	1181.04	1187.20
EdgeCross	734.03	741.36	753.73	760.99	768.30	773.93	777.97
Optimization	32.04%	33.00%	33.71%	34.56%	34.53%	34.47%	34.47%

Table 3. Cost optimization for offline testing under a 90% bursting threshold and a 0.1Mbps flow filter threshold.

Memory (MB)	Prefix Index	Routing Index
RadixTree[12]	204.37	556.48
EdgeCross	11.55	95.31
Optimization	17.69×	5.93×

Table 4. Memory optimization

subtree can store up to seven routes, so we employ an 8-bit bitmap to encode non-null nodes and achieve 238 (i.e., 11101110), with the least significant bit reserved.

ForestBitmap uses dictionary encoding to eliminate redundancy and acquire a dictionary table of the route attribute information. The encoded routing information table is shared by all routes, whose id is stored as the index of routing information according to the valid bits of the bitmap, e.g., 1,1,2,3,1,3. Thus, we can compress a 3-layer subtree with routing table entries into a memory entry.

Remark. Taking IPv4 as an example, a 5-layer subtree that stores 31 routing table entries can be represented by a Map entry. This allows for the compression of IPv4 BGP routing entries with over 900,000 unique prefixes to less than 170,000 Map entries. Compared to simple tree structures, the memory compression ratio is higher due to the utilization of compact structures and aligned integer encoding. Besides, it is more suitable for cache placement, thus enhancing the hit rate.

6 Evaluations

We evaluate EdgeCross through the topology given in Figure 9, which consists of 4 areas, 16 PoPs, and 860 peering links. 8 of the PoPs are equipped with monitoring devices, while the other 8 PoPs (in white) solely work as transit PoPs connected by the backbone. The total number of links in the backbone network is 54. The amount of flows in the test input is approximately 2 million per time slot and the flows are all from real business traffic of 1 week (from May 22, 2023 to May 28, 2023). The data from different weeks are very large (Tbits) but we have demonstrated the consistency of data in Table 3. Performance-sensitive flows are identified based on the application service class, constituting approximately 4% of the total flow count. The algorithm runs on a single server with 64GB of memory and 32 CPU cores. There are two parameters to be determined. We filter non-performance-sensitive small flows based on the flow filter threshold FT. Considering the congestion risk in the product network, we set the maximum percentage of peering link bursting capacity, named bursting threshold BT.

6.1 Cost Optimization

We first conduct offline tests for 1 to 7 days with hyperparameters such as a bursting threshold of 90% and a flow filter threshold of 0.1Mbps. Table 3 illustrates the 95^{th} percentile bandwidth costs of EdgeCross and traditional BGP under different test durations. The results indicate that, compared to BGP, EdgeCross achieves ~32.04-34.4% cost saving on different days. Moreover, this optimization ratio remains relatively stable on different days.

We then test the cost optimization of EdgeCross under different hyperparameters, benchmarking against traditional BGP. As shown in Figure 10, with a fixed bursting threshold, the percentage of cost savings increases with the decrease of the flow filter thresholds. However, when the flow filter threshold becomes sufficiently small (e.g., 0.1Mbps), further decreasing the flow filter threshold does not significantly increase the cost. This is mainly because an excessively low FT would lead the algorithm to schedule additional small flows. However, the total bandwidth proportion of these small flows is small, resulting in a negligible impact on cost optimization. Besides, BT significantly influences the results of cost optimization, as changes in BT imply variations in the maximum capacity of each peering link.

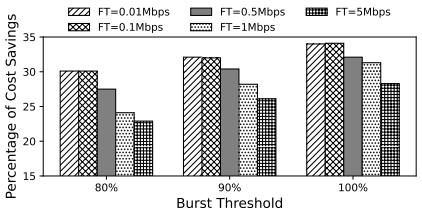


Fig. 10. Cost optimization with different bursting thresholds and flow filter thresholds.

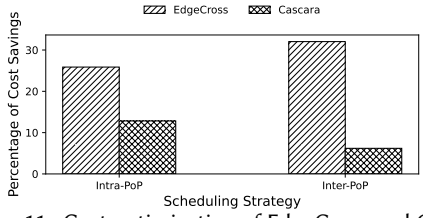


Fig. 11. Cost optimization of EdgeCross and Cascara.

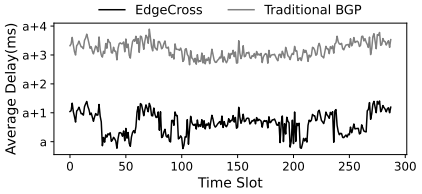


Fig. 12. Average delay of performance-sensitive flows using EdgeCross and traditional BGP.

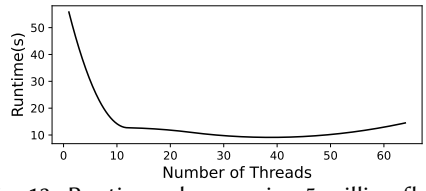


Fig. 13. Runtime when serving 5 million flows under different numbers of threads.

Finally, we compare the cost saving of EdgeCross and Cascara[25]. We consider two scenarios (i) flows can only be scheduled within a PoP and (ii) flows can be scheduled among PoPs. Notice that Cascara does not consider the backbone network, we added additional constraints of link capacity in their backbone for inter-PoP flow scheduling. As shown in Figure 11, EdgeCross achieves higher cost optimization compared to Cascara in our network topology. For example, for inter-PoP flow scheduling, EdgeCross achieves cost savings by 32% while Cascara achieves 6% cost optimization. For intra-PoP scheduling, EdgeCross still outperforms Cascara because it maximizes the utilization of the bursting peering links' free bandwidth. This is mainly attributed to EdgeCross's carefully designed algorithm that selects the bursting peering links for each time slot. Cascara's heuristic bursting peering links selection may result in underutilization of the free bandwidth of bursting peering links. Particularly in inter-PoP scheduling, insufficient backbone network link capacity means that only a few flows can be assigned to the bursting peering links.

6.2 Performance Optimization

We evaluate the performance of EdgeCross on performance-sensitive flows. Notice that Cascara aims to optimize cost in the first phase of EdgeCross for billable bandwidth, we can not demonstrate the performance optimization result of Cascara here. Our chosen performance metric is the delay of flows, encompassing both the delay at peering links and the backbone network delay. We evaluate the average latency of performance-sensitive flows in each time slot under both EdgeCross and traditional BGP routing. The test involved 288 time slots, employing a bursting threshold of 90% and a flow filter threshold of 0.1 Mbps. As illustrated in Figure 12, EdgeCross exhibits an approximate 2 ms reduction in the latency of performance-sensitive flows compared to BGP. Furthermore, roughly 67% of these performance-sensitive flows gain performance improvements. We achieve the similar relusts by using different parameters $FT = \{0.01, 0.1, 0.5, 1, 5\}$Mbps and $BT = \{100\%, 90\%, 80\%\}$.

6.3 Scalability

We scale up the number of flows to approximately 5 million by triplicating each flow, without filtering any flows, while other experimental settings remain consistent. We compare the average runtime of the algorithm per time slot under different numbers of threads. We allocate an equal number of threads for each PoP, enabling parallel scheduling both across PoPs and within each

Time (ms)	Index Build	Longest Match	Full Match	Subnet Match	Delete	Origin_as Filter	Update
RadixTree[12]	1534	4.463	8.251	758.3	122.6	75.44	1306
EdgeCross	533.8	1.453	2.554	39.31	25.76	55.79	81.31
Optimization	2.87×	3.07×	3.23×	19.29×	4.75×	1.35×	16.06×

Table 5. Routing computation optimization of ForestBitmap

Functionalities	**Edge Fabric[23]**	**Espresso [32]**	**Cascara [25]**	**OneWAN [19]**	**EdgeCross**
Optimization objective	Performance	Performance	Cost	Performance	Cost + performance
Flow granularity	Dest prefix	PoP + Service + dest prefix	Bandwidth	Node + Service + dest prefix	PoP + Service + dest prefix
Backbone capacity limitation	No	No	No	Yes	Yes
Distinguished service classes	No	Yes	No	Yes	Yes
Scheduling scope	Internet	Internet	Internet	Internet and datacenter	Internet

Table 6. Comparison on transport features of EdgeCross and other solutions.

PoP. As shown in Figure 13, when employing parallel execution with 32 threads, the algorithm's average runtime is approximately 80% shorter than the sequential execution. As we mentioned before, if we filter the flows by a *FT* of 0.1Mbps, the remaining number of flows will be decreased by about 94%. This indicates that if filtering is applied to flows with a *FT* of 0.1Mbps, our method can achieve our target, which is to schedule around 1 billion flows in approximately 10 seconds with 32 threads.

In summary, EdgeCross's scalability is attributed to three optimization strategies in the time slot scheduling algorithm: 1) The model for choosing the bursting egress reduces the time complexity of the original problem from exponential to linear in terms of the number of flows; 2) The flow filtering approach balances cost and performance optimization while reducing the number of flows the algorithm needs to consider 3) The computation of virtual link capacities between PoPs enables parallel execution of inter-PoP scheduling.

6.4 Efficient Routing Table Compression

We test the performance and computation efficiency based on the live network datasets using Google RadixTree (i.e., baseline) and our proposed compact data structure ForestBitmap. The results in Table 4 demonstrate that ForestBitmap achieves a compression ratio of 17.69 times when storing about one million unique prefixes, compared to the baseline for pure prefix index. For the complete routing index with extra attribute information, ForestBitmap achieves a compression ratio of 5.83 times. Moreover, we test the cumulative time of performing different routing computations on thousands of routes with the baseline and ForestBitmap respectively. As shown in Table 5, compared with the baseline, ForestBitmap improves computation efficiency by 1.35X to 19.29X. ForestBitmap achieves significant improvement in both compression effect and computation efficiency. Moreover, with the increasing number of routing table entries, ForestBitmap introduces greater benefits in terms of memory usage. As shown in Figure 14, the memory usage of ForestBitmap is 4.88 to 5.775 times less in comparison with RadixTree when the number of route entries is 100K and 900K respectively.

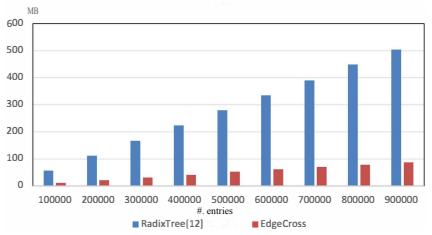


Fig. 14. Memory Usage

7 Related Work

Traffic Engineering in SDN-based Cloud WAN. In comparison to BGP [23, 32], EdgeCross differentiates between the service classes of flows. EdgeCross can tailor specific performance functions based on their requirements. By probing performance metrics for each backbone network

link and peering link, EdgeCross effectively assigns performance-sensitive flows to optimal peering links. Cloud providers also use traffic engineering at the edge of their networks to allocate traffic between the cloud and ISPs [14, 20, 21, 23, 32, 33] [10, 25]. TIPSY [22] and PAINTER [17] focuses on ingress traffic engineering. Espresso[32] and Edge Fabric[23] are SDN-based systems designed by Google and Facebook respectively. For example, Espresso prioritizes traffic based on the service class, ensuring superior performance for high-priority traffic. Edge Fabric identifies a flow by the destination prefix. However, they do not explicitly address backbone network capacity limitations, which is a challenge faced by EdgeCross. From the perspective of the controller, Espresso and EdgeCross use a global controller for route table update, while Edge Fabric employs distributed controllers for each PoP.

Unifying inter-domain TE and intra-domain TE. A group of works [4, 9, 19, 33] focus on the dynamic path selection and load balancing of traffic given the traffic matrix among the peering edge and data center. OneWAN [19] is a recent work that unifies inter-datacenter TE and Internet TE. OneWAN faces the primary challenge of selecting the optimal path within the cloud network using a unified single controller. EdgeCross and other works [2, 8, 14, 21, 23, 25, 28, 32, 33] focus on the inter-domain traffic routing between cloud and Internet through the PoP. EdgeCross chooses the optimal peering links, where WAN controllers and Internet controllers are independent. EdgeCross implements traffic engineering within the Internet controller, but considers backbone network link capacity as a constraint for flow scheduling. Table 6 shows the clear difference between EdgeCross and other network frameworks.

Routing Table Lookup. Google RadixTree is a 15-year-old longest-prefix-match implementation, which is widely used and stable in product environment. The limitation of memory resources can significantly impede system performance and usability. Therefore, the optimization of memory consumption, particularly in terms of the compression ratio of RIB, is regarded as a primary design objective in the original version. Notice that the LPM in DPDK can not be applied in our system because it consumes large memory for performance. We have reviewed multiple works such as Tree BitMap[7], SAIL[31], and Poptrie[1]. ForestBitmap outperforms these results, e.g. ForestBitmap achieves 39% faster than Poptrie.

8 Conclusion

We report our effort to realize traffic management of peering edges from the perspective of modern cloud providers. We optimize both cost and performance for large-scale flow scheduling. EdgeCross optimizes approximately 32% of the cost and reduces the average flow latency by 2 milliseconds. The network slicing schemes and routing compression algorithms allow EdgeCross to have the potential to complete the scheduling task for 1 billion flows within 10 seconds. After deploying EdgeCross, it successfully attracted and satisfied the demands of new customers.

Acknowledgments

We would like to thank our shepherd, the anonymous reviewers for their insightful comments and suggestions on this paper. This work has been partially supported by Jiangsu Key R&D BK20243053, NSFC (No. 62172204, 62325205, 62072228). The Fundamental Research Funds for the Central Universities, the Collaborative Innovation Center of Novel Software Technology and Industrialization, and the Jiangsu Innovation and Entrepreneurship (Shuangchuang) Program.

References

[1] Hirochika Asai and Yasuhiro Ohara. 2015. Poptrie: A Compressed Trie with Population Count for Fast and Scalable Software IP Routing Table Lookup. In Proceedings of ACM SIGCOMM.

[2] Matthew Caesar, Donald Caldwell, Nick Feamster, Jennifer Rexford, Aman Shaikh, and Jacobus Van Der Merwe. 2005. Design and implementation of a routing control platform. In Proceedings of USENIX NSDI.

[3] Martin Casado, Teemu Koponen, Scott Shenker, and Amin Tootoonchian. 2012. Fabric: a retrospective on evolving SDN. In Proceedings of ACM SIGCOMM.

[4] Fangfei Chen, Ramesh K Sitaraman, and Marcelo Torres. 2015. End-User Mapping: Next Generation Request Routing for Content Delivery. Proceedings of ACM SIGCOMM.

[5] Yunfei Cui, Zhiqiang Geng, Qunxiong Zhu, and Yongming Han. 2017. Multi-objective optimization methods and application in energy saving. Energy 125 (2017), 681–704.

[6] Tansel Dokeroglu and Ahmet Cosar. 2014. Optimization of one-dimensional bin packing problem with island parallel grouping genetic algorithms. Computers & Industrial Engineering 75 (2014), 176–186.

[7] Will Eatherton, George Varghese, and Zubin Dittia. 2004. Tree bitmap: hardware/software IP lookups with incremental updates. ACM SIGCOMM Computer Communication Review 34, 2 (2004), 97–122.

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

[9] Ashley Flavel, Pradeepkumar Mani, David Maltz, Nick Holt, Jie Liu, Yingying Chen, and Oleg Surmachev. 2015. FastRoute: A Scalable Load-Aware Anycast Routing Architecture for Modern CDNs. In Proceedings of USENIX NSDI.

[10] David K Goldenberg, Lili Qiuy, Haiyong Xie, Yang Richard Yang, and Yin Zhang. 2004. Optimizing cost and performance for multihoming. Proceedings of ACM SIGCOMM.

[11] Google. [n. d.]. OR-Tools. https://github.com/google/or-tools.

[12] Google. [n. d.]. Radix Tree. https://code.google.com/archive/p/radixtree/.

[13] Nyoman Gunantara. 2018. A review of multi-objective optimization: Methods and its applications. Cogent Engineering 5, 1 (2018), 1502242.

[14] Arpit Gupta, Laurent Vanbever, Muhammad Shahbaz, Sean P Donovan, Brandon Schlinker, Nick Feamster, Jennifer Rexford, Scott Shenker, Russ Clark, and Ethan Katz-Bassett. 2014. SDX: A Software Defined Internet Exchange. Proceedings of ACM SIGCOMM.

[15] 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 Proceedings of ACM SIGCOMM.

[16] 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 ACM SIGCOMM.

[17] Thomas Koch, Shuyue Yu, Sharad Agarwal, Ethan Katz-Bassett, and Ryan Beckett. 2023. PAINTER: Ingress Traffic Engineering and Routing for Enterprise Cloud Networks. In Proceedings of ACM SIGCOMM.

[18] Abdullah Konak, David W Coit, and Alice E Smith. 2006. Multi-objective optimization using genetic algorithms: A tutorial. Reliability engineering & system safety 91, 9 (2006), 992–1007.

[19] Umesh Krishnaswamy, Rachee Singh, Paul Mattes, Paul-Andre C Bissonnette, Nikolaj Bjørner, Zahira Nasrin, Sonal Kothari, Prabhakar Reddy, John Abeln, Srikanth Kandula, et al. 2023. OneWAN is better than two: Unifying a split WAN architecture. In Proceedings of USENIX NSDI.

[20] Raul Landa, Lorenzo Saino, Lennert Buytenhek, and João Taveira Araújo. 2021. Staying alive: Connection path reselection at the edge. In Proceedings of USENIX NSDI.

[21] Hongqiang Harry Liu, Raajay Viswanathan, Matt Calder, Aditya Akella, Ratul Mahajan, Jitendra Padhye, and Ming Zhang. 2016. Efficiently delivering online services over integrated infrastructure. In Proceedings of USENIX NSDI.

[22] Michael Markovitch, Sharad Agarwal, Rodrigo Fonseca, Ryan Beckett, Chuanji Zhang, Irena Atov, and Somesh Chaturmohta. 2022. TIPSY: predicting where traffic will ingress a WAN. In Proceedings of ACM SIGCOMM.

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

[24] David B Shmoys and Éva Tardos. 1993. An approximation algorithm for the generalized assignment problem. Mathematical programming 62, 1-3 (1993), 461–474.

[25] Rachee Singh, Sharad Agarwal, Matt Calder, and Paramvir Bahl. 2021. Cost-effective cloud edge traffic engineering with Cascara. In Proceedings of USENIX NSDI.

[26] Keith Sklower. 1991. A tree-based packet routing table for Berkeley unix.. In USENIX Winter, Vol. 1991. 93–99.

[27] Wojciech Szpankowski. 1990. Patricia tries again revisited. Journal of the ACM (JACM) 37, 4 (1990), 691–711.

[28] Vytautas Valancius, Bharath Ravi, Nick Feamster, and Alex C Snoeren. 2013. Quantifying the Benefits of Joint Content and Network Routing. In Proceedings of ACM SIGMETRICS.

[29] Marcel Waldvogel, George Varghese, Jon Turner, and Bernhard Plattner. 1997. Scalable high speed IP routing lookups. In Proceedings of ACM SIGCOMM.

[30] Kaicheng Yang, Yuanpeng Li, Sheng Long, Tong Yang, Ruijie Miao, Yikai Zhao, Chaoyang Ji, Penghui Mi, Guodong Yang, Qiong Xie, Hao Wang, Yinhua Wang, Bo Deng, Zhiqiang Liao, Chengqiang Huang, Yongqiang Yang, Xiang Huang, Wei Sun, and Xiaoping Zhu. 2023. AAsclepius: Monitoring, Diagnosing, and Detouring at the Internet Peering Edge. In 2023 USENIX Annual Technical Conference (ATC).

[31] Tong Yang, Gaogang Xie, YanBiao Li, Qiaobin Fu, Alex X. Liu, Qi Li, and Laurent Mathy. 2014. Guarantee IP lookup performance with FIB explosion. In Proceedings of ACM SIGCOMM.

[32] Kok-Kiong Yap, Murtaza Motiwala, Jeremy Rahe, Steve Padgett, Matthew Holliman, Gary Baldus, Marcus Hines, Taeeun Kim, Ashok Narayanan, Ankur Jain, et al. 2017. Taking the Edge off with Espresso: Scale, Reliability and Programmability for Global Internet Peering. In Proceedings of ACM SIGCOMM.

[33] Zheng Zhang, Ming Zhang, Albert G Greenberg, Y Charlie Hu, Ratul Mahajan, and Blaine Christian. 2010. Optimizing Cost and Performance in Online Service Provider Networks. In Proceedings of USENIX NSDI.

A Appendix

A.1 Estimating 95^{th} Percentile Billable Bandwidth Algorithm

In this section, we show in detail the algorithm for estimating the 95^{th} percentile billable bandwidth formalization problem as mentioned in §3.3. The purpose of this algorithm is to give an estimated billable bandwidth p_i for each peering link l_i based on the bandwidth forecast within each PoP for the next month. The formulation is depicted in Algorithm 1.

Decision variables. Peering link bursting is the maximum available bandwidth to be allocated to the peering link, which avoids congestion. Specifically, when k_i^j is 1, the bandwidth allocation of egress l_i can be extended to C_i, and we refer to this as bursting. When k_i^j is 0, the bandwidth allocation of egress l_i can only be extended to p_i, and we refer to this as not bursting.

Constraints. The constraint 1a states that 5% of the total time slots in a month should be reserved for peering link bursting. The constraint 1b specifies that the estimated billable bandwidth of peering links should be greater than the guaranteed bandwidth. The constraint 1c establishes that the bandwidth along the backbone link must be within its capacity constraint. The constraint 1d ensures that the total bandwidth demand during a specific time slot doesn't exceed the total capacity of all peering links. The constraints 1e and 1f dictate that the remaining bandwidth at the PoP after scheduling - accounting for both incoming and outgoing bandwidth - must not be greater than the total available capacities of peering links at the PoP and not be less than zero. The last four constraints 1g, 1h, 1i and 1j imply that if the peering link l_i bursts in time slot t_j, then the maximum available capacity y_i^j is between 0 and C_i; otherwise, it is between 0 and p_i. The large constant M is used to convert nonlinear constraints into linear constraints.

A.2 Flow Scheduling

In this section, we show in detail the algorithm for the flow scheduling formalization problem as mentioned in §3.4. The algorithm schedules flows online based on the set of p_i given by Algorithm 1. It considers multiple optimization objectives scheduling, including performance and cost optimization. The formulation is depicted in Algorithm 2.

Decision variables. The flow scheduling scheme assigns network flow to peering links in L in the current time slot, $t \in [1, .., n]$. We introduce the decision variable $b_{(u,v)}$ to represent the bandwidth allocation on backbone link (u, v) from PoP u to PoP v in the current time slot. y_i is the maximal available bandwidth in the current time slot. The binary decision variable x_i^f indicates whether flow f egresses through peering link l_i within the current time slot, and the binary decision variable $x_{(u,v)}^f$ indicates whether it traverses backbone link (u, v) within the current time slot.

Algorithm 1 Estimating 95^{th} Percentile Billable Bandwidth Algorithm

Inputs:

Table 2: Common input parameters

b_u^j: Total traffic of PoP u in time slot t_j

d_j: Egress demand from the WAN in time slot t_j

Outputs:

$b_{(u,v)}^j$: Bandwidth of backbone link from PoP u to PoP v in time slot t_j

p_i: Estimated 95th-percentile billable bandwidth of peering link l_i

k_i^j: Whether peering link l_i bursts in time slot t_j

y_i^j: Maximal available bandwidth of peering link l_i in time slot t_j. When k_i^j equals 1, y_i^j takes the value of p_i; conversely, when k_i^j equals 0, y_i^j assumes the value of C_i

Minimize:

$$\sum_{l_i \in L} c_i p_i$$

Subject to:

$$\forall l_i \in L : \sum_{t_j \in T} k_i^j = \frac{n}{20} \tag{1a}$$

$$\forall l_i \in L : ub_i \leq p_i \tag{1b}$$

$$\forall (u,v) \in E, \forall t_j \in T : b_{(u,v)}^j \leq C_{(u,v)} \tag{1c}$$

$$\forall t_j \in T : d_j \leq \sum_{l_i \in L} y_i^j \tag{1d}$$

$$\forall u \in V, \forall t_j \in T : \sum_{(v,u) \in E_u^D} b_{(v,u)}^j - \sum_{(u,v) \in E_u^S} b_{(u,v)}^j + b_u^j \leq \sum_{l_i \in L_u} y_i^j \tag{1e}$$

$$\forall u \in V, \forall t_j \in T : \sum_{(u,v) \in E_u^D} b_{(u,v)}^j - \sum_{(u,v) \in E_u^S} b_{(u,v)}^j + b_u^j \geq 0 \tag{1f}$$

$$\forall t_j \in T, \forall l_i \in L : y_i^j \leq p_i + M k_i^j \tag{1g}$$

$$\forall t_j \in T, \forall l_i \in L : y_i^j \geq p_i - M k_i^j \tag{1h}$$

$$\forall t_j \in T, \forall l_i \in L : y_i^j \leq C_i + M \left(1 - k_i^j\right) \tag{1i}$$

$$\forall t_j \in T, \forall l_i \in L : y_i^j \geq C_i - M \left(1 - k_i^j\right) \tag{1j}$$

Constraints. The constraint 2a ensures that each flow needs to be scheduled to exact one peering link. The constraint 2b restricts that the sum of the bandwidth of flows scheduled to each peering link cannot exceed the available capacity of that peering link. The constraint 2c specifies that the utilization capacity of each backbone link should not exceed its maximum available capacity. The constraints 2d and 2e signify that for each flow f, if it's scheduled to PoP u, then a unique egressing peering link for this flow can be found within PoP u. For other PoPs, this flow might transit through, utilizing an equal amount of inbound and outbound backbone network links for that PoP. The constraints 2f and 2g ensure that the sum of the bandwidth for flows within each PoP should be non-negative and not exceed the aggregate available capacity of the peering links within the PoP after scheduling. The last five constraints 2h, 2i, 2j, 2k and 2l describe the relationship between the

maximum available capacity of each peering link. If peering link l_i bursts, the maximum available capacity is C_i; otherwise, it is p_i.

A.3 Bandwidth Allocation Formulation

We model the bandwidth allocation problem, which is depicted in Algorithm 3.

Input parameters. In addition to the common inputs listed in Table 2, the algorithm requires only the current time slot and the total traffic in each PoP in the current time slot.

Variables. We set two variables. $b_{(u,v)}$ denotes the usage of the backbone link from PoP u to PoP v in the current time slot. This variable will be used to indicate the utilized capacity of each physical backbone link. k_i is a binary variable indicating if peering link i needs to burst in the current time slot. This variable will be used to determine the maximum available capacity for each peering link.

Objective function. To minimize the cost, the objective function is set to minimize the sum of unused capacity at each peering link.

Constraints. Constraints mainly stem from limited peering link capacity and backbone link capacity. The first and second constraints ensure that the bandwidth allocated to each peering link does not exceed the maximum available capacity of that peering link. The third constraint ensures that the usage of each backbone link does not exceed its maximum capacity. The last two constraints imply that the sum of the bandwidth of the remaining traffic within each PoP after scheduling cannot be negative and cannot exceed the sum of the available capacity of all the peering links within that PoP.

Algorithm 2 Flow Scheduling Algorithm

Inputs:

Table 2: Common input parameters

b_u: Total traffic bandwidth of PoP u in the current time slot

β: A hyperparameter adjusts the trade-off between cost and performance

Outputs:

$b_{(u,v)}$: Bandwidth of backbone link from PoP u to PoP v in the current time slot

p_i: Estimated 95^{th} percentile billable bandwidth of peering link i

k_i: Whether peering link i bursts in the current time slot

y_i: Maximal available bandwidth of peering link i in the current time slot

x_i^f: Whether f egress via peering link i in the current time slot

$x_{(u,v))}^f$: Whether f go through backbone link (u,v) in the current time slot

Minimize:

$$\min \frac{\beta}{|F_s|} \sum_{f \in F_s} \sum_{l_i \in L} w_1(f) x_i^f perf(i,f) + (1-\beta) \frac{\sum_{l_i \in L} y_i}{\sum_{l_i \in L} C_i}$$

Subject to:

$$\forall f \in F : \sum_{l_i \in L_f} x_i^f = 1 \tag{2a}$$

$$\forall l_i \in L : \sum_{f \in F} V_f x_i^f \leq y_i \tag{2b}$$

$$\forall (u,v) \in E : \sum_{f \in F} V_f x_{(u,v)}^f \leq C_{(u,v)} \tag{2c}$$

$$\forall f, \forall u \text{ where } S(f) = u : \sum_{f \in F} x_{(u,v)}^f + \sum_{l_i \in L_u} x_i^f = 1 \tag{2d}$$

$$\forall f, \forall u \text{ where } S(f) \neq u : \sum_{f \in F} x_{(u,v)}^f + \sum_{l_i \in L_u} x_i^f = \sum_{f \in F} x_{(v,u)}^f \tag{2e}$$

$$\forall u \in V : \sum_{f \in F} \sum_{(v,u) \in E_u^D} V_f x_{(v,u)}^f - \sum_{f \in F} \sum_{(u,v) \in E_u^S} V_f x_{(u,v)}^f + b_u \leq \sum_{l_i \in L_u} y_i \tag{2f}$$

$$\forall u \in V : \sum_{f \in F} \sum_{(v,u) \in E_u^D} V_f x_{(v,u)}^f - \sum_{f \in F} \sum_{(u,v) \in E_u^S} V_f x_{(u,v)}^f + b_u \geq 0 \tag{2g}$$

$$\forall l_i \text{ without free slots } : y_i = p_i \tag{2h}$$

$$\forall l_i \text{ with free slots } : y_i \leq p_i + M k_i \tag{2i}$$

$$\forall l_i \text{ with free slots } : y_i \geq p_i - M k_i \tag{2j}$$

$$\forall l_i \text{ with free slots } : y_i \leq C_i + M(1 - k_i) \tag{2k}$$

$$\forall l_i \text{ with free slots } : y_i \geq C_i - M(1 - k_i) \tag{2l}$$

Algorithm 3 Bandwidth Allocation Formulation

Inputs:

Table 2: Common input parameters

b_u: Total traffic bandwidth of PoP u in the current time slot.

Outputs:

$b_{(u,v)}$: Bandwidth of backbone link from PoP u to PoP v in the current time slot.

k_i: Whether peering link i bursts in the current time slot.

Minimize:

$$\sum_{l_i \in L} y_i$$

Subject to:

$$\begin{aligned}
&\forall l_i \text{ without free slots}: && y_i = p_i \\
&\forall l_i \text{ with free slots}: && y_i = p_i(1-k_i) + C_i k_i \\
& && \textstyle\sum_{u \in V} b_u \le \sum_{l_i \in L} y_i \\
&\forall (u,v) \in E: && b_{(u,v)} \le C_{(u,v)} \\
&\forall u \in V: && \textstyle\sum_{(u,v) \in E_u^D} b_{(u,v)} - \sum_{(u,v) \in E_u^S} b_{(u,v)} \\
& && +b_u \le \textstyle\sum_{l_i \in L_u} y_i \\
&\forall u \in V: && \textstyle\sum_{(u,v) \in E_u^D} b_{(u,v)} - \sum_{(u,v) \in E_u^S} b_{(u,v)} \\
& && +b_u \ge 0
\end{aligned}$$

A.4 Heuristic Time Slot Flow Scheduling Algorithm

The heuristic time slot flow scheduling algorithm is introduced in Algorithm 4.

Algorithm 4 Flow Scheduling to Destination PoP

Require: Set of PoPs V,
recommended physical link capacity usage $r_{(u,v)}$ between each PoP u and PoP v,
set of flows F
Ensure: Destination PoP $dest_f$ for each flow $f \in F$

Initialize virtual link capacity $B(u, v) = 0$ for all $u \in V, v \in V$
for each pair of PoP nodes (u, v), where $u \in V, v \in V, u \neq v$ **do**
  while an augmenting path exists within all r on the path from u to v **do**
    Let the capacity of the augmenting path be x
    Update virtual link capacity: $B_{(u,v)}$ += x
  end while
end for
Sort flows in F by bandwidth in descending order
for each flow $f \in F$ **do**
  if f is performance-sensitive **then**
    Find the peering link l_i that minimizes the performance function $perf(i, f)$
    Set $dest_f = v$ where v is the PoP associated with l_i
    Update $B(u, v)$ −= bandwidth of flow f
  end if
end for
for each flow $f \in F$ **do**
  Let u be the PoP flow f is outbound from
  for each PoP $v \in V$, where $v \neq u$ **do**
    if f is not performance-sensitive AND $B(u, v) \geq$ bandwidth of flow f **then**
      Set $dest_f = v$
      Update $B(u, v)$ −= bandwidth of flow f
      break
    end if
  end for
  if $target_f$ is $NULL$ **then**
    Set $dest_f = u$
  end if
end for
return $dest$

Received June 2024; revised September 2024; accepted October 2024