Taming the Shift: Investigating Flow Shifting Strategies for Traffic Engineering

Benjamin Schichtholz*, Oliver P. Waldhorst†, Roland Bless*, Michael König*, and Martina Zitterbart*
*Institute of Telematics, Karlsruhe Institute of Technology, Karlsruhe, Germany
†Institute of Data-Centric Software Systems (IDSS), Karlsruhe University of Applied Sciences, Karlsruhe, Germany
{schichtholz, bless, m.koenig, zitterbart}@kit.edu, oliver.waldhorst@h-ka.de

***Abstract*—Traffic Engineering (TE) typically shifts flows to different paths to optimize resource utilization in the network. These shifts can cause out-of-order packets or unnecessary retransmissions, resulting in a performance deterioration for flows. Minimizing this by keeping paths persistent for a majority of flows may reduce the overall impact of flow shifts. We investigate different shifting strategies that are subject to a fundamental tradeoff between reducing the number of shifted flows and fast convergence times, i.e., the time to reach the updated TE configuration. We developed the flow-level simulation toolchain TEFlowSim to evaluate the shifting strategies based on real-world traffic. We find that keeping most flows on the same path while shifting only high-bandwidth, long-running elephant flows reduces the number of shifted flows to a minimum while fulfilling the targeted reallocation of bandwidth. Our experiments show that compared to state-of-the-art hash-based strategies, only shifting elephant flows reduces the number of shifted flows by up to 45x while achieving fast convergence.**

***Index Terms*—Traffic Engineering, Shifting, Simulation**

I. Introduction

Global data center providers [1], cloud providers [2]–[6], ISPs [7], and research networks [8] carry increasingly large traffic volumes. Traffic engineering (TE) allows the optimization of the traffic distribution in the corresponding networking domain to achieve predefined performance goals, such as minimizing maximum link utilization or maximizing throughput. This optimization is performed in regular *TE intervals* traditionally in the order of 5–15 minutes [9]–[11]. It uses traffic measurements or predictions as input and generates *split ratios* that define how traffic should be distributed across multiple paths between the domain's source-destination pairs.

As split ratios change over time, traffic flows may need to be shifted to other paths. We define a flow as a unidirectional sequence of packets that shares the same 5-tuple (src/dst IP, src/dst port and transport protocol). The paths to which flows are shifted may have different characteristics, e.g., regarding the round trip time (RTT) or bottleneck bandwidth. However, different RTTs between the new and old paths can lead to either packets arriving out of order at their destination or retransmission timeouts due to late arrival of a packet. This can result in congestion control algorithms falsely detecting congestion [12], reducing the sending rate, and causing unnecessary retransmissions. As a consequence, these path shifts can lead to *disruptions* such as a reduced achieved throughput [13]–[15] and prolonged flow completion times. The latter especially affects the performance of short-lived flows adversely [16].

Recent work has considered sub-minute [17], [18], or even sub-second [19] TE intervals. TE mechanisms can support such short TE intervals (up to 10 s) using machine learning (ML) [9], [10], [20], which provides faster computation times than traditional approaches based on Linear Programming. They allow more fine-grained optimization of traffic distribution by reacting rapidly to short-term traffic fluctuations, e.g., bursts. However, the drawbacks for the shifted flows intensify with an increasing number of affected flows.

As split ratios change more frequently, *shifting strategies* that select flows to be shifted become more relevant. Existing hashing-based strategies shift randomly selected flows proportional to the split ratio to the set of running flows. Although this random selection of shifted flows is stateless and requires no additional knowledge about the flows, it does not align with real-world traffic characteristics: The selection of flows to be shifted is based on quasi-uniform flow distributions, even though flow duration and throughput distributions are typically heavy-tailed [21]–[23]. As a result, many short-lived, low-throughput *mice flows* are shifted. Even though they experience disruption, each of these mice flows contributes only a negligible amount of load.

In this paper, we focus on how to reduce the number of shifted flows to minimize disruption while realigning the load distribution to the split ratios fast. Specifically, we compare the hashing-based strategy with multiple improved shifting strategies. These strategies either rely on newly arriving flows to realign the load or shift only selected long-lived, high-throughput elephant flows. To thoroughly evaluate these strategies in realistic scenarios, we developed *TEFlowSim*: a toolchain for conducting simulation experiments. It operates on flow-level granularity, thus allowing efficient measurements on the overall flow performance for large real-world traces (up to 1.5 TB). As a result, the main contributions of this paper consist of detailed experiments on the shifting strategies as well as the toolchain itself:

Support for CAIDA's Internet Traces is provided by the National Science Foundation (CNS-2120399) and (OAC-2131987)

ISBN 978-3-903176-82-9 ©2026 IFIP

- A thorough evaluation of the tradeoffs of different shifting strategies with respect to reducing disruption while realigning the load in time.
- A shifting strategy that does not shift any running flows but merely uses newly started flows to reach the load distribution. Our results show that this does not align with short TE intervals.
- A shifting strategy that shifts only selected elephant flows leads to fast convergence. The proposed elephant shifting strategy can also reduce the number of shifts by up to 45× compared to state-of-the-art hashing-based strategies.
- TEFlowSim: A novel and highly efficient toolchain for flow-based performance evaluations of TE flow shifting strategies on real-world traffic traces.

II. Related Work

TE with Reduced Flow Shifts. Adjusting the TE optimization to limit split ratio updates has been proposed to reduce the number of flow shifts [19], [24]. As the split ratios do not operate on a per-flow granularity, many mice flows may still be shifted. Alternatively, [25] proposed only shifting traffic between a subset of source-destination pairs while statically routing the rest. This may result in unfairness between source-destination pairs, as the shifted flows suffer from disruption, while the statically routed flows are unaffected.

TE with Short Time Intervals. TE mechanisms with shorter TE intervals have been proposed to react rapidly to dynamic traffic changes [19], [26], [27]. In [18], the authors observed that flows between Amazon's data centers are reassigned to different paths in approximately 10-second intervals. These mechanisms typically focus more on realigning the network load distribution fast, and often do not consider disruptions for frequently shifting running flows.

Traffic Splitting. Several different traffic splitting mechanisms have been proposed in related work. In [28]–[30], the authors measured how accurately traffic splitting mechanisms align the load distribution with a static split ratio. In contrast, our work focuses on shifting strategies that realign the load for updated split ratios. The shifting strategy introduced in [31] faces the same challenges as the evaluated hashing-based shifting strategy. [32], [33] suggested applying a set of TCAM rules to implement traffic splitting and focus on polynomial-time algorithms to calculate a minimal set of these TCAM rules. Although [32] also addressed the drawbacks of shifts, it did not consider elephant flows and still approximately shifted a number of flows proportional to the split ratio difference (see Section III-A).

Simulation Toolchains. REPETITA [34] and Yates [35] simulate TE algorithms using traffic matrices for whole topologies. These traffic matrices model only traffic aggregates between source-destination pairs, while evaluating flow shifting strategies requires modeling 5-tuple flows. In contrast, packet-level network simulators (e.g., ns-3 [36]) struggle to simulate high datarates (e.g., 100 Gbps) and are too fine-grained, because the burstyness of individual flows is negligible for evaluating persistent load changes.

flow	through-put
f_1	20
f_2	20
f_3	10
f_4	30
f_5	20

	initial split ratio	updated split ratio
ABD	0.5	0.2
ACD	0.5	0.8

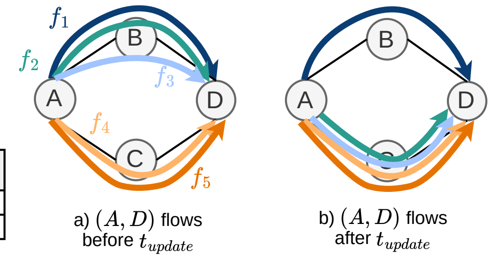

Fig. 1. TE shift example. (a) Five flows are distributed across two paths between A and D, according to split ratio at t_1. (b) As split ratios change at t_2, two flows are shifted to bottom path.

III. TE Flow Shifting

This section describes basic TE concepts, discusses why shifts are necessary, and introduces shifting strategies.

A. Basic TE Example

TE optimizes the network's traffic flow, typically using a *traffic matrix* as input. The traffic matrix denotes the average throughput over a time interval between each source-destination pair of a network domain. TE optimizes towards an optimization goal, e.g., minimizing maximum link utilization [10], [20], or maximizing throughput [9], [37]. Typical TE mechanisms produce *split ratios* that specify the distribution of traffic between each source-destination pair across n predefined paths.

The example in Figure 1 shows (A, D) as a source-destination pair and a set of five flows. In order to (accurately) match the initial split ratio $[0.5, 0.5]$, the traffic aggregate is split across the two paths (ABD, ACD) that interconnect A with D. In Figure 1, at t_{update}, TE updates the initial split ratio $[0.5, 0.5]$ to the updated split ratio $[0.2, 0.8]$. Consequently, a *shifting strategy* realigns the load distribution by selecting a subset of flows and shifting them to different paths. In our simple example, the updated split ratio can be reached accurately by moving flows f_2 and f_3 from path ABD to path ACD.

The number of flows to shift is influenced by the *split ratio difference* d, i.e., the fraction of total load that must be moved. We define d as half the sum of positive and negative differences across paths between the initial and updated split ratios: $d = 0.5 \cdot \sum_i |initial_i - updated_i|$. This is half the L1 norm over the vector of absolute differences across paths. For example, updating $[0.5, 0.5]$ to $[0.2, 0.8]$ yields $d = 0.5 \cdot (|0.5 - 0.2| + |0.5 - 0.8|) = 0.3$. Therefore, 30 % of the total load must be shifted from ABD to ACD (e.g., by using flows f_2 and f_3). For an arbitrary number of n paths, $l_{\text{shift}} = updated_i - initial_i$ $(i \in \{1, \cdots, n\})$ indicates the relative amount of load that each path must either shed or gain to achieve the new split ratios.

B. Shifting Strategies

Ideally, shifting strategies realign the load after t_{update} as fast as possible while minimizing the number of shifted flows.

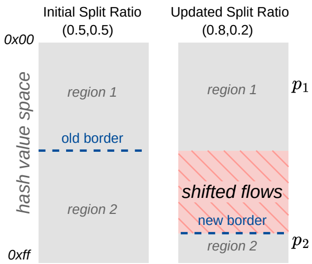

Fig. 2. HashShift Example: For the updated split ratio, flows assigned to new regions are shifted.

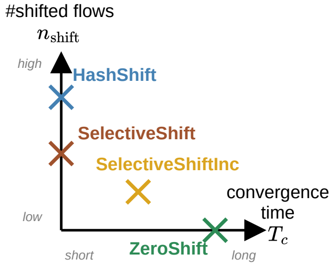

Fig. 3. Tradeoff between convergence times T_c and number of shifted flows n_{shift} for each shifting strategy.

A shifting strategy addresses this by shifting either running flows to different paths, and/or optimizing the assignment of paths for newly arriving flows. Thus, a shifting strategy comprises four components:

a) *Candidate Flow Selection*: Selects the flow candidates F_{cand}, a subset of the flows that are running at t_{update}, i.e., F_{running}.
b) *Shifted Flow Selection*: From F_{cand}, selects n_{shift} flows to shift: F_{shift}. We split the shifted flow selection into two steps (Candidate and Shifted Flow Selection), as F_{cand} may comprise a higher load than the split ratio difference d. Thus, F_{shift} is used to approximate d more precisely.
c) *Shifting Policy*: Determines number and frequency of shifting flows in F_{shift}. A *one-shot* policy shifts all flows from F_{shift} to a different path at once. An *incremental* policy shifts k flows with frequency f.
d) *Path Assignment Mechanism*: Determines which path a newly arriving flow should be assigned to.

We use two different path assignment mechanisms for our evaluated shifting strategies. First, *Hashing-based Path Assignment* hashes IP-header fields of flows (e.g., 5-tuple) and separates the hash space into regions (shown in Figure 2 left). Each region is mapped to a path. Therefore, as a flow hash falls into a certain region, the flow is assigned to the corresponding path. By scaling the region sizes according to the split ratios, the load distribution approximates the split ratios probabilistically. However, the approximation error increases with less uniform flow throughput distributions. Second, *Largest-gap-first* assigns each newly arriving flow to the single path with the largest negative gap between the current load distribution and the split ratio. For n paths, the path with the largest negative gap is: i.e., $path_i = \arg\min_{i \in \{1,\cdots,n\}}(\text{load}_i - \text{splitratio}_i)$. It provides flow-level control over the path assignment and can approximate the split ratio more precisely than the hashing-based mechanism.

We explore four different shifting strategies:

1) HashShift: Shifts a subset of running flows, determined by the updated regions in the hash value space, as shown in Fig. 2. All flows are potential candidates to be shifted: $F_{\text{running}} = F_{\text{cand}}$. As this strategy uses hash-based path assignment, each flow hash falls into one region. After t_{update}, regions are resized according to the updated split ratio (e.g., $[0.8, 0.2]$). Using the one-shot shifting policy, all flows with hash values falling in a different region are shifted. This results in approximately $n_{\text{shift}} = |F_{\text{running}}| \cdot d$ flows to be shifted.
Example: $d = 0.3$, $|F_{\text{running}}| = 100 \Rightarrow 30$ flows are shifted.
Pros: Does not require per-flow state and is implemented in most of today's switches.
Cons: Flows to be shifted are selected from a quasi-uniform distribution. However, flow throughput distributions typically have heavy-tailed characteristics. Therefore, many short-running, low-throughput mice flows are shifted. Each mice flow is disrupted even though it has only a negligible impact on the load distribution.

2) ZeroShift: Does not shift any flows (i.e., $F_{\text{cand}} = \emptyset, n_{\text{shift}} = 0$), but uses largest-gap-first to forward each newly started flow to the path where the load has to be increased.
Example: Initial split ratio $[0.5, 0.5]$, updated split ratio $[0.8, 0.2]$: No flows are shifted, but newly arriving flows are assigned to first path, until the load distribution aligns with the updated split ratio.
Pros: No disruption, as no running flows are shifted.
Cons: This strategy may not reach the updated split ratio fast, thus introducing a convergence time: $T_c = t_{\text{converge}}$-$t_{\text{update}}$ with t_{converge} being the point in time when the load distribution is within a target boundary for a certain duration.

3) SelectiveShift: Selects high-throughput, long running elephant flows as shift candidates F_{cand}. We assume that the elephant bandwidth share s of the total bandwidth is stable and can be estimated. We show that this is a reasonable assumption in Sec. V-B. From F_{cand}, a number of elephants proportional to the split ratio difference d and the elephant bandwidth share s are shifted. These flows F_{shift} are shifted to the other path using the one-shot shifting policy. In the general case with n paths, the flows in F_{shift} are distributed to each path by dividing the vector of shifted load l_{shift} (Sec. III-A) by s: $l_{\text{selshift}} = \frac{l_{\text{shift}}}{s}$. This strategy uses largest-gap-first for path assignment. This means that after t_{update}, while flows in F_{shift} are shifted directly, all other newly arriving flows are assigned to paths using largest-gap-first. This ensures that convergence is achieved if F_{shift} provides insufficient bandwidth share to realign the load.

Example: $d = 0.3$, $s = 0.6$, $|F_{\text{cand}}| = 20 \Rightarrow 10$ elephants are shifted, $10 = |F_{\text{shift}}| = |F_{\text{cand}}| \cdot \frac{d}{s} = 20 \cdot \frac{0.3}{0.6}$.
Pros: Shifts only a low number of flows that contribute significantly to the load and keeps the paths for all other flows unchanged.
Cons: Requires an elephant flow detection mechanism. If flows in F_{shift} are shifted directly and convergence is achieved instantly, newly arriving flows are not necessary for convergence. By shifting only a subset of F_{shift} and simultaneously leveraging newly arriving flows, n_{shift} can be reduced even further (using SelectiveShiftInc).

4) SelectiveShiftInc: Works like SelectiveShift, but instead of using the one-shot policy that shifts a subset of elephants at once, it shifts them incrementally and also relies on other newly starting flows to reach the target split ratio. For the incremental policy, the parameters k and f must be estimated.

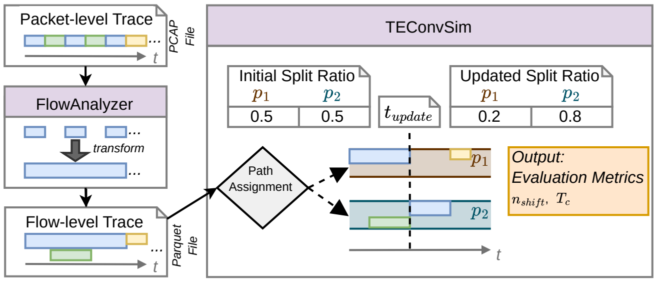

Fig. 4. TEFlowSim toolchain: The FlowAnalyzer transforms a pcap-based traffic trace into a parquet-based flow-level trace. The simulator TEConvSim applies a split ratio change and shifts flows using a shifting strategy.

Example: $d = 0.3$: From the selected elephant flows F_{cand}, $k = 3$ elephants are shifted every $f^{-1} = 100\,\text{ms}$.
Pros: May reduce n_{shift}, depending on the choice of k and f. Fewer elephants are shifted at regular frequencies f, while at the same time newly starting flows are assigned to the to-be-increased path as well.
Cons: Requires an elephant flow detection mechanism and has longer convergence times compared to SelectiveShift.

The shifting strategies make different assumptions regarding scalability and functionality. First, the strategies relying on largest-gap-first require separating and detecting newly started flows. This could be realized by detecting SYN-packets for TCP flows or employing scalable probabilistic data structures (e.g., bloom filters). Second, elephant detection mechanisms are required for SelectiveShift and SelectiveShiftInc, and have been proposed in related work [38], [39]. However, concrete implementations of these are beyond the scope of this paper. Our focus is a "what-if" analysis: To what extent can we ideally reduce shifts while supporting convergence within short TE intervals (up to 10 s), given real-world characteristics? Figure 3 illustrates that these shifting strategies span a tradeoff between two extremes: either many flows have to be shifted in order to converge fast, or no flows are shifted at all while relying on newly starting flows to reach the updated split ratio with the drawback of longer convergence times. The scalability of tracking individual elephant flows compared to using stateless hashing (ZeroShift) is out of scope for our "what-if" analysis, and has been examined in [38].

IV. Evaluation Methodology

To generate generalizable and reproducible results, we simulate the shifting strategies presented above with TEFlowSim using multiple real-world traffic traces and realistic split ratios from existing TE optimizers.

A. TEFlowSim

The TEFlowSim toolchain is summarized in Figure 4, and operates in two steps. First, the *FlowAnalyzer*[1] transforms a pcap trace into a flow-level trace where the start/end times, transmitted bytes (sum of total length fields of all IP packets belonging to the flow) and the mean throughput (bytes divided by duration) are stored for each flow. Second, *TEConvSim*[2] uses the flow-level trace to replay each flow with constant throughput. It simulates flow shifts over two paths using the shifting strategy, initial and updated split ratios and t_{update}. At a given update time t_{update}, TEConvSim applies a shifting strategy to reach the updated split ratio. Using TEConvSim, we measure n_{shift} and T_c.

1) TEFlowSim Scope: Because flow-level performance impacts depend on the shifting strategies and real-world traffic characteristics, our evaluation focuses on a single source-destination pair in a TE-controlled domain. Therefore, we do not apply a specific TE optimizer, but evaluate realistic split ratios (Sec. IV-C) with multiple t_{update} times (Sec. IV-D).

Why only two paths? We restrict our evaluation to two paths, because the amount of load that needs to be shifted is determined by the split ratio difference d rather than the number of paths. Having more paths merely requires allocating the shifted load to multiple paths (Sec. III-A) rather than one path, which is an orthogonal problem to our shifting strategies and is therefore out of scope. We discuss the impact of different numbers of paths on d in Section IV-C. In principle, TEConvSim could be extended to more than two paths.

Why flow-level simulations? Our simulation operates on flow-level granularity and simulates each flow with a constant throughput, because we are more interested in persistent load distribution changes rather than transient fluctuations. Such persistent changes are mainly influenced by flows that constantly transmit at high rates and persist over time compared to flows with negligible impact on total throughput. Although this flow-level granularity may smooth out some burstyness of the original trace, we found that it does not alter the traffic characteristics significantly. We compare the simulated flow-level throughput with the real throughput of the packet-level trace, which is calculated by dividing the sum of all transmitted bytes (total length in IP packet) in a certain time bin (we use 5 s) by the time bin duration. As shown in Figure 5, the throughput of our simulated output provides good approximates of the real throughput.

2) FlowAnalyzer Details: The FlowAnalyzer uses PCAP traces as input, extracts flow information and outputs a flow-level trace that can be used by TEConvSim. It analyzes each packet of the trace, associates the packet to the corresponding flow and assigns flow start/end timestamps based on packet timestamps and increments the per-flow number of transmitted bytes using the length of the IP packet (Total Length Field). Flows are identified by the 5-tuple and the timestamp of the first packet. TCP flows are required to start with a SYN packet, and end with a FIN packet. If the end time of the TCP flow exceeds the trace duration, we use the last seen packet of the 5-tuple in the trace. UDP flows that share the same 5-tuple, but have longer pauses in between transmissions are separated by an inactive threshold of 15 s, as done in [40]. When the pause time exceeds the inactive threshold, a new flow is created.

[1]https://gitlab.kit.edu/kit/tm/telematics/teconvsim

[2]https://gitlab.kit.edu/kit/tm/telematics/pcapanalyzer

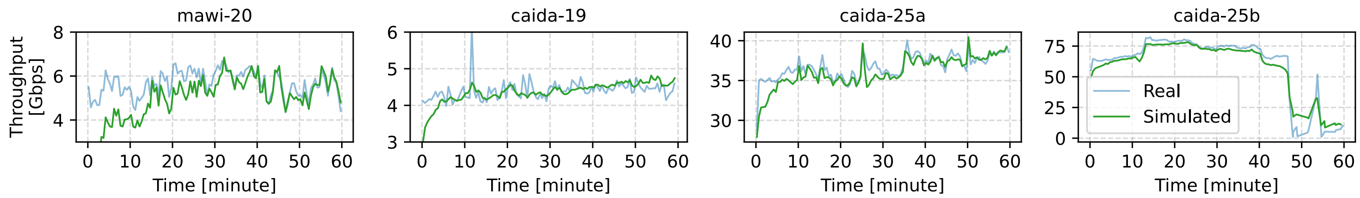

Fig. 5. Simulated flow-level TEConvSim throughput compared to real packet-level throughput.

TABLE I
TRAFFIC TRACES. ALL TRACES HAVE A DURATION OF 1H

Trace	Date	Link Cap. (Gbps)	Mean Rate (Gbps)	Max. Running Flows	Packets
Mawi-20 [21]	20/04/08	10	6.5	6.5e4	5.8e8
Caida-19 [22]	19/01/17	10	4.5	6.6e5	2.4e9
Caida-25a [23]	25/01/23	100	38	1.6e7	2.6e10
Caida-25b [23]	25/02/20	100	61	3.6e7	6.5e10

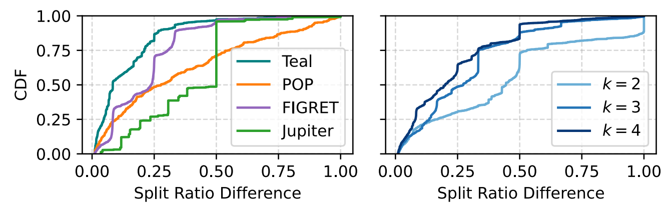

Fig. 6. GÉANT split ratio differences using four different TE optimizers and $n = 4$ paths (left) and differences for n paths over all TE optimizers (right)

The flow-level trace also includes an event list (i.e., flow start/flow end event) that can be directly used by TEConvSim. Creating this event list while processing the trace makes the next step (TEConvSim) more efficient, since using only the flow list would require sorting all flows by start/end times and keeping per-flow state during the simulation. We store the the flow/event lists using the compression-based parquet format [41] to reduce memory and accelerate read operations.

3) TEConvSim Details: TEConvSim uses the event list output by the FlowAnalyzer to run a flow-level simulation over two paths. For a start event, TEConvSim adds the flow to a set of running flows, assigns the flows to a path (according to the path assignment mechanism), and adds the flow's throughput to the path's total throughput. For an end event, TEConvSim removes the flow from the set of running flows, and subtracts the flow's throughput from the path's total throughput. TEConvSim calculates the load distribution for each event by dividing the total throughput by the path's throughput. Before t_{update}, flows are assigned to paths according to the initial split ratio. At t_{update}, split ratios are updated, causing the shifting strategy to move a fraction of traffic to the path with the increased split ratio. Starting with t_{update}, TEConvSim applies the chosen shifting strategy and measures n_{shift}. By comparing the load distribution with the split ratios, the state of convergence and T_c are measured.

B. Traffic Traces

TEFlowSim uses packet-level traces (summarized in Table I) recorded on different backbone links as input. For each trace type, we also ran more simulations with traces at different dates. For Mawi-20 and Caida-19, we omit the results for additional traces at different dates, because they were comparable with respect to mean throughput and flow characteristics. In contrast, the Caida-25 traces show more significant differences in throughput, which is why we report results for two traces with different characteristics (Caida-25a and Caida-25b). As our evaluations are based on traces recorded on three backbone links with different capacities and locations, we expect our results to be generalizable for Tier-1 backbone traffic.

C. Split Ratios

TEFlowSim relies on split ratios to simulate flow shifts. Split ratio differences d determine how much traffic volume is shifted. To obtain realistic values for d, we extract them from existing TE optimizers. We evaluate d for multiple TE optimizers using the GÉANT topology [8] and the corresponding real-world traffic matrices [42]. We chose the TE optimizers listed in Table II to include both ML- and LP-based TE optimizers and to include two optimization goals, i.e., minimizing the maximum link utilization (MLU) and maximizing total flow.

For each existing TE optimizer implementation in Table II, we exported the split ratios, allowing us to calculate d. Figure 6 (left) shows the distribution of d for $n = 4$ shortest paths per source destination pair in the GÉANT topology using four TE optimizers. For POP, 74% of d values are lower or equal to 0.5. For Teal, FIGRET, and Jupyter, d is lower or equal to 0.5 in 90% of the cases. While these three TE optimizers perform a global optimization algorithm, POP decomposes the global problem into multiple subproblems, solves them in parallel and merges them afterwards. Thus, the different views during optimization (global vs. local) may explain POP's deviation in d values. Finally, the split ratio differences (d) calculated by the TE optimizers most often are smaller than 0.5.

We verify whether increasing the number of paths would lead to other split ratio differences and measure d for $n = 2$, $n = 3$ and $n = 4$ paths per source-destination pair. Figure 6 (right) shows results for the different n values and the aggregated d values for all four TE optimizers. When increasing $n = 2$ to $n = 4$, the share of d values lower or equal to 0.5 increases from 70 to 90%. Intuitively, a higher number of

TABLE II
TE OPTIMIZERS USED TO EVALUATE SPLIT RATIOS

TE Optimizer	*Optimization Method*	*Objective*
Teal [9]	GNN + DNN + MARL	Max Total Flow
FIGRET [43]	DNN	Min. MLU
POP [44]	LP	Max Total Flow
Jupiter [45]	LP	Min. MLU

paths results in a lower d, because the amount of shifted traffic volume can be split onto more paths.

The results showed that in most cases, existing TE optimizers produce initial and updated split ratios with differences of up to 0.5. In TEFlowSim, we consider split-ratio changes up to 0.5 and simulate all combinations of initial and updated split ratios at 0.1 granularity (e.g., $[0.2, 0.8]$ rather than $[0.25, 0.75]$). With this discretization, we explore a finite, but wide range of possible split ratio combinations for each d.

D. Update Time t_{update}

We evaluate multiple t_{update} times for each trace because the traffic composition at the time of shift influences the shifting performance, e.g., a sudden bursty arrival of flows before the shift can increase the difference between load distribution and split ratios and can ultimately prolong convergence. As all traces have a duration of 1h (Table I) and we seek to leverage most of each trace's duration, we apply shifts first at minute 10, then at minute 20, and so on up to minute 50.

E. Convergence

In our experiments, we define convergence at t_{converge} as the earliest time when the load distribution remains within 0.05 of the updated split ratios for at least 1 s on average.

F. Example Run

An example run for the ZeroShift strategy, an initial split ratio of $[0.5, 0.5]$ and a updated split ratio of $[0.2, 0.8]$ for a 15-minute Mawi trace [46] is shown in Figure 7. Because all newly arriving flows are assigned to Path 1 at t_{update} (360 s), the upper figure (Fig. 7) shows that the throughput on the upper path increases. In contrast, the departure rate of flows that were already active before t_{update} gradually reduces throughput on Path 2. The figure shows slow convergence (around 240 s) for this split ratio combination when applying ZeroShift.

V. EVALUATION

We evaluate the different shifting strategies using flow-level simulation experiments (described in Section III-B). Specifically, our evaluation focuses on these questions:

1) How feasible is the ZeroShift strategy that does not shift any flows at all? (Section V-A)
2) How do few elephant flows influence convergence?
   a) Chat characterizes a flow as an elephant flow with an impact on convergence? (Section V-B1)
   b) To what extent do elephant flows block convergence? (Section V-B2)
3) Does SelectiveShift reduce the number of shifted flows and converge fast? (Section V-C)

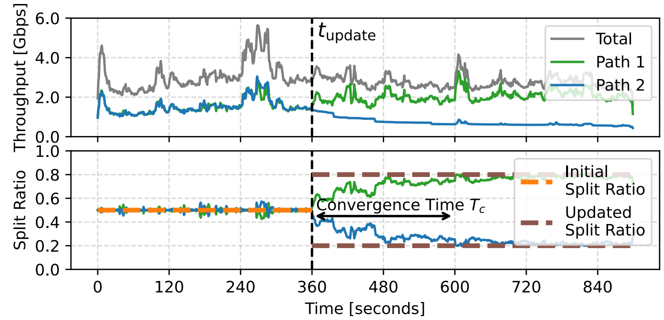

Fig. 7. Example Single-Run TEConvSim Output. Total datarate (top) and split ratios (bottom) measured running an example Mawi trace [46]. At minute 6, the split ratios are changed from 0.5/0.5 to 0.2/0.8 using ZeroShift.

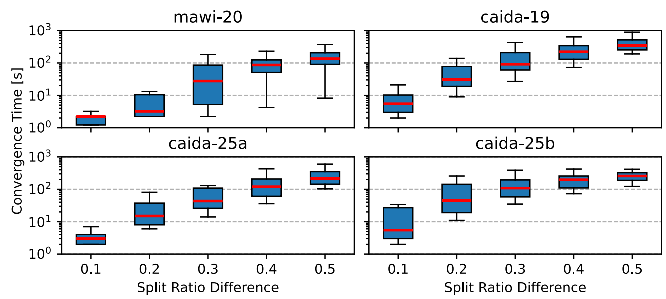

Fig. 8. Convergence times T_c in relation to split ratio difference for shifting strategy ZeroShift for four different traffic traces

A. ZeroShift Convergence

We first evaluate ZeroShift, that does not shift any flows even if split ratios change ($F_{\mathrm{shift}} = \emptyset$). With largest-gap-first path assignment, it uses newly arriving flows to reach the updated split ratio. This experiment seeks to determine T_c for ZeroShift. Figure 8 shows T_c for each split ratio difference up to 0.5 (as motivated in Sec. IV-C) and each trace. Intuitively, larger split ratios imply longer convergence times, since more flows are required to shift a larger share of bandwidth. While for the lowest split ratio difference of 0.1, the median convergence time is always less than 10 s for all traces, in the worst case convergence can take up to 14 minutes. This does not align with short TE intervals of up to 10 s, as the load distribution does not converge in time before TE provides the next split ratios.

Interestingly, T_c significantly differs between the traces. Mawi-20 has shorter median T_c values compared to all other traces. This is due to the flow statistics (Fig. 9): individual flows in Mawi-20 have higher throughput than in the other traces. The higher throughput for individual flows can be explained by larger byte-/and packet-wise flow sizes with shorter flow durations of Mawi-20 compared to the other traces. As a result, each newly arriving flow contributes more bandwidth, can fill up the to-be-increased path faster, thus reducing T_c. Differences also appear between two traces on the same 100 Gbps link, where larger variations in mean throughput across traces are more common: For $d = 0.2$,

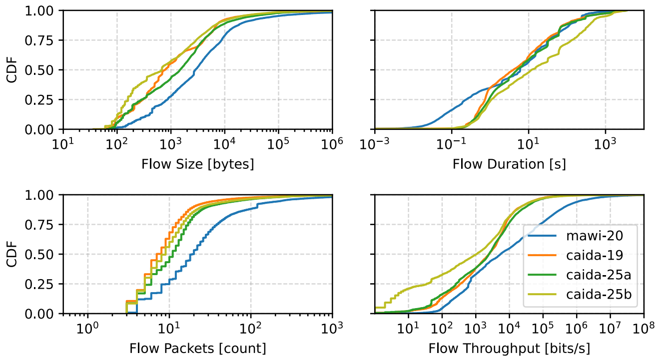

Fig. 9. Flow size, duration, packets and throughput distributions for four different traffic traces.

Caida-25a results in $T_c = 15\,s$ compared to $T_c = 45\,s$ for Caida-25b. Caida-25b has a higher mean throughput of 61 Gbits/s compared to Caida-25a, and also a higher number of concurrently running flows (Table I). Therefore, the additional throughput for Caida-25b is distributed across more flows, so individual flows receive less bandwidth on average. Ultimately, the ZeroShift strategy that relies on newly arriving flows to reach the updated split ratios often results in convergence times of more than a minute. Consequently, ZeroShift most often cannot accommodate short TE intervals.

B. Impact of Elephant Flows on Convergence

In this section, we analyze long-running and high throughput *elephant flows* that contribute significantly to total throughput and may have a significant impact on convergence times.

1) Elephant Flow Characteristics: For convergence, two flow properties are relevant to distinguish elephant flows from other flows: High throughput (leverage heavy tail of flow throughput distribution, see Figure 9) and long duration (persistent heavy flows). We consider the number of sent packets and the total transmitted bytes as candidate metrics. Consequently, any flow that transmits more than x bytes/packets is labeled as elephant flow. We then assess how well these candidate metrics discriminate elephant flows from the remaining traffic.

Figure 10 depicts 20k sampled flows for the Mawi-20 trace with multiple byte (left) and packet (right) thresholds. The most significant elephants approach the upper right corner of each plot, as these are the elephant flows with a long duration and a high throughput. Using flow size thresholds (bytes) identifies these elephants sufficiently, while the packet counter thresholds include flows with shorter durations and lower throughput at times. For instance, a flow with relatively few packets can still have a high average throughput if a burst of packets arrives in quick succession, inflating its mean throughput without indicating sustained high volume; conversely, a flow with many small packets may not achieve high throughput. Results for other traces (omitted for brevity) also confirm that a flow's transmitted bytes are a good metric for extracting high-throughput, long-running elephant flows.

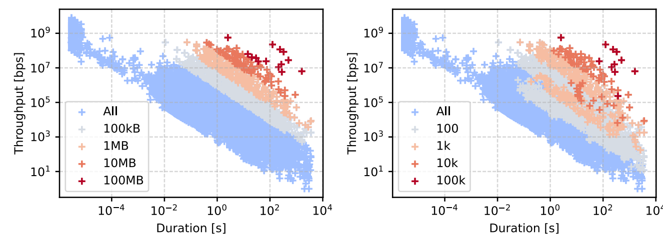

Fig. 10. Throughput and duration of flows with different elephant thresholds for Mawi-20. Left: Size (bytes) Threshold, Right: Packet count threshold

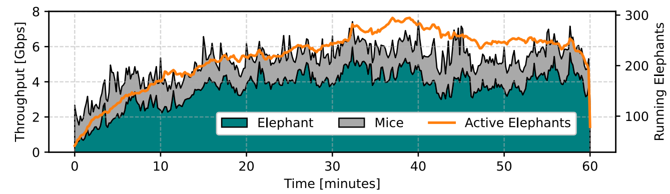

Fig. 11. Bandwidth share of elephant flows and running elephants (here: flow size threshold of 100 MB) for Mawi-20

We are now interested in the bandwidth share of the set of elephants and the number of concurrently running elephants for different threshold values. The total bandwidth share of elephants allows estimating the fraction of bandwidth blocked on a certain path, which may prolong convergence times for ZeroShift. Figure 11 shows the stacked throughput of elephant (100 MB threshold) and mice flows for Mawi-20 [21]. Here, running elephants are measured by slicing the trace into 5 s time bins and counting the number of elephants for each time bin. The plot confirms that for this trace, up to 300 running elephants (i.e., on average, less than 0.4% of all running flows), contribute to 66% of total bandwidth. Figure 12 shows the bandwidth share and the number of running elephants for multiple traces and thresholds. Higher elephant thresholds reduce the bandwidth share and also the number of elephants. Also, the thresholds can result in substantially different bandwidth shares for different traces. For example, the 100 MB threshold captures a mean 65% bandwidth share in Mawi-20, but only 38% for Caida-19. Also, the number of running elephants is much higher in the Caida traces, which also have a higher overall number of running flows (as shown in Table I).

We now check our assumption that the elephant bandwidth share remains stable over time. An elephant bandwidth share

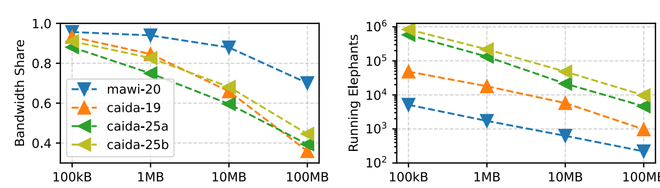

Fig. 12. Mean bandwidth share of elephant flows (right) and mean concurrently running elephants for different thresholds and traces (left)

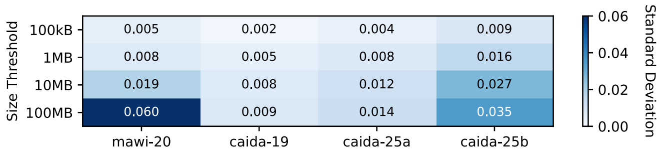

Fig. 13. Standard deviation of elephant bandwidth share for all traces

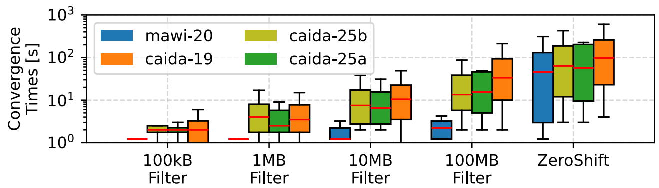

Fig. 14. Convergence times for ZeroShift measured on filtered runs that exclude elephant flows beginning before t_{update}.

estimation is required for SelectiveShift to determine the fraction of shifted elephant flows. A more unstable bandwidth share would result in longer convergence times for SelectiveShift, as either too many or too few elephants would be shifted. We slice the trace into 5 s bins and measure the elephant bandwidth share by dividing the total bandwidth by the elephant bandwidth in each bin. Figure 13 shows the standard deviation of the elephant bandwidth shares for all traces. All values are smaller than 0.06, indicating a stable elephant bandwidth share. Lower thresholds result in a more stable bandwidth share due to statistical multiplexing given the higher number of elephant flows. Higher elephant thresholds result in a more unstable elephant bandwidth share (e.g., 0.06 for 100 MB, Mawi-20). This is because fewer flows are in the set of elephants, e.g., 220 flows for 100 MB compared to 5124 flows for 100 kB (as shown in Fig. 12 right). Fewer elephant flows result in higher fluctuations, as each flow has a higher share of elephant bandwidth.

Due to significant differences between the traces, thresholds must be carefully chosen depending on the traffic characteristics and the targeted elephant bandwidth share. Also, the elephant bandwidth share is relatively stable and fluctuates by less than 6 percentage points. As these elephants contribute to up to 96% of bandwidth, they most likely impact convergence times significantly.

2) Elephant Flows Block Convergence: In this section, we confirm that a subset of elephant flows that were running before t_{update} block a fraction of bandwidth on the to-be-increased path, thus prolonging convergence times. For ZeroShift, we have shown that shifting elephants starting after t_{update} (together with all other flows starting after t_{update}) results in long convergence times. To determine the impact of elephants running before t_{update} that may block load on this path, we filter these out to determine how much shorter the convergence time would have been without them.

Figure 14 shows that filtering running elephants reduces convergence times for all evaluated traces. For Mawi-20 and all elephant thresholds, convergence times are reduced to less than 10 seconds. For Caida-25b, filtering running elephants reduces median convergence times for all thresholds to less than 14 seconds. As a result, we have identified elephant flows running before t_{update} as main convergence blockers. Therefore, shifting these elephant flows has the potential to significantly reduce convergence times.

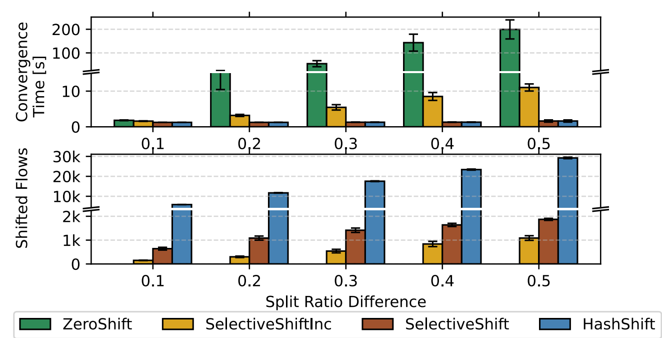

Fig. 15. Convergence times (top) and shifted flows (bottom) for Mawi-20 and all shifting strategies. An elephant threshold of 1 MB was used. SelectiveShiftInc shifts 10 elephants every 100 ms.

C. SelectiveShift: Shift Elephants

This section explores the possibilities of shifting only elephant flows while all other flows are never shifted, and compares them with the other shifting strategies. Figure 15 shows T_c and n_{shift} of different shifting strategies for the Mawi-20 trace and an elephant threshold of 1 MB. SelectiveShiftInc was configured to shift $k = 10$ elephant flows every $f^{-1} = 100$ ms. As expected, there are two extremes. First, ZeroShift can lead to mean T_c values of up to 291 seconds, thus not fulfilling short TE intervals. Second, HashShift converges fast but shifts more than 29k flows (from up to 65k running flows) for the split ratio difference of 0.5. In contrast, SelectiveShift and SelectiveShiftInc achieve not only fast convergence but drastically reduce the number of shifted flows by up to 39× compared to HashShift (For $d = 0.1$, HashShift shifts 5853 flows, while SelectiveShiftInc shifts only 151 flows on average). SelectiveShiftInc can be used to control the tradeoff between T_c and n_{shift} more precisely by adjusting f and k.

Figure 16 shows T_c and n_{shift} for all traces. It shows a clear trend: Compared to HashShift, SelectiveShift reduces the number of shifted flows significantly while providing the same convergence times. SelectiveShiftInc reduces the median number of shifted flows by 43–45 times, i.e., for Caida-25a from 3.2 million to only 72.5k shifted flows.

Overall, SelectiveShift and SelectiveShiftInc drastically reduce n_{shift} while maintaining T_c, compared to HashShift. Therefore, shifting high-throughput and long-running elephant flows avoids disruption for most flows by keeping their paths persistent. These strategies are especially suitable with short TE intervals, as they minimize disruption while quickly re-aligning the load according to the split ratios.

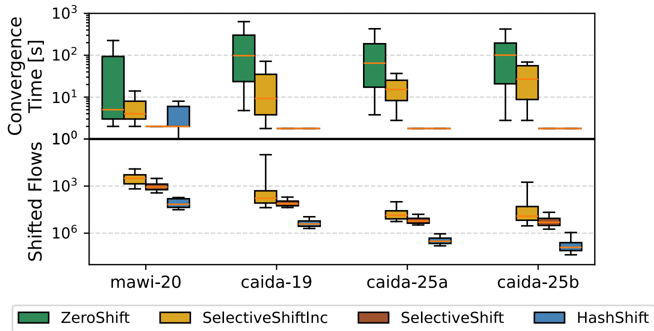

Fig. 16. Convergence times (top) and shifted flows (bottom). Elephant thresholds of 100 kB were used. SelectiveShiftInc shifts 10/50/100/500 flows every 100 ms for each traffic trace.

VI. Conclusion

This paper investigates flow shifting strategies to reduce the overall impact of flow shifts in the network. Because the performance of these shifting strategies is linked to the traffic characteristics, we evaluate different shifting strategies on multiple real-world traffic traces. A strategy that avoids disruption by using newly arriving flows for reaching the load distribution is not compatible with short TE intervals (up to 10 s). We identify high-throughput, long-running elephant flows running before the shift as main blockers for convergence and propose to shift only these flows while keeping the paths of all other flows persistent. This strategy reduces the total number of shifted flows by up to 45× compared to existing hash-based strategies while achieving similarly short convergence times.

References

[1] "Equinix Data Centers," Nov. 2025, [Online] Available: https://www.equinix.com/data-centers.

[2] C.-Y. Hong, S. Kandula *et al.*, "Achieving high utilization with software-driven WAN," in *SIGCOMM 2013*. ACM, 8 2013.

[3] S. Jain, A. Kumar *et al.*, "B4: Experience with a Globally-Deployed Software Defined WAN," in *SIGCOMM 2013*. ACM, 8 2013.

[4] Z. Wang, Z. Li *et al.*, "Large-Scale Measurements and Prediction of DC-WAN Traffic," *IEEE Transactions on Parallel and Distributed Systems*, vol. 34, 5 2023.

[5] M. Denis, Y. Yao *et al.*, "EBB: Reliable and Evolvable Express Backbone Network in Meta," in *SIGCOMM 2023*. ACM, 9 2023.

[6] C. Tang, "Meta's Hyperscale Infrastructure: Overview and Insights," *Communications of the ACM*, vol. 68, 2 2025.

[7] T. Schueller and M. Horneffer, "Advancing DT's Traffic Engineering for SRv6," SRv6OPS WG, Jul. 2025.

[8] "GÉANT," Nov. 2025, [Online] Available: https://geant.org/.

[9] Z. Xu, F. Y. Yan *et al.*, "Teal: Learning-Accelerated Optimization of WAN Traffic Engineering," in *SIGCOMM 2023*. ACM, 9 2023.

[10] Y. Perry, F. V. Frujeri *et al.*, *DOTE: Rethinking (Predictive) WAN Traffic Engineering*. USENIX Association, 2023.

[11] U. Krishnaswamy *et al.*, "Decentralized cloud wide-area network traffic engineering with BLASTSHIELD," in *NSDI 22*. Usenix, 4 2022.

[12] U. Ranadive and D. Medhi, "Some observations on the effect of route fluctuation and network link failure on TCP," in *ICCCN*, 10 2001.

[13] T. Zhang, Y. Lei *et al.*, "Fine-grained load balancing with traffic-aware rerouting in datacenter networks," *Journal of Cloud Computing*, vol. 10, 12 2021.

[14] R. Cârpa *et al.*, "Evaluating the impact of SDN-induced frequent route changes on TCP flows," in *2017 13th International Conference on Network and Service Management (CNSM)*. IEEE, 11 2017.

[15] J. Karlsson, P. Hurtig *et al.*, "Impact of multi-path routing on TCP performance," in *WoWMoM*, 6 2012.

[16] A. Zapletal and F. Kuipers, "Slowdown as a Metric for Congestion Control Fairness," in *Proc. the 22nd ACM Workshop on Hot Topics in Networks*. New York, NY, USA: ACM, 11 2023.

[17] T. Benson, A. Anand *et al.*, "MicroTE," in *CoNEXT*. ACM, 12 2011.

[18] W. Reda, K. Bogdanov *et al.*, "Path persistence in the cloud," *ACM SIGCOMM CCR*, vol. 50, 5 2020.

[19] F. Gui, S. Wang *et al.*, "RedTE: Mitigating Subsecond Traffic Bursts with Real-time and Distributed Traffic Engineering," in *SIGCOMM 2024*. ACM, 8 2024.

[20] A. A. AlQiam, Y. Yao *et al.*, "Transferable Neural WAN TE for Changing Topologies," in *SIGCOMM 2024*. ACM, 8 2024.

[21] MAWI Working Group, "Packet traces from WIDE backbone, samplepoint G, 2020/04/08," https://mawi.wide.ad.jp/mawi/ditl/ditl2020-G/, 2020, accessed: 2025-06-30.

[22] CAIDA, "The CAIDA UCSD Anonymized Internet Traces - 2019/01/17," https://www.caida.org/catalog/datasets/passive_dataset, 2019, accessed: 2025-06-30.

[23] CAIDA, "Anonymized Two-Way Traffic Packet Header Traces - 2025/01/23, 2025/02/20," https://www.caida.org/catalog/datasets/passive_100g_dataset, 2025, accessed: 2025-12-21.

[24] J. Zhang, Z. Guo *et al.*, "SmartEntry," in *Proc. the Workshop on Network Meets AI & ML*. ACM, 8 2020.

[25] M. Ye, J. Zhang *et al.*, "FlexDATE: Flexible and Disturbance-Aware Traffic Engineering With Reinforcement Learning in Software-Defined Networks," *IEEE/ACM Transactions on Networking*, vol. 31, 8 2023.

[26] S. Kandula, D. Katabi *et al.*, "Walking the tightrope," *ACM SIGCOMM CCR*, vol. 35, 10 2005.

[27] S. Fischer, N. Kammenhuber *et al.*, "REPLEX," in *Proc. the 2006 ACM CoNEXT conference on - CoNEXT '06*. ACM Press, 12 2006.

[28] Z. Cao *et al.*, "Performance of hashing-based schemes for Internet load balancing," in *Proceedings IEEE INFOCOM 2000*. IEEE, 3 2000.

[29] S. Prabhavat, H. Nishiyama *et al.*, "On Load Distribution over Multipath Networks," *IEEE Communications Surveys & Tutorials*, vol. 14, 9 2012.

[30] D. Tuncer, M. Charalambides *et al.*, "Flexible Traffic Splitting in OpenFlow Networks," *IEEE Transactions on Network and Service Management*, vol. 13, 9 2016.

[31] K.-F. Hsu, P. Tammana *et al.*, "Adaptive Weighted Traffic Splitting in Programmable Data Planes," in *Proc. the Symposium on SDN Research*. New York, NY, USA: Association for Computing Machinery, 3 2020.

[32] N. Kang, M. Ghobadi *et al.*, "Efficient traffic splitting on commodity switches," in *PACMNET*. New York, NY, USA: ACM, 12 2015.

[33] Y. Sadeh, O. Rottenstreich *et al.*, "Optimal weighted load balancing in tcams," *IEEE/ACM Transactions on Networking*, vol. 30, 6 2022.

[34] S. Gay, P. Schaus *et al.*, "REPETITA: repeatable experiments for performance evaluation of traffic-engineering algorithms," *CoRR*, vol. abs/1710.08665, 10 2017.

[35] P. Kumar, C. Yu *et al.*, "Yates: Rapid prototyping for traffic engineering systems," in *SOSR '18*. ACM, 3 2018.

[36] G. F. Riley and T. R. Henderson, *The ns-3 Network Simulator*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 15–34.

[37] F. Abuzaid, S. Kandula *et al.*, "Contracting Wide-area Network Topologies to Solve Flow Problems Quickly," in *NSDI 21*. USENIX Association, 4 2021.

[38] R. Azorin, A. Monterubbiano *et al.*, "Taming the Elephants: Affordable Flow Length Prediction in the Data Plane," *Proc. the ACM on Networking*, vol. 2, 3 2024.

[39] J. Moraney and D. Raz, "On the Practical Detection of Hierarchical Heavy Hitters," in *2020 IFIP Networking*, 6 2020, pp. 37–45.

[40] B. Claise (Ed.), B. Trammell (Ed.) *et al.*, "Specification of the IP Flow Information Export (IPFIX) Protocol for the Exchange of Flow Information," RFC 7011 (Internet Standard), IETF, Sep. 2013.

[41] Apache, "Parquet," 2026, [Online] Available: https://parquet.apache.org.

[42] S. Uhlig, B. Quoitin *et al.*, "Providing public intradomain traffic matrices to the research community," *SIGCOMM CCR*, vol. 36, Jan. 2006.

[43] X. Liu, S. Zhao *et al.*, "FIGRET: Fine-Grained Robustness-Enhanced Traffic Engineering," in *SIGCOMM 2024*. ACM, 8 2024.

[44] D. Narayanan *et al.*, "Solving Large-Scale Granular Resource Allocation Problems Efficiently with POP," in *SOSP 2021*. ACM, 10 2021.

[45] L. Poutievski, O. Mashayekhi *et al.*, "Jupiter evolving: transforming google's datacenter network via optical circuit switches and software-defined networking," in *SIGCOMM 2022*. ACM, 8 2022.

[46] MAWI Working Group, "Packet traces from WIDE backbone, samplepoint G, 2020/06/10," https://mawi.wide.ad.jp/mawi/samplepoint-G/2020/202006101400.html, 2020, accessed: 2025-06-30.