AccelToR: Accelerating TCP for Circuit/Packet Hybrid Data Centers with Packet Scheduling

Wei Chen, Ye Tian, *Member, IEEE,* and Xinming Zhang, *Senior Member, IEEE*

***Abstract*—To overcome the inherent limitation of the optical circuit switch (OCS) while utilizing its high bandwidth, circuit/packet hybrid networks are widely proposed for modern data centers. However, as today's OCS has reduced the reconfiguration delay to microseconds, the circuit from a source rack to a destination rack typically lasts fewer than 10 RTTs. Such a short circuit time brings a critical challenge to TCP, as it is difficult for a TCP sender to sufficiently grow its congestion window (CWND) and utilize the optical bandwidth. To address this problem, in this work, we present *AccelToR*, a top-of-rack switch for improving TCP performance in circuit/packet hybrid data center networks. AccelToR leverages end-host congestion control to "accelerate" a blocked TCP flow by temporarily scheduling its packets to be transferred through the packet network a few RTTs before its circuit is established, and after enlarging the flow's CWND with the acceleration, the switch buffers the last window of packets. During the circuit time, the switch sends out the buffered packets, and the accelerated flows, which have their CWNDs already grown large, continue to send packets at high rates to achieve a high optical bandwidth utilization. Experiment results show that AccelToR achieves high throughputs for elephant flows and utilizes over 90% of the optical bandwidth, and it preserves short flow completion times for mice flows at the same time. In addition, AccelToR is robust under unexpected packet losses, and can benefit a wide range of TCP congestion control algorithms.**

***Index Terms*—Optical circuit switch (OCS), circuit/packet hybrid data center network, TCP congestion control, packet scheduling**

I. Introduction

With the approach of the end of Moore's law, packet-switched networks face a challenge that for CMOS-based electrical packet switches (EPSes), cost and power consumptions increase at faster rates than switching capacity [1]. Meanwhile, emerging giant AI models such as ChatGPT [2] require massive bandwidth, and are facing a "bandwidth wall" when running on computing clusters inter-connected by conventional packet-switched networks [3], [4].

To provide higher bandwidth, optical technologies seem to be a promising direction, and during the past decade, innovative designs that introduce optical circuit switches (OCSes) to the data center network architecture have been proposed and practiced [5]–[20]. Compared with EPS, OCS has several merits: 1) OCS is agnostic to data rate, thus does not need to be upgraded when the network evolves to a higher data rate; 2) OCS has a much lower per-bit power consumption as it is bufferless and passively handles photons without per-packet processing; 3) OCS-based network is inexpensive as there is no need to use transceivers for conducting the expensive optical-electrical-optical conversions in the network core.

One major drawback of OCS is that the switch can only provide point-to-point connectivity between its ingress and egress ports at a time, and reconfiguring the connectivity among the ports (called a *matching*) incurs a non-trivial delay, during which the switch can not handle any traffic. To overcome this limitation, a natural approach is to have a circuit/packet hybrid network, in which the OCS-based network provides high bandwidth and an EPS-based network provides continuous connectivity in parallel [5], [6], [8], [9].

To reconfigure OCS frequently while achieving a circuit uptime ratio above 90%, the interval between two consecutive configurations is typically a few to a dozen times of the reconfiguration delay. Meanwhile, modern OCSes can be reconfigured in microseconds, which means that in today's data centers with microsecond-scale propagation delays [21], the optical circuit from a source rack to a destination rack typically lasts fewer than 10 RTTs [22]. Such a short circuit time brings a critical challenge to TCP, as it is difficult for a TCP sender, which may be either recovering from packet losses due to the previous OCS reconfiguration, or just switched from the low-bandwidth packet network, to sufficiently grow its congestion window (CWND). As a result, TCP flows in the hybrid network are unable to efficiently utilize the optical bandwidth and achieve high throughputs [22], [23].

Previous works addressing this problem focus on end-hosts. For example, Mukerjee et al. [22] suggest that a TCP sender at end-host increases CWND on receiving an explicit circuit state feedback from the top-of-rack (ToR) switch, and Chen et al. [23] develop a new TCP variant named Time-division TCP (TDTCP) that maintains multiple sets of independent congestion control states for the low-bandwidth packet network and the high-bandwidth circuit network separately. However, in these solutions, all network flows are transferred through the packet network when the corresponding optical circuits are unavailable. As a consequence, congestions are brought by the bandwidth-hungry elephant flows to the packet network, and greatly impact the delay-sensitive mice flows. Moreover, the existing solutions [22], [23] require end-hosts to coordinate closely with the circuit network, thus impose considerable complexity by substantially modifying the OS protocol stack.

In this paper, we present *AccelToR*, a novel ToR switch for improving TCP performance in circuit/packet hybrid data center networks. As in many hybrid networks [5], [9], [24], [25], AccelToR segregates TCP traffic into elephant and mice

This work was supported in part by the National Natural Science Foundation of China (NSFC) under Grant 61672486 and Grant 62072425. *(Corresponding author: Ye Tian.)*

The authors are with Anhui Key Laboratory on High-Performance Computing, School of Computer Science and Technology, University of Science and Technology of China, Hefei, Anhui, China, 230026.

flows, and transfers them over the circuit and packet networks separately. The benefit is that by restricting elephant flows from the packet network, mice flows can preserve their short flow completion times (FCTs). The challenge is that within a circuit time shorter than 10 RTTs, elephant flows do not have enough time to grow their CWNDs and sufficiently utilize the optical bandwidth.

For boosting performances of the elephant flows without sacrificing the mice flows, the AccelToR switch "accelerates" the elephant flows by scheduling their packets to be transferred through the packet network a few RTTs before their circuits to the destinations are established, and buffers the last windows of their packets. With the acceleration, the elephant flows can enlarge their CWNDs ahead of time. During the circuit time, the switch sends out the buffered packets to fill the capacity of the optical link, and the elephant flows, which have their CWNDs already enlarged, continue to send packets at high rates to sufficiently utilize the optical bandwidth.

AccelToR has the following desired properties. 1) AccelToR enables bandwidth-hungry elephant flows to achieve high throughputs over the hybrid network and sufficiently utilize the optical bandwidth. 2) AccelToR preserves short FCTs for delay-sensitive mice flows without bringing congestions to the packet network. 3) AccelToR is scalable to network size with a constant and moderate memory overhead for packet buffering, thus is practical with today's switch hardware. 4) AccelToR does not require any explicit coordination from end-hosts, thus is easy to deploy. 5) AccelToR is widely applicable as it benefits a wide range of TCP variants using different congestion control algorithms (CCAs). In the design, development, and evaluation of AccelToR, we have made the following contributions.

- **Analysis**. We make an insightful analysis on the challenges and design space of the TCP data transportation in a circuit/packet hybrid data center network, and show that for bandwidth-hungry elephant flows, when their source-destination circuits are not available, there is a dilemma on whether to block the flows or transfer them through the packet network.
- **Design**. We propose a novel method for "accelerating" the elephant TCP flows. Our method is centered around the AccelToR switch, in which we employ the disciplines of *priority queue* and *calendar queue* for scheduling packets of different TCP flows. In particular, we formulate the TCP flow accelerating problem as an integer optimization problem, and design a packet scheduling algorithm based on the problem solution.
- **Evaluation**. We carry out extensive packet-level simulations to evaluate AccelToR and compare it with alternative solutions. The experiment results show that AccelToR significantly increases elephant flows' throughputs and utilizes over 90% of the optical bandwidth, and it preserves short FCTs for mice flows at the same time. In addition, AccelToR is robust against congestions and unexpected packet losses, and benefits a wide range of TCP CCAs.

The remainder part of this paper is organized as follows.

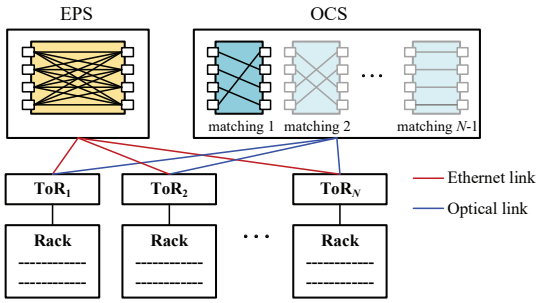


Fig. 1. Model of a circuit/packet hybrid data center network composed of N racks inter-connected by a giant EPS and an OCS in parallel.

We introduce the background and analyze the challenges that motivate this work in Sec. II; Sec. III presents the detailed design of AccelToR; We evaluate AccelToR in Sec. IV; Sec. V discusses the related works and we conclude in Sec. VI.

II. Background and Motivation

A. Optical Circuit Switching in Data Center

Optical switched networking is a promising direction for data center networks. Compared with the network based on legacy EPS, OCS is agnostic to data rate, and can provide very high switching capacity at a lower per-bit cost and power consumption.

Both EPS and OCS establish a *matching* between their respective ingress and egress ports. While an EPS can reconfigure its matching instantaneously, it takes time for an OCS to change its current matching to a new one, and during the reconfiguration, the switch can not carry any traffic. OCS reconfiguration delay varies from milliseconds to microseconds, depending on the adopted optical switching technology, e.g., 2D/3D MEMS [5], [7], [8], [10], [12], [16], AWGR [11], [15], free-space optics mirror assembly [26], [27], etc.

OCS matchings can be configured in either a *demand-aware* or a *demand-agnostic* manner. In a demand-aware way [5], [7]–[9], an OCS controller first estimates the communication demands among the racks, then it computes a series of matchings for satisfying the demands [24], [25], [28], [29]. A major drawback of this approach is the latency incurred for demand estimating and matching computation at the centralized controller, as for OCSes with microsecond-scale reconfiguration delays, such a latency is too high.

Recently, people propose to configure OCS in a demand-agnostic manner [12], [14]. In such an approach, the OCS locally cycles through a series of static matchings in a round-robin way, regardless of the inter-rack communication demands. The benefit is that the time-consuming centralized computation is avoided, and network complexity is reduced. In particular, since OCS matchings are static and recurrent, scheduling can be planned ahead rather than ad-hoc.

B. Circuit/Packet Hybrid Network Model

To overcome the inherent limitation of the OCS-based network while preserving its desired properties, people propose to develop circuit/pacekt hybrid networks for data centers [5], [6],

[8], [9], [22], [23]. In a hybrid network, a separate electrical packet network runs in parallel with an optical circuit network. Typically, the packet network has a much lower bandwidth, but has a negligible reconfiguration delay.

In this work, we consider a circuit/packet hybrid data center network model similar to the ones in [5], [6], [9], [24], [25]. As demonstrated in Fig. 1, the network contains N racks, where each rack connects to the rest part of the network via a ToR switch. In particular, the N ToR switches are inter-connected by both an electrical packet switched network and an optical circuit switched network. For simplicity, in our model, we view the entire packet network as one giant EPS, and assume that one OCS connects to all the ToR switches[1]. Each ToR switch has an uplink connecting to the giant EPS and an uplink connecting to the OCS separately.

The OCS realizes an $N \times N$-connectivity among the N racks over time. After being configured with a matching, which decides how the N ingress ports are connected to the N egress ports, the OCS maintains the matching for a period of time, which is called a *day*, then it takes an interval of time, which is called a *night*, to be reconfigured to the next matching that decides a new connectivity. To achieve a high reconfiguration frequency and a high uptime ratio, the duration of a day is typically a few to a dozen times of a night.

The OCS cycles through a series of $N-1$ static matchings in a demand-agnostic manner to enforce an $N \times N$-connectivity over time [12], and such a circle is referred to as a *week*. For example, if a day lasts $240\,\mu s$ and a night lasts $20\,\mu s$, then for a network containing $N = 50$ racks, a week is composed of 49 days and nights, and has a duration of $12.74\,\mathrm{ms}$.

C. TCP Data Transportation in Hybrid Data Center Network

TCP is heavily used in data centers [30], and studies [31], [32] show that sizes and durations of the TCP flows in a data center are highly skewed: there are a few long-lived and heavy-traffic flows, which are referred to as *elephant flows*, while the other *mice flows* are small in size and short in duration. Generally, elephant flows are used for bulk data transportation such as migrating virtual machines and replicating video contents, although these applications are bandwidth hungry, they are not sensitive to latencies. On the other hand, mice flows are generated by delay-sensitive applications like web services and online gaming, and for these applications, the flow completion time (FCT) is the key. Given microsecond-scale propagation delays widely observed in modern data centers [21], retransmission caused by congestion is the major reason behind the long FCTs of the delay-sensitive mice flows.

Many hybrid data center networks choose to transfer elephant flows through the high-bandwidth circuit network, and send mice flows via the low-bandwidth packet network [5], [6], [8], [9]. The benefit is that by restricting elephant flows from the packet network, congestions can be mitigated. However, transferring elephant flows only through the circuit network suggests that when the source-destination circuit is unavailable, the flows will be blocked. To avoid packet losses due to the blocking, the source rack is supposed to maintain many queues, one for each destination, to buffer all the on-the-fly packets after the corresponding flows are blocked [8], [20].

However, buffering packets with per-destination queues raises a scalability concern, as the more racks a network contains, the larger packet buffer each rack needs to maintain. For example, suppose the optical bandwidth is $400\,\mathrm{Gbit/s}$ and RTT of the circuit network is $60\,\mu s$, then the bandwidth-delay product (BDP) is $3\,\mathrm{MB}$, which means that the source rack needs to buffer $3\,\mathrm{MB}$ of packet data in a queue dedicated to each destination rack. For a network containing 50 racks, the source rack must buffer as large as $147\,\mathrm{MB}$ of packet data.

Unfortunately, for state-of-the-art hardware switches, in-switch packet buffer is small in size and expensive. For example, the Xilinx Virtex-II Pro NetFPGA has only $10\,\mathrm{MB}$ of SRAM [33], and the Intel Tofino chip has $20\,\mathrm{MB}$ of packet buffer, which is shared by up to 32 ports [34]. Clearly, with the limited memory resource, it is impractical for a ToR switch to maintain per-destination queues to buffer packets for all the blocked elephant flows.

Instead of blocking elephant flows, recent studies [22], [23] propose to transfer elephant flows through the packet network when their optical circuits are unavailable. However, since a source-destination circuit lasts only one day in a week, for most time, elephant flows are competing bandwidth with mice flows in the packet network, and greatly prolong the latters' FCTs by bringing congestions to the packet network.

Moreover, a recent study [22] shows that as modern OCSes can be reconfigured in microseconds, a day typically lasts fewer than 10 RTTs. In such a hybrid network, when a congested elephant flow is switched from the low-bandwidth packet network to the high-bandwidth circuit network, it does not have enough time to grow its CWND, thus is unable to sufficiently utilize the optical bandwidth. Previous study [22] and our experiment in Sec. IV show that considerable optical bandwidth is unused in this case.

In summary, as today's OCSes have reduced the reconfiguration delay to microseconds, when the source-destination circuits for elephant TCP flows are unavailable, we are facing a *dilemma* on whether to block the flows or transfer them through the packet network. By blocking elephant flows, a rack needs to maintain per-destination queues with a total size proportional to the size of the network, which is unscalable and impractical with today's switch hardware. On the other hand, by transferring elephant flows through the packet network, FCTs of the mice flows will be prolonged due to the congestions brought by the elephant flows, and the optical bandwidth is under-utilized.

III. Accelerating TCP with AccelToR

A. Objective and Challenge

In this work, we present a novel ToR switch named *AccelToR* for improving TCP performance in a circuit/packet hybrid data center network. With AccelToR, we pursue the following objectives.

- **Objective 1**: We aim to achieve high throughputs for elephant flows and efficiently utilize the optical bandwidth.

[1]Commodity OCSes have radices on the order of 100 ports.

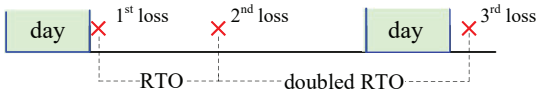


Fig. 2. `RTO` doubled after successive retransmission timeouts.

- **Objective 2**: We seek to preserve short FCTs for mice flows.
- **Objective 3**: AccelToR should be practical with today's switch hardware by being scalable. In particular, the memory overhead for queueing and buffering packets should be small and irrelevant to network size.

To achieve these objectives, we make the following design choices. First, AccelToR blocks elephant flows when their source-destination circuits are unavailable. By blocking elephant flows, we avoid congesting the packet network, and accomplish **Objective 2**. Second, AccelToR does not maintain per-destination queues for buffering packets of the blocked network flows. Without maintaining per-destination queues, we realize **Objective 3** regarding the scalability.

However, with the above design choices, it is challenging to fulfill **Objective 1**, which pursues high elephant flow throughput and optical bandwidth utilization. This is because without per-destination queues, when an elephant flow has its packets lost after being blocked, it will spend a time interval called retransmission timeout (`RTO`) to confirm the loss, and reduce its CWND to the minimal value (i.e., 1 MSS); furthermore, since the flow is eligible to send packets in only one day per week, it is very likely that the flow will encounter successive losses when trying to retransmit the lost packets, as illustrated in Fig. 2. Note that after each retransmission timeout, the sender's `RTO` is doubled, and its `ssthresh`, which decides the upper bound of the CWND in the next slow-start phase, is set as half of its current window size. Recall that for state-of-the-art OCSes, a day lasts fewer than 10 RTTs, therefore it is difficult for an elephant flow, which may either wait for the retransmission timer to expire or has a minimum-sized CWND, to sufficiently grow its CWND and utilize the optical bandwidth.

To overcome this problem, AccelToR chooses to "accelerate" the blocked elephant flows before their optical circuits are about to be established. The key idea is to schedule the blocked elephant flows to resume sending packets through the packet network a few RTTs before their circuit days. By leveraging the slow-start phase in TCP congestion control, the elephant flows can rapidly enlarge their CWNDs. During the last RTT, the elephant flows are blocked and have their packets buffered within the AccelToR switch. When the circuit to a destination rack is established, the switch sends out the buffered packets to fill the capacity of the optical link, and the elephant flows, which have their CWNDs already grown large because of the accelerating, continue to inject packets to the optical link at high rates for realizing a high bandwidth utilization.

Note that unlike the previous works [9], [12], [14], [22], we seek to restrict our solution within the ToR switch without requiring any explicit coordination from end-hosts.

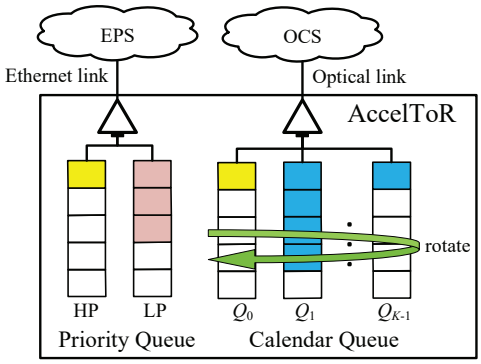


Fig. 3. The priority queue and calendar queue of AccelToR switch.

B. Switch Design

Before describing the detailed design of the AccelToR switch, we first introduce some notations. In a circuit/packet hybrid data center network as presented in Fig. 1, we refer to the current day as day_0, and refer to the subsequent days as day_1, day_2, $\cdots$. Consider a specific source rack S, we denote the group of the elephant flows from S that have the optical circuit to their destination rack currently available as $\mathbf{F}_0$, and denote the group of the elephant flows with their circuit established in day_i as $\mathbf{F}_i$.

As shown in Fig. 3, in AccelToR, each egress port to the packet network is associated with a *priority queue* [35] composed of two queues, namely the high-priority (HP) queue and the low-priority (LP) queue, and the LP queue can dequeue packets only when the HP queue is empty.

Each egress port to the optical switched network is associated with a *calendar queue* [36] composed of K queues (K is a constant independent to the network size N), denoted as $Q_0, Q_1, \cdots, Q_{K-1}$, where Q_i is the queue for packets from the flows in $\mathbf{F}_i$, $0 \leq i < K$. Note that in calendar queue, only Q_0 is allowed to dequeue packets, while all the other queues are paused, as no circuits to their destination racks is available. For packets belonging to $\mathbf{F}_i$ with $i \geq K$, they are simply dropped as there is no queue for them.

The calendar queue in AccelToR supports a *rotation* operation that synchronizes with the OCS reconfiguration. More specifically, during the night when the OCS is reconfigured after day_0, the AccelToR switch rotates Q_i to Q_{i-1} (for $1 \leq i < K$), and the previous Q_0 becomes Q_{K-1}. After the night, the switch pauses Q_{K-1}, the former Q_0, and unpauses the new Q_0 to dequeue its buffered packets. Note that after the rotation, the previous $\mathbf{F}_1$ becomes $\mathbf{F}_0$ and its belonging flows start to transfer through the circuit network. For the other flow groups, $\mathbf{F}_i \rightarrow \mathbf{F}_{i-1}, 1 < i < K$, and a new flow group $\mathbf{F}_{K-1}$ containing the elephant flows of day_{K-1} is formed and associated with Q_{K-1}. For example, if $K = 3$, then after each night, $Q_2 \rightarrow Q_1$, $Q_1 \rightarrow Q_0$, and $Q_0 \rightarrow Q_2$.

Note that both priority queue and calendar queue are mature disciplines supported by commodity switches. Priority queue is widely supported by nearly all the off-the-shelf switches [35], and calendar queue can be realized on existing programmable hardware switches [36], such as the Intel Tofino switch [34].

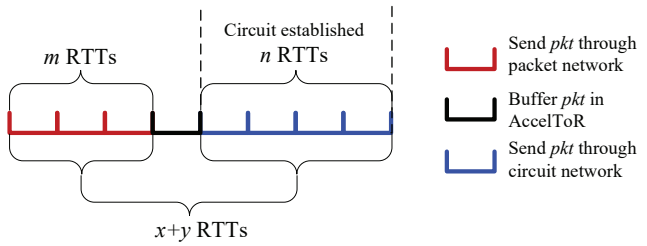


Fig. 4. Demonstration of TCP flow acceleration.

C. Accelerating TCP with Packet Scheduling

As we will describe in Sec. III-G, at each source rack, we select elephant flows from the top-k largest flows to each destination rack, group them to $\mathbf{F}_0$, $\mathbf{F}_1$, $\cdots$, $\mathbf{F}_{N-1}$, and label the other flows as mice flows. An AccelToR switch enforces the following rules to schedule packets based on their labels.

- **Case I**: If a packet is from a delay-sensitive mice flow, enqueue it to the HP queue for transferring through the packet network.
- **Case II**: If a packet is from an elephant flow f in $\mathbf{F}_0$, enqueue it to Q_0 for transferring through the circuit network.
- **Case III**: If a packet is from a flow f in $\mathbf{F}_i$, where $1 \leq i < K$, then,
  - Either accelerate the flow by enqueueing the packet to the LP queue for transferring through the packet network;
  - Or enqueue the packet to Q_i for buffering.
- **Case IV**: If a packet is from a flow f in $\mathbf{F}_i$, where $K \leq i < N$, drop the packet.

However, in the above procedure, there are two unsolved problems: First, how many queues are required in the calendar queue? That is, how to decide K? Second, for a packet in **Case III**, when to transfer it through the packet network for accelerating and when to buffer it in the calendar queue?

To answer the questions, we first suppose that an elephant flow is accelerated for m RTTs and has its packets buffered in the calendar queue in the last RTT before its circuit day. If m is known, then the flow's packets should be enqueued to the calendar queue K_1 days before its circuit is established, where

$$K_1 = \lceil \frac{(m+1) \times \text{RTT}_{\text{pkt}}}{\tau_{\text{day}} + \tau_{\text{night}}} \rceil \tag{1}$$

In (1), RTT_{pkt} is the RTT of the packet network, τ_{day} and τ_{night} are the durations of day and night respectively. Note that RTT_{pkt}, τ_{day}, and τ_{night} are either measurable, or configurable network parameters, thus are already known. In the following, we present and solve an integer optimization problem to decide m, the only unknown parameter in (1).

Let $n = \lfloor \frac{\tau_{\text{day}}}{\text{RTT}_{\text{cir}}} \rfloor$ be the number of the circuit RTTs in a day, where RTT_{cir} is the RTT of the circuit network. As shown in Fig. 4, for the TCP sender of an elephant flow, it sends packets through the packet network for m RTTs and through the circuit network for n RTTs. Suppose that during the $m+n$ RTTs, the sender spends x RTTs in the slow-start phase growing its CWND from 1 MSS, and in the y subsequent RTTs, it is in the congestion-avoidance phase, then we have

$$x + y = m + n \tag{2}$$

We then study the relationship between x and y. In the first x RTTs in slow-start, an elephant flow's CWND is enlarged to X between 2^{x-1} and 2^x, and is linearly grown to $X+y$ after y RTTs in congestion avoidance. After detecting a timeout caused by packet losses that happen after the circuit day, the flow's CWND is reduced to 1 and its `ssthresh` is set as $\frac{X+y}{2}$, which means that next time the sender will exit the slow-start phase at a CWND size of $\frac{X+y}{2}$. Clearly, at equilibrium, we have $\frac{X+y}{2} = X$, which leads to $2^{x-1} < y = X \leq 2^x$.

Given the above constraints, we aim to accelerate an elephant flow as early as possible, so as to sufficiently enlarge its CWND. To this end, we aim to find the largest m by solving the following integer optimization problem.

$$\begin{aligned} \max \quad & m \\ s.t. \quad & m + n = x + y \\ & 2^{x-1} < y \leq 2^x \\ & n = \lfloor \tfrac{\tau_{\text{day}}}{\text{RTT}_{\text{cir}}} \rfloor \\ & m, x, y \in \mathbb{Z}^+ \end{aligned} \tag{3}$$

Note that in (3), n is a fixed integer parameter decided by τ_{day} and RTT_{cir}, m, x, and y are integer variables, and we aim to maximize m.

Unfortunately, the problem in (3) is unbounded, as we can have x and y as large as possible, and compute m as $(x+y-n)$. However, if we accelerate an elephant flow many RTTs before its circuit day, we are indeed transferring the flow through the packet network, which brings congestions as we have discussed in Sec. II.

To avoid congesting the packet network, when accelerating an elephant flow, we prefer to transfer its packets through the packet network only in the slow-start phase. This is because when c packets are transferred in an RTT in the slow-start phase, the flow's CWND is expected to grow to $2c$, but if the flow is under congestion avoidance, its CWND only grows to $c+1$. In other words, it is more cost-effective to accelerate a TCP flow in its slow-start phase. For this reason, we require that the RTTs for accelerating a flow must not exceed the RTTs of its slow-start phase, i.e., $m \leq x$, and by incorporating this constraint, the problem in (3) becomes

$$\begin{aligned} \max \quad & m \\ s.t. \quad & m + n = x + y \\ & m \leq x \\ & 2^{x-1} < y \leq 2^x \\ & n = \lfloor \tfrac{\tau_{\text{day}}}{\text{RTT}_{\text{cir}}} \rfloor \\ & m, x, y \in \mathbb{Z}^+ \end{aligned} \tag{4}$$

For the problem in (4), we have the following result.

Theorem 1. *Given a specific value of n, i.e., number of RTTs in a circuit day, the problem in (4) is bounded and has an unique integer solution, in which $x = m$ and $y = n$.*

Proof. Since we seek to maximize m, from $m + n = x + y$ and $m \leq x$, it is easy to see that in the optimal solution, we

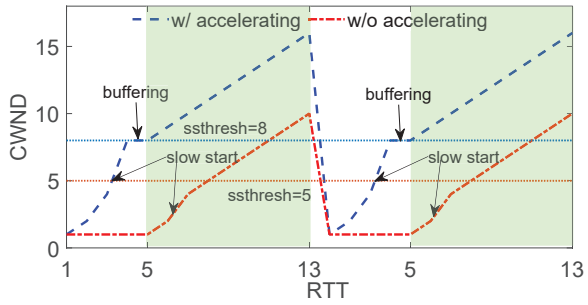


Fig. 5. Comparison of CWND evolvements of an accelerated elephant flow and a flow not accelerated, where shaded regions represent circuit days.

have $x = m$ and $y = n$. From the third constraint, we can see that $2^{x-1} < n \leq 2^x$, so given a concrete value of n, we can find a unique solution for x, and solve the problem in (4). □

We use an example to demonstrate how an elephant flow is accelerated. Suppose $n = 8$, i.e., a day contains 8 RTTs, then the only feasible solution for (4) is $x = m = 3$ and $y = 8$. As shown in Fig. 5, the AccelToR switch accelerates the elephant flow for $m = 3$ RTTs, during which the flow is under the slow-start phase and grows its CWND to $2^m = 8$ MSS. During the circuit day, the flow is under congestion avoidance and linearly grows its CWND to $2^m + n = 16$ MSS. After a timeout caused by blocking, the flow's CWND is reduce to 1 MSS, and its `ssthresh` is set as $\frac{2^m+n}{2} = 8$ MSS, which is the size that the flow can grow its CWND after being accelerated for $m = 3$ RTTs before its circuit day in the next week.

After m is decided by solving (4), we can apply (1) to compute K_1. In the above example, with τ_{day} =240 µs, τ_{night} =20 µs, and RTT_{pkt} =30 µs, we have $K_1 = 1$ and $K = K_1 + 1$, as day_0 also consumes a queue in the calendar queue. However, in our practice, we set $K = K_1 + 2$, and use one additional queue in the calendar queue as a *safe margin*, which we explain in Sec. III-E.

Finally, after obtaining m and K, we detail the packet scheduling rules in **Case III** as:

- **Case III**: If a packet is from a flow f in $\mathbf{F}_i$, where $1 \leq i < K$, then,
  - **Subcase III-A**: If there is $\geq \text{RTT}_{\text{pkt}}$ left before day_i starts, accelerate the flow f by enqueueing the packet to the LP queue for accelerating.
  - **Subcase III-B**: If there is $< \text{RTT}_{\text{pkt}}$ left before day_i starts, enqueue the packet to Q_i for buffering.

Algorithm 1 presents the complete packet scheduling algorithm.

D. Analysis and Comparison

In this section, we analytically compare the case that an elephant TCP flow is accelerated with the case that it is not accelerated (e.g., [5]). In both cases, we optimistically assume that the flow resumes to send packets and enlarges its CWND as soon as it is accelerated or unblocked. Similar to (4), for the non-accelerating case, the RTTs x and y that the flow spends

Algorithm 1: Packet scheduling algorithm

```
Algorithm Packet scheduling on AccelToR
    Input : packet pkt of flow f
    switch f do
        case f is a mice flow do
            Enqueue pkt to HP queue ;  /* Case I */
        case f ∈ F_0 do
            Enqueue pkt to Q_0 ;  /* Case II */
        case f ∈ F_i, 1 ≤ i < K /* Case III */
        do
            if there is ≥ RTT_pkt left before day_i then
                Enqueue pkt to LP queue ; /* Subcase
                  III-A */
            else
                Enqueue pkt to Q_i ; /* Subcase III-B
                  */
        case f ∈ F_i, K ≤ i < N do
            Drop pkt ;  /* Case IV */
```

in the slow-start and congestion-avoidance phases are obtained by solving the following problem.

$$\begin{cases} x + y = n \\ 2^{x-1} < y \leq 2^x \end{cases} \tag{5}$$

where $x, y \in \mathbb{Z}^+$ are unknown integers and $n = \lfloor \frac{\tau_{\text{day}}}{\text{RTT}_{\text{cir}}} \rfloor$ is the RTTs in a circuit day. For example, given $n = 8$, the only feasible solution for (5) is $x = 3$ and $y = 5$.

Fig. 5 presents the CWND evolvements of an elephant flow being accelerated for $m = 3$ RTTs and a flow without accelerating. In both cases, a day contains $n = 8$ RTTs. One can see that the accelerated flow grows its CWND to 16 MSS by the end of a circuit day, while the flow that is not accelerated has its CWND grown to 10 MSS. Moreover, during a circuit day, the accelerated flow sends 2.19 times more packets than the flow without accelerating, thanks to its larger CWND and the packets buffered within the AccelToR switch. From the comparison, we can see that by accelerating, an elephant flow can better utilize the optical bandwidth and transfer more data.

We also compare the buffering space required by AccelToR with the approach that maintains per-destination queues in the ToR switch (e.g., [8]). With $m = 3$ and $K = 3$ as in the above example, the calendar queue of the AccelToR switch needs to buffer $2^m = 8$ packets for each elephant flow. If $|\mathbf{F}_i| = 32$, i.e., there are 32 elephant flows to each destination, then the total size of the calendar queue should be $K \times 32 \times 8 = 768$ packets. On the other hand, consider a network composed of $N = 50$ racks, in which each ToR switch maintains 49 per-destination queues. Given the same packet buffering space, each per-destination queue can buffer no more than 16 packets for 32 flows. In other words, an elephant flow can buffer fewer than one packet on average, while having all the other on-the-fly packets dropped after being blocked. From the comparison,

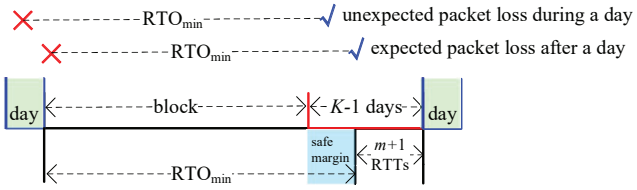


Fig. 6. By enqueueing packets to LP queue one day earlier than RTO_{min}, AccelToR can recover from both expected packet losses caused by blocking and unexpected packet losses that happen during the previous circuit day.

we can see that AccelToR is scalable to network size by imposing an $O(1)$ memory overhead, thus is compatible with today's switch hardware.

Finally, unlike the approaches that unblock all the elephant flows all the time [22], [23], AccelToR only allows the elephant flows to transfer through the packet network for a very short time of m RTTs. In addition, since all the packets from the accelerated elephant flows are enqueued to the LP queue, they will not jam the mice flows, whose packets are enqueued to the HP queue and get transferred with a higher priority. For these reasons, AccelToR is capable to preserve short FCTs for the mice flows in the packet network.

E. RTO_{min} and Safe Margin

In the above sections, we optimistically assume that a blocked elephant flow always resumes sending packets as soon as it is accelerated. However, at the moment when an elephant flow is eligible to send packets, it may still wait for the retransmission timer to expire, especially after successive packet losses with its `RTO` doubled multiple times.

In standard TCP [37], after the initial packet loss, a TCP sender computes its `RTO` as $\text{SRTT} + 4 \times \text{RTTVAR}$, where SRTT and RTTVAR are the smoothed RTT and RTT variance respectively. Meanwhile, modern OSes have a lower bound on `RTO` named RTO_{min}, which is a configurable parameter, and in many Linux versions, RTO_{min} is set as 200 ms by default. Obviously in today's data centers with SRTT and RTTVAR in microseconds, the millisecond-scale default RTO_{min} is indeed the initial `RTO` enforced by TCP senders [38].

We leverage the configurable RTO_{min} at end-host to ensure that a blocked elephant flow resumes to send packets as soon as it is accelerated. In particular, we configure RTO_{min} as

$$\text{RTO}_{\text{min}} = \tau_{\text{week}} - \tau_{\text{day}} - (m+1) \times \text{RTT}_{\text{pkt}} \qquad (6)$$

where τ_{week} is the duration of a week. For example, given τ_{week} =12.74 ms, τ_{day} =240 µs, RTT_{pkt} =30 µs, and $m = 3$, RTO_{min} should be set as 12.38 ms. Clearly, by configuring RTO_{min} with (6), after the circuit to the destination becomes unavailable and packets are dropped, the elephant flows in $|\mathbf{F}_i|$ will always resume to send packets after the AccelToR switch starts to accelerate it in the next week.

Note that according to (1) and (6), we have $K_1 = \left\lceil \frac{\tau_{\text{week}} - \tau_{\text{day}} - \text{RTO}_{\text{min}}}{\tau_{\text{day}} + \tau_{\text{night}}} \right\rceil$, which means that after an elephant flow is blocked, its retransmission timer is supposed to expire within K_1 days and nights before the flow's circuit day in the next week. However, as we employ one additional queue in the calendar queue as the safe margin (i.e., $K = K_1 + 2$), the flow's packets are indeed eligible to be enqueued to the calendar queue one day earlier. This is because an elephant flow has chances to accidentally lose packets during its circuit day. For example, packets may be corrupted in the optical network for reasons such as connector contaminations and decaying transmitters [39]; and when there is a lot of intra-rack traffic, packets may also be dropped due to the congestion within the source or destination rack. As demonstrated in Fig. 6, when such incidents happen, if a flow is eligible to enqueue packets to the calendar queue one day earlier, it can recover from these unexpected packet losses.

F. Configuring Queue Size and Right-sizing $\mathbf{F}_i$

We configure the size of each queue in the calendar queue slightly smaller than $C \times \text{RTT}_{\text{cir}}$, where C is the capacity of the optical path, because of the following reasons. First, the data buffered in a queue should not exceed the bandwidth-delay product (BDP) of the optical path. Second, to achieve a high utilization of the optical bandwidth, the size of each queue Q_i in the calendar queue should be close to $C \times \text{RTT}_{\text{cir}}$.

Once the queue size B is decided, we carefully control the size of the set $\mathbf{F}_i$ to avoid over-filling Q_i. Recall that each flow has grown its CWND up to 2^m MSS after being accelerated for m RTTs, if a queue in the calendar queue can buffer up to B packets, we require that $2^m \times |\mathbf{F}_i| \leq B$, so that a queue will not overflow after each flow in $\mathbf{F}_i$ has been accelerated for m RTTs.

To right-size $\mathbf{F}_i$, for each source-destination rack pair, we maintain no more than $\frac{B}{2^m}$ elephant flows in $\mathbf{F}_i$, and label the other flows as mice flows.

G. Selecting Elephant Flow

To select elephant flows, we can apply the methods for detecting the heavy-hitter flows, such as the ones proposed in [40] and [41]. In particular, the data structure for tracking heavy hitter flows (referred to as *sketch*) can be deployed at the ingress pipeline of each AccelToR switch, and monitors all the traversing packets. For each source-destination rack pair, the module identifies the top-k largest flows, and label no more than $\frac{B}{2^m}$ of them as the elephant flows in $\mathbf{F}_i$. For each elephant flow, the AccelToR uses a dedicated counter to count its size. If a counter does not increase for some time, the corresponding elephant flow is considered to leave the network, and in this case, it is removed from $\mathbf{F}_i$, and a new flow is selected from the heavy-hitter flows and added to $\mathbf{F}_i$. Note that replacing elephant flows take time, during which the optical link would be slightly under-utilized.

IV. Evaluation

A. AccelToR in `ns-3`

We have realized AccelToR as well as a circuit/packet hybrid data center network in `ns-3` [42]. Realizing such a network in `ns-3` is non-trivial, as the existing `ns-3` modules do not support pausing, unpausing, and rotating queues as required by AccelToR. To overcome these limitations, we

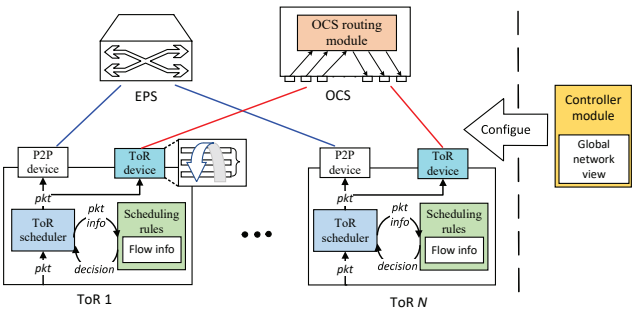


Fig. 7. Overview of the AccelToR `ns-3` realization.

develop a new device class namely `ToRNetDevice`, and realize all the required operations from scratch.

An overview of our `ns-3` realization is presented in Fig. 7. Besides the ToR device module, we also develop a number of other modules that are necessary for our experiment. In particular, we develop a Controller module, which maintains a global network view and configures both ToR and OCS switches in the simulated network. We develop an OCS routing module to enforce a cycle of matchings from the Controller on OCS. Within a ToR switch, we develop the ToR scheduler module that applies Algorithm 1 to locally decide how to handle packets. In addition to the modules, we also develop the corresponding helpers for facilitating users to construct a circuit/packet hybrid data center network. We make our AccelToR `ns-3` realization publicly available[2].

B. Experiment Setup

With the `ns-3` AccelToR realization, we carry out packet-level simulations on a circuit/packet hybrid data center network composed of N racks, where each rack connects to an ESP switch and an OCS switch as in Fig. 7. We configure the durations of a day and a night as τ_{day}=240 µs and τ_{night}=20 µs, and set the capacities of the uplinks connecting to the EPS and the OCS as 10 Gbit/s and 80 Gbit/s respectively. For any pair of inter-rack end-hosts, the RTT of the optical circuit network is set as 55 µs, and the RTT of the packet network is 60 µs. An experiment lasts 5 seconds.

Note that given the above configuration, the week duration and number of weeks contained in the 5-second simulation depend on the network size. For example, suppose that the network has 72 racks, then each week lasts 18.46 ms, and 5 seconds contain 270 weeks. When the network size decreases, more weeks are contained in the simulation.

We simulate network flows with their sizes drawn from pFabric [43]. At the beginning of each experiment, for each pair of source-destination racks, we select $|\mathbf{F}_i| = 80$ largest flows as elephant flows, and label the others as mice flows. Subsequent flows arrive to the network following a Poisson process, and we vary the Poisson arrival rate to impose different *load*, which is defined as the ratio between the aggregated rate of the mice flows and the EPS bandwidth, on the packet network. We use Count-Min (CM) [44], the simplest heavy-hitter detecting sketch to select elephant flows: Once an elephant flow leaves the network, we employ the CM sketch to select a new heavy-hitter flow to replace it. In addition, after a mice flow is labeled as an elephant flow, we add a new mice flow with its size as the original flow's residual size, so that the overall load on the packet network is unchanged. We do not simulate intra-rack traffic for simplicity.

We compare the following solutions for TCP data transportation on the circuit/packet hybrid data center network.

- **Blocking**. In such a solution, elephant flows are blocked when their circuits to the destination racks are unavailable, and mice flows are transferred through the packet network all the time. Each ToR switch maintains N queues. Among them, one queue is for the packet network that is never blocked, and the other $N-1$ queues are for the circuit network, where each queue is dedicated to one destination rack and buffers packets of the corresponding elephant flows when the circuit to the destination rack is unavailable. The size of the queue for the packet network is 300 kB, and the $N-1$ per-destination queues for the circuit network have a total size of 1.5 MB. For fairness, we set RTO_{min} on end-hosts as

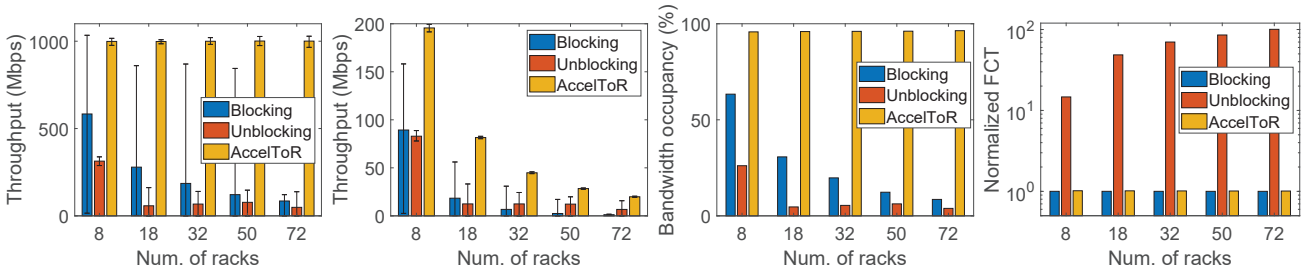
$$\text{RTO}_{\text{min}} = \tau_{\text{week}} - \tau_{\text{day}} \tag{7}$$

  so that a blocked elephant flow can resume sending packets as soon as it is unblocked. Note that the systems in [5], [6], [8], [9], [20] choose to block elephant flows, and among them, Mordia [8] and NegotiaToR [20] maintain per-destination queues within the ToR switch.
- **Unblocking**. In this solution, the ToR switch always transfers the mice flows through the packet network, while for the elephant flows, they are sent through the circuit network when their circuits are established, and are transferred through the packet network during the other time. As in [22], the egress ports connecting to the EPS and OCS share a queue, and to enable a fair comparison, the queue size is set as 1.8 MB. Note that existing systems in [22], [23] transfer elephant flows via the packet network when their circuits are unavailable.
- **AccelToR**. Our proposed AccelToR follows Algorithm 1 to schedule packets from different flows. In AccelToR, the HP and LP queues of the priority queue each has a size of 150 kB, and according to Sec. III-C and the network parameters, we have $m = 2$ and use $K = 3$ queues in the calendar queue. Each queue has a size B of 500 kB, which is slightly smaller than $C \times \text{RTT}_{\text{cir}}$ =550 kB, and satisfies the condition $B \geq 2^m \times |\mathbf{F}_i|$ =480 kB, as we have discussed in Sec. III-F. The total packet buffering space is 1.8 MB as in Blocking and Unblocking. We apply (6) to configure RTO_{min}.

Note that in the three solutions, mice flows are transferred through the packet network all the time, and the solutions differ in how the elephant flows send packets through the packet network. We focus on the following performance metrics in the experiments.

- **Weekly throughput**. For each elephant flow, we measure its throughput averaged through an entire week.

[2]https://github.com/HPCC724/RDCNSimulate



(a) Daily throughput (b) Weekly throughput (c) Bandwidth occupancy (d) Normalized FCT

Fig. 8. Comparisons of Blocking, Unblocking, and AccelToR on elephant flows' (a) daily and (b) weekly throughputs with error bars indicating 10^{th} and 90^{th} percentiles, (c) optical link's bandwidth occupancy, and (d) mice flows' normalized FCTs, in a circuit/packet hybrid data center network containing various numbers of racks.

- **Daily throughput**. We also measure an elephant flow's throughput during a day when the circuit to its destination rack is available.
- **Optical link bandwidth occupancy**. During each circuit day, we measure the usage of the optical link's bandwidth in percentage.
- **Normalized flow completion time (FCT)**. For each mice flow, we compute its normalized FCT as the flow's actual FCT normalized to its ideal FCT when no other flows are active in the packet network, i.e.,

$$\text{Norm. FCT} = \frac{\text{FCT under examined solution}}{\text{FCT if is the only flow in packet network}} \quad (8)$$

- **Aggregated congestion window (CWND)**. We study the evolvement of the CWND aggregated from all the elephant flows during a circuit day.

The above metrics are aggregated from all the weeks in the 5-second simulation. Finally, we use NewReno [45] as the end-host TCP congestion control if not otherwise specified.

C. AccelToR vs. Blocking

In the first experiment, we vary the network size from 8 to 72 racks, and evaluate the solutions of Blocking, Unblocking, and AccelToR. The load on the packet network introduced by the mice flows is fixed as 50%. We present the daily and weekly throughputs of the elephant flows, the occupancy of the optical link bandwidth, and the normalized FCTs of the mice flows in Fig. 8. From Fig. 8(a), (b), and (c), we can see that comparing with Blocking, our proposed AccelToR achieves much higher throughputs for elephant flows and a much higher optical link bandwidth occupancy. In addition, the daily throughput and the optical bandwidth occupancy of AccelToR are stable in different-sized networks.

On the contrary, under Blocking, the daily throughputs and bandwidth occupancy decrease as the network contains more racks. This is because given a constant packet buffering space, when the network size increases, the size of each per-destination queue decreases. In addition, Fig. 8(a) and (b) shows that the elephant flows under AccelToR exhibit much smaller throughput variance than under Blocking. This is because the flows accelerated by AccelToR have similar rates, while under Blocking, many flows do not send any data as they fail in competing the limited buffering space of the per-destination queues. Our observation suggests that by accelerating elephant flows a few RTTs before their circuit days, AccelToR substantially outperforms Blocking regarding the throughputs and optical bandwidth utilization.

To better understand the strength of AccelToR over Blocking, we fix the network size as 50 racks, and present the distributions of the weekly throughputs of the elephant flows from rack 5 to rack 37 in Fig. 9. We also estimate the elephant flows' FCTs by assigning each flow a size randomly selected from the top-200 largest flows in pFabric, and present the distribution in Fig. 10. Fig. 9 shows that under AccelToR, most of the elephant flows have a weekly throughout between 26.9 and 29.7 Mbit/s, and the variance is small. Fig. 10 suggests that comparing to Blocking, AccelToR can substantially reduce the flows' FCTs.

In Fig. 11, we plot the evolvements of the aggregated CWNDs under different solutions during the day. From the figure, we can see that under AccelToR, the CWND has already grown to 4 MSS at the beginning of the day, and it continues to grow during the circuit day. On the other hand, the elephant flows under Blocking have an aggregated CWND as small as 1 MSS at the beginning of the day, and about 90% elephant flows do not send packets and enlarge their CWNDs during the circuit day, as the senders of these flows are suffering packet losses due to buffer overflow.

Fig. 8(d) shows that both Blocking and AccelToR preserve short FCTs for the mice flows, and in Fig. 12, we present the distributions of the normalized FCTs for the mice flows under the three solutions. We can see that Blocking has normalized FCTs close to 1 for most mice flows, as the elephant flows are restricted from the packet network. From the inset figure of Fig. 12, we can see that AccelToR has the normalized FCTs slightly larger, as it exploits the packet network to accelerate the elephant flows. However, AccelToR still preserves short FCTs for the mice flows with the distribution curve basically overlapping with the one of Blocking, as it introduces only a limited volume of traffic to the packet network, and employs a priority queue to prevent the elephant flows from competing bandwidth with the mice flows.

From the above experiment we can see that under Blocking, given a limited size of the packet buffering space, when the network contains more racks, the per-destination queue has its size reduced, and consequently the elephant flows suffer more

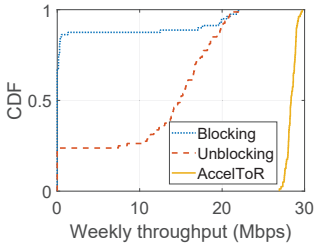


Fig. 9. Distributions of elephant flows' weekly throughputs under Blocking, Unblocking, and AccelToR.

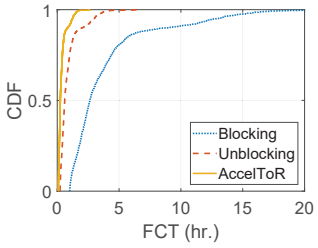


Fig. 10. Distributions of estimated FCTs of elephant flows under Blocking, Unblocking, and AccelToR.

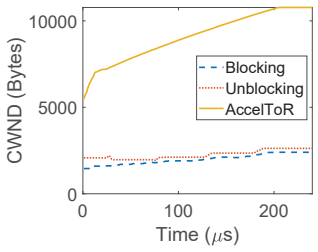


Fig. 11. Evolvements of aggregated CWNDs during a day under Blocking, Unblocking, and AccelToR.

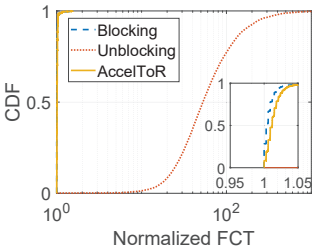
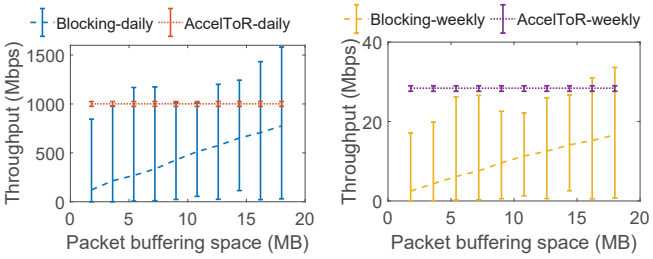


Fig. 12. Distributions of normalized FCTs of mice flows under Blocking, Unblocking, and AccelToR. Note that the figure and the inset figure use log and linear scales on x-axes respectively.



(a) Daily throughput (b) Weekly throughput

Fig. 13. Comparing AccelToR with Blocking on (a) daily and (b) weekly throughputs with error bars indicate 10^{th} and 90^{th} percentiles, when the latter has larger packet buffers.

packet losses due to buffer overflow. A question one may ask is, if the ToR switch can employ a larger packet buffer, will Blocking outperforms AccelToR? In the following experiment, we presume that Blocking can employ up to 10 times larger packet buffer than AccelToR, and compare the elephant flows' daily and weekly throughputs of the two solutions in Fig. 13. The figure shows that even with a per-port packet buffer as large as 18 MB, which is impractical on today's switch hardware, the throughputs achieved by Blocking are still significantly lower than AccelToR. The result suggests that enlarging the in-switch packet buffer for Blocking is neither practical nor effective.

D. AccelToR vs. Unblocking

Fig. 8 also presents performances of Unblocking and compares it with AccelToR. From Fig. 8(a), (b), and (c), we can see that Unblocking has the lowest daily throughput and optical link bandwidth occupancy among the three solutions, and its weekly throughput is also significantly lower than AccelToR due to its low utilization of the optical bandwidth, despite that it allows elephant flows to transfer all the time. The elephant flows under Unblocking exhibit a greater variance than under AccelToR. For example, Fig. 9 shows that under Unblocking, an individual flow has a weekly throughput varying from 0 to 22 Mbit/s, depending on how it is congested in the packet network. Moreover, as many as 80% elephant flows do not send any packet during the circuit day. Fig. 10 suggests that for the elephant flows, their FCTs are significantly longer under Unblocking than under AccelToR, and Fig. 11 shows that comparing with AccelToR, the aggregated CWND under Unblocking is much smaller and grows slowly, as many flows stall waiting for the retransmission timers to expire since their last packet losses due to the congestions in the packet network.

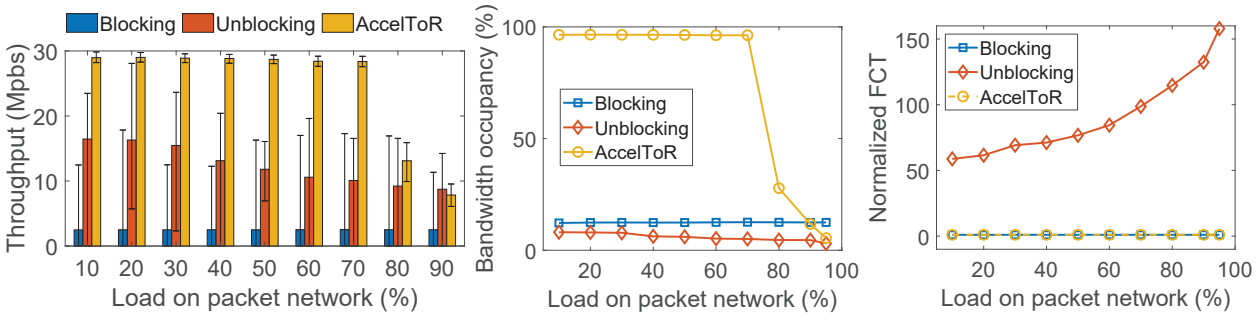
From Fig. 8(d), we can see that Unblocking has much longer FCTs for the mice flows than AccelToR and Blocking. The reason is that under Unblocking, when the circuits are unavailable, the elephant flows compete bandwidth with the mice flows, and introduce severe congestions to the packet network. To better understand the impact of the congestions on the mice flows, we study the FCT distributions in Fig. 12. We can see that Unblocking causes very long FCTs for many mice flows, as these flows have slim chances to successfully send packets when competing bandwidth with the elephant flows, and stall after experiencing successive packet losses.

E. Impact of Congestion in Packet Network

The basic idea of AccelToR is to use the packet network to accelerate elephant TCP flows before their circuit paths are established. However, when the packet network is congested, the acceleration should be interrupted. In this experiment, we examine the impact of the congestions in the packet network, and in particular, we focus on a network containing 50 racks, and change the Poisson arrival rate of the mice flows to make the load on the packet network vary from 10% to 95%.

Fig. 14 presents performances of Blocking, Unblocking, and our proposed AccelToR under various loads on the packet network. From Fig. 14(a), we can see that the elephant flows under Blocking are not impacted, as they do not transfer through the packet network at all. The weekly throughputs of the elephant flows under Unblocking decrease as the packet network becomes more and more congested. This is because the TCP senders' congestion states such as CWND and `ssthresh` are largely shaped by the congested packet network. Under AccelToR, we find that the weekly throughputs of the elephant flows remain high as long as the load imposed on the packet network is below 70%, but when the packet network is severely congested with the load exceeding 70%, the throughputs decrease dramatically.

In Fig. 14(b), we present the optical bandwidth occupancies. The result conforms with the above observation that AccelToR utilizes over 90% of the optical bandwidth as long as the



(a) Weekly throughput (b) Bandwidth occupancy (c) Normalized FCT

Fig. 14. Comparisons of Blocking, Unblocking, and AccelToR on (a) elephant flows' weekly throughputs with error bars indicating 10^{th} and 90^{th} percentiles, (b) optical link's bandwidth occupancy, and (c) mice flows' normalized FCTs, in a circuit/packet hybrid data center network with various loads imposed on packet network.

load on the packet network is below 70%, but the bandwidth occupancy drops quickly as the packet network gets severely congested. Fig. 14(c) presents the normalized FCTs of the mice flows, from which we can see that Blocking and AccelToR do not inflate the mice flows' FCTs, as they either consume little bandwidth on the packet network, or prioritize the mice flows over the elephant flows; but under Unblocking, the FCTs increase with the congestion level, as all the flows compete bandwidth in the packet network.

Our observation from Fig. 14 shows that when the load of the mice flows on the packet network is high (e.g., above 70%), the elephant flows' weekly throughputs and the optical link bandwidth occupancy degrade rapidly. This is because when the HP queue of the priority queue in the AccelToR switch is busy sending packets from the mice flows, the LP queue for accelerating the elephant flows is jammed, and drops packets when the queue is full. In this case, the elephant flows waste their optical days waiting for the retransmission timers to expire rather than growing their CWNDs.

The experiment result suggests that when the packet network is severely congested, AccelToR chooses preserving short FCTs for mice flows over accelerating elephant flows. We believe that such a choice is reasonable, as most mice flows are delay-sensitive, and their FCTs directly impact users' service experiences. As bandwidth in data center tends to be overprovisioned to cope with the unpredictable usage patterns, the long-term load is generally low, so in most time, AccelToR can preserve short FCTs for mice flows and accelerate elephant flows simultaneously.

F. Effectiveness of Safe Margin

In Sec. III-E, we propose to use one additional queue in the calendar queue as safe margin for recovering from unexpected packet losses that happen during the circuit days. In this experiment, we examine the effectiveness of this design. In particular, we compare the AccelToR solutions with and without the safe margin, and examine the optical link bandwidth occupancies under the two cases. Note that when AccelToR runs without the safe margin, we use $K = K_1 + 1 = 2$ queues in the calendar queue, and the elephant flows are eligible to send packets through the packet network one day later comparing with the safe margin case. In both cases, the network contains 50 racks.

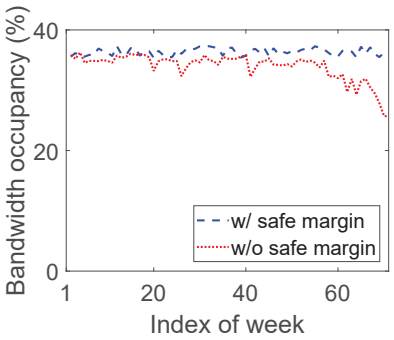


Fig. 15. Optical link bandwidth occupancies in circuit days of 70 consecutive weeks under AccelToR with and without safe margin.

We focus on a pair of source-destination racks, and to introduce unexpected packet losses, we set the loss rate within the destination rack as 0.01. Fig. 15 presents the optical link's bandwidth occupancies achieved by both solutions in the circuit days of 70 consecutive weeks. One can see that with the safe margin, the optical bandwidth occupancy remains stable at above 90%. But in the case without the safe margin, the bandwidth occupancy has a decreasing trend and is reduced to 65% after 70 weeks. This is because without the safe margin, when an elephant flow encounters an unexpected packet loss that happens during its circuit day, there are chances that the flow can not recover from the loss and stops sending packets in the subsequent weeks, as we have discussed in Sec. III-E. When many flows stall because of the irrecoverable packet losses, we observe a reduction on the bandwidth occupancy.

Note that although we do not introduce intra-rack traffic in our simulation, however, the experiment result in Fig. 15 suggests that when there is a lot of intra-rack traffic that causes congestions within the racks, AccelToR can successfully recover from the packet losses.

G. Impact of Queue Overflow

In Sec. III-F, we discuss that to avoid queue overflow, the condition $|\mathbf{F}_i| \leq \frac{B}{2^m}$ should be satisfied. In the following, we experiment a case when this condition is violated. More

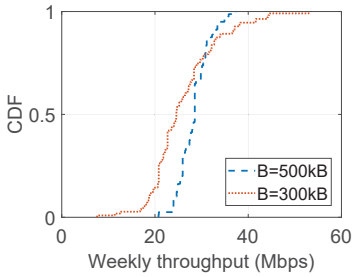
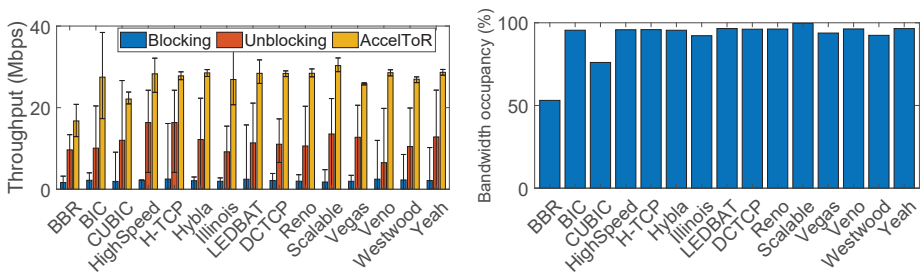


Fig. 16. Distributions of elephant flows' weekly throughputs with queue size set as 300 kB and 500 kB respectively.



(a) Weekly throughput (b) Bandwidth occupancy

Fig. 17. Elephant flows' (a) weekly throughputs achieved by Blocking, Unblocking, and AccelToR with error bars indicating 10^{th} and 90^{th} percentiles and (b) optical link bandwidth occupancy achieved by AccelToR when applying different TCP CCAs.

specifically, we set the size of each queue in the calendar queue as 300 kB, which is smaller than $2^m \times |\mathbf{F}_i|$ =480 kB, and to avoid packet loss in this case, if Q_i is full when a packet of a flow in $\mathbf{F}_i$ arrives, we schedule the packet to be transferred through the LP queue. The experiment is conducted in a 50-rack network.

In Fig. 16, we present the distribution of the 80 elephant flows' weekly throughputs with the queue size set as 300 kB, and compare with the distribution when the queue size is 500 kB that satisfies the condition. We find that with the 300 kB-queues, the elephant flows exhibit a greater variance than the flows under the 500 kB-queues. By investigating the packet traces, we find that when setting the queue size as 300 kB, there are many fast retransmissions caused by packet reordering, and some flows encounter more reordered packets than the other flows, which hinder them from growing their CWNDs. We also find that the optical bandwidth utilization is slightly reduced. The experiment result suggests that it is necessary to right-size $\mathbf{F}_i$ to avoid queue overflow.

H. AccelToR with TCP Variant

In the previous experiments, we use NewReno as the default TCP congestion control. In the following, we experiment as many as 15 other TCP congestion control algorithms (CCAs) available in `ns-3` [46]. The algorithms fall into four categories:

- Loss-based CCA: The TCP variants of BIC [47], CUBIC [48], HighSpeed [49], H-TCP [50], Hybla [51], Reno [52], Scalable [53], Veno [54], and Westwood [55] are loss-based congestion controls. In these algorithms, packet loss is interpreted as a signal of congestion, and sender reduces CWND on detecting packet losses.
- Delay-based CCA: The TCP variants of LEDBAT [56] and Vegas [57] are delay-based. In these algorithms, the sender monitors the end-end delays, and uses the change of the delay as an indication of congestion.
- Hybrid CCA: The TCP variants of Illinois [58], BBR [59], and YeAH [60] combine losses, delays, and other signals in their congestion controls.
- Switch-assisted CCA: The TCP variant of DCTCP [30], [61], which is dedicatedly designed for data center networks, leverages switches to provide Explicit Congestion Notifications (ECNs) to end hosts. In our evaluation, we deploy DCTCP at EPS of the packet network with a default ECN marking threshold as 65 packets.

In Fig. 17(a), we present the weekly throughputs of the elephant flows under the solutions of Blocking, Unblocking, and AccelToR in a network containing 50 racks with a 50% load on the packet network, and experiment the 15 different TCP variants under the three solutions. In Fig. 17(b), we present the optical link bandwidth occupancies achieved by AccelToR when applying different CCAs. From Fig. 17(a), we can see that with all the CCAs, AccelToR achieves higher throughputs for the elephant flows than Blocking and Unblocking, and Fig. 17(b) shows that most TCP variants can realize an optical link bandwidth occupancy as high as above 90%. The observations suggest that AccelToR is effective with most TCP variants under the evaluation, as these CCAs have a slow-start or a similar phase for rapidly seizing the available bandwidth, thus can benefit from the accelerating.

In Fig. 17(b), we find that CUBIC [48] and BBR [59] have their optical bandwidth occupancies as 53.1% and 76.0% respectively, which are relatively lower than the other TCP variants. The reasons are as follows: CUBIC employs a cubic function to increase CWND, and such a function makes the sender less aggressive to explore bandwidth when approaching to the window size of the previous packet loss, which happens at the end of the circuit day.

For BBR, the sender seeks to keep the queue empty by always working at the point with the base RTT. However, since AccelToR buffers packets, the BBR sender will detect a significant RTT increase caused by the buffering, and enter into the drain state by proactively reducing its packet sending rate for draining the queue. For this reason, BBR does not achieve a throughput as high as the other TCP variants.

Finally, we summarize our findings from the experiments as follows.

- AccelToR substantially improves elephant flows' throughputs, and greatly increases utilization ratio of the optical bandwidth. For example, in a network composed of 50 racks, AccelToR achieves 8.24× and 12.92× optical bandwidth occupancies comparing with the alternative solutions that block and unblock the elephant flows respectively.
- AccelToR is effective in preserving short FCTs for mice flows by avoiding congesting the packet network. For

example, in a network composed of 50 racks, AccelToR only increases the FCT of the mice flows 0.97% comparing with the Blocking solution, and reduce 98.82% of the FCT comparing with Unblocking.
- AccelToR incurs a moderate memory overhead that is independent of the network size.
- AccelToR is robust against unexpected packet losses that happen during the circuit day without stalling the TCP senders, and avoids queue overflow by right-sizing the set of the elephant flows.
- AccelToR can benefit a wide range of TCP CCAs by achieving an optical link bandwidth occupancy above 90% for most of the TCP variants under the evaluation.

V. Related Work

To provide a higher bandwidth at a lower cost, in the past decade, people explore to introduce optical circuit switched networks to data centers [5]–[16], [18], [20], [62], in which an optical circuit network core is developed to inter-connect the racks. For configuring OCS, various algorithms are proposed to compute a series of OCS matchings based on the communication demand estimations [24], [25], [28], [29], [63], and coflows are carefully scheduled over the hybrid network [64], [65]. As today's OCSes have reduced the reconfiguration delay from milliseconds to microseconds, people propose to configure OCSes in a demand-agnostic manner, in which the time-consuming computation at the OCS controller is skipped, and an OCS locally cycles through a series of static matchings in a round-robin way [12], [14].

Since OCS reconfiguration incurs non-trivial delays, to preserve the desired properties of the circuit network while providing a continuous connectivity among the racks, circuit/packet hybrid data center networks are proposed [5], [6], [8], [9], [22], [23]. In such a network, when the circuits to the destination racks are unavailable, some works propose to block the corresponding flows [5], [6], [8], [9], and buffer the packets in the per-destination queues maintained either by the ToR switch [8], [20], or by end-hosts [9]. However, Porter et al. [8] show that maintaining per-destination queues raises a scalability concern. Recent works propose to allow all the flows to transfer through all the available paths, so as to reduce the network complexity [22], [23]. However, as we have shown in this work, without segregating network flows, mice flows will be impacted by the congestions brought by elephant flows, and have their FCTs significantly prolonged. AccelToR follows the design choice of segregating elephant and mice flows, but unlike the previous works, AccelToR does not maintain per-destination queues, thus is scalable to network size.

To reconfigure OCS frequently and have a high circuit uptime ratio, in today's OCS-based data center networks, the optical circuit from a source rack to a destination rack typically lasts fewer that 10 RTTs [22]. With such a short circuit time, TCP faces a critical challenge as it is difficult for a TCP sender to sufficiently enlarge its CWND and efficiently utilize the optical link bandwidth. To address this problem, Mukerjee et al. [22] suggest that ToR switches dynamically resize the queues for buffering packets, and a TCP sender at end-host increases CWND on receiving an explicit circuit state feedback sent from the ToR switch. Chen et al. [23] develop a new TCP variant named Time-division TCP (TDTCP) that maintains multiple sets of independent congestion control states for low-bandwidth packet network and high-bandwidth circuit network separately. However, these solutions require end-hosts to closely coordinate with the circuit network, and substantially modify the OS protocol stack. Unlike these approaches, AccelToR does not demand any explicit coordination from end-hosts, and efficiently utilizes the optical bandwidth by achieving high throughputs for elephant flows.

There is a rich literature on exploiting switches for boosting TCP performances in general. For example, active queue management (AQM) techniques are applied for reducing queueing delays for TCP flows [66]–[68]. Turkovic et al. [69] propose to monitor network flows passing a switch based on their congestion control behaviors, and apply appropriate measures to improve the fairness. Chen et al. [70] enhance fairness among TCP flows with weighted fair queueing on programmable switches. For improving TCP at end-host, TAS [71] is an OS service that accelerates TCP by reducing the packet processing overhead. AccelTCP [72] accelerates TCP by offloading TCP operations to NIC hardware. SUSS [73] is a sender-side add-on to safely expedite the growth of the CWND. Unlike these works, we focus on accelerating TCP in a circuit/packet hybrid data center network, and restrict our solution within the ToR switch.

VI. Conclusion

Circuit/packet hybrid data center networks are widely proposed to provide high bandwidth of a circuit network and preserve a continuous connectivity among the racks with a packet network in parallel. As today's OCSes have reduced the reconfiguration delay to microseconds, to achieve a high uptime ratio and a reconfiguration frequency, a circuit day typically lasts fewer than 10 RTTs. Such a short circuit time brings a critical challenge to TCP, as a TCP sender does not have sufficient time to enlarge its CWND and utilize the optical bandwidth.

To address this problem, in this paper, we present AccelToR, a novel ToR switch for improving TCP performance in circuit/packet hybrid data center networks. For enlarging elephant flows' CWNDs, AccelToR "accelerates" the flows by scheduling their packets to be transferred through the packet network a few RTTs before their circuit days, and buffers the last windows of their packets. When the circuit to the destination rack is established, the AccelToR switch sends out the buffered packets to fill the capacity of the optical link, and the accelerated flows, which have their CWNDs already enlarged, continue to increase their windows for sufficiently utilizing the optical bandwidth. We have realized AccelToR in `ns-3`, and packet-level simulations show that comparing with the alternative solutions, AccelToR achieves high throughputs for elephant flows and highly utilize the optical link bandwidth, and it preserves short FCTs for mice flows at the same time; in addition, AccelToR is robust under unexpected packet

losses and can benefit a wide range of TCP congestion control algorithms.

REFERENCES

[1] A. Singh *et al.*, "Jupiter rising: A decade of Clos topologies and centralized control in Google's datacenter network," in *Proc. SIGCOMM'15*, London, UK, Aug. 2015, pp. 183–197.

[2] "ChatGPT," accessed on Oct. 15, 2023. [Online]. Available: https://openai.com/chatgpt

[3] "How giant AI workloads and the looming 'bandwidth wall' are impacting system architectures," Oct. 2022. [Online]. Available: https://www.hpcwire.com/2022/10/24/how-giant-ai-workloads-and-the-looming-bandwidth-wall-are-impacting-system-architectures/

[4] T. Plumb, "AI modeling is eating your network bandwidth - how Broadcom looks to change that," Apr. 2023. [Online]. Available: https://www.sdxcentral.com/articles/analysis/ai-modeling-is-eating-your-network-bandwidth-how-broadcom-looks-to-change-that/2023/04/

[5] N. Farrington, G. Porter, S. Radhakrishnan, H. H. Bazzaz, V. Subramanya, Y. Fainman, G. Papen, and A. Vahdat, "Helios: A hybrid electrical/optical switch architecture for modular data centers," in *Proc. SIGCOMM'10*, New Delhi, India, Aug. 2010, pp. 339–350.

[6] G. Wang, D. G. Andersen, M. Kaminsky, K. Papagiannaki, T. S. E. Ng, M. Kozuch, and M. Ryan, "c-Through: Part-time optics in data centers," in *Proc. SIGCOMM'10*, New Delhi, India, Aug. 2010, pp. 327–338.

[7] K. Chen, A. Singla, A. Singh, K. Ramachandran, L. Xu, Y. Zhang, X. Wen, and Y. Chen, "OSA: An optical switching architecture for data center networks with unprecedented flexibility," in *Proc. NSDI'12*, San Jose, CA, USA, Apr. 2012, pp. 1–14.

[8] G. Porter, R. Strong, N. Farrington, A. Forencich, P. Chen-Sun, T. Rosing, Y. Fainman, G. Papen, and A. Vahdat, "Integrating microsecond circuit switching into the data center," in *Proc. SIGCOMM'13*, Hong Kong, China, Aug. 2013, pp. 447–458.

[9] H. Liu, F. Lu, A. Forencich, R. Kapoor, M. Tewari, G. M. Voelker, G. Papen, A. C. Snoeren, and G. Porter, "Circuit switching under the radar with REACToR," in *Proc. NSDI'14*, Seattle, WA, USA, Apr. 2014, pp. 1–15.

[10] S. Legtchenko, N. Chen, D. Cletheroe, A. Rowstron, H. Williams, and X. Zhao, "XFabric: A reconfigurable in-rack network for rack-scale computers," in *Proc. NSDI'16*, Santa Clara, CA, USA, Mar. 2016, pp. 15–29.

[11] T. Ye, T. T. Lee, M. Ge, and W. Hu, "Modular AWG-based interconnection for large-scale data center networks," *IEEE Trans. Cloud Comput.*, vol. 6, no. 3, pp. 785–799, 2016.

[12] W. M. Mellette, R. McGuinness, A. Roy, A. Forencich, G. Papen, A. C. Snoeren, and G. Porter, "RotorNet: A scalable, low-complexity, optical datacenter network," in *Proc. SIGCOMM'17*, Los Angeles, CA, USA, Aug. 2017, pp. 267–280.

[13] L. Chen, K. Chen, Z. Zhu, M. Yu, G. Porter, C. Qiao, and S. Zhong, "Enabling wide-spread communications on optical fabric with MegaSwitch," in *Proc. NSDI'17*, Boston, MA, USA, Mar. 2017, pp. 577–593.

[14] W. M. Mellette, R. Das, Y. Guo, R. McGuinness, A. C. Snoeren, and G. Porter, "Expanding across time to deliver bandwidth efficiency and low latency," in *Proc. NSDI'20*, Santa Clara, CA, USA, Feb. 2020, pp. 1–18.

[15] H. Ballani, P. Costa, R. Behrendt, D. Cletheroe, I. Haller, K. Jozwik, F. Karinou, S. Lange, B. Thomsen, K. Shi, and H. Williams, "Sirius: A flat datacenter network with nanosecond optical switching," in *Proc. SIGCOMM'20*, New York, USA, Aug. 2020, pp. 782–797.

[16] H. Liu, R. Urata, K. Yasumura, X. Zhou, R. Bannon, J. Berger, P. Dashti, N. Jouppi, C. Lam, S. Li, E. Mao, D. Nelson, G. Papen, M. Tariq, and A. Vahdat, "Lightwave fabrics: At-scale optical circuit switching for datacenter and machine learning systems," in *Proc. SIGCOMM'23*, New York, NY, USA, Aug. 2023, pp. 499–515.

[17] C. Caldeira, O. Souza, O. Goussevskaia, and S. Schmid, "OpticNet: Self-adjusting networks for ToR-matching-ToR optical switching architectures," in *Proc. IEEE INFOCOM'23*, New York City, NY, USA, May 2023, pp. 1–10.

[18] H. Yang and Z. Zhu, "Traffic-aware configuration of all-optical data center networks based on Hyper-FleX-LION," *IEEE/ACM Trans. Netw.*, vol. 32, no. 3, pp. 2675–2688, 2024.

[19] S. Das, A. Silva, and T. S. E. Ng, "Rearchitecting datacenter networks: A new paradigm with optical core and optical edge," in *Proc. IEEE INFOCOM'24*, Vancouver, Canada, May 2024, pp. 1–10.

[20] C. Liang, X. Song, J. Cheng, M. Wang, Y. Liu, Z. Liu, S. Zhao, and Y. Cui, "NegotiaToR: Towards a simple yet effective on-demand reconfigurable datacenter network," in *Proc. SIGCOMM'24*, Sydney, Australia, Aug. 2024, pp. 415–432.

[21] C. Guo, L. Yuan, D. Xiang, Y. Dang, R. Huang, D. Maltz, Z. Liu, V. Wang, B. Pang, H. Chen, Z.-W. Lin, and V. Kurien, "Pingmesh: A large-scale system for data center network latency measurement and analysis," in *Proc. SIGCOMM'15*, London, UK, Aug. 2015, pp. 139–152.

[22] M. K. Mukerjee, C. Canel, W. Wang, D. Kim, S. Seshan, and A. C. Snoeren, "Adapting TCP for reconfigurable datacenter networks," in *Proc. NSDI'20*, Santa Clara, CA, USA, Feb. 2020, pp. 651–666.

[23] S. S. Chen, W. Wang, C. Canel, S. Seshan, A. C. Snoeren, and P. Steenkiste, "Time-division TCP for reconfigurable data center networks," in *Proc. SIGCOMM'22*, Amsterdam, Netherlands, Aug. 2022, pp. 19–35.

[24] H. Liu, M. K. Mukerjee, C. Li, N. Feltman, G. Papen, S. Savage, S. Seshan, G. M. Voelker, D. G. Andersen, M. Kaminsky, G. Porter, and A. C. Snoeren, "Scheduling techniques for hybrid circuit/packet networks," in *Proc. CoNEXT'15*, Heidelberg, Germany, Dec. 2015, pp. 1–13.

[25] S. B. Venkatakrishnan, M. Alizadeh, and P. Viswanath, "Costly circuits, submodular schedules and approximate carathéodory theorems," in *Proc. SIGMETRICS'16*, Antibes Juan-Les-Pins, France, Jun. 2016, pp. 75–88.

[26] M. Ghobadi, R. Mahajan, A. Phanishayee, N. Devanur, J. Kulkarni, G. Ranade, P.-A. Blanche, H. Rastegarfar, M. Glick, and D. Kilper, "ProjecToR: Agile reconfigurable data center interconnect," in *Proc. SIGCOMM'16*, Florianopolis, Brazil, Aug. 2016, pp. 216–229.

[27] B. Chen, J. Zhu, S. Zhang, W. Sun, and W. Hu, "Performances of traffic offloading in data center networks with steerable free-space optical communications," *IEEE/ACM Trans. Netw.*, vol. 32, no. 3, pp. 2189–2204, 2024.

[28] S.-H. Tseng, B. Bai, and J. C. Lui, "Hybrid circuit/packet network scheduling with multiple composite paths," in *Proc. IEEE INFOCOM'18*, Honolulu, HI, USA, Apr. 2018, pp. 882–890.

[29] J. Zheng, Z. Du, Z. Zha, Z. Yang, X. Gao, and G. Chen, "Learning to configure converters in hybrid switching data center networks," *IEEE/ACM Trans. Netw.*, vol. 32, no. 1, pp. 520–534, 2024.

[30] M. Alizadeh, A. Greenberg, D. A. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta, and M. Sridharan, "Data center TCP (DCTCP)," in *Proc. SIGCOMM'10*, New Delhi, India, Aug. 2010, pp. 63–74.

[31] T. Benson, A. Akella, and D. A. Maltz, "Network traffic characteristics of data centers in the wild," in *Proc. IMC'10*, Melbourne, Australia, Nov. 2010, pp. 267–280.

[32] Y. Chen, S. Jain, V. K. Adhikari, Z.-L. Zhang, and K. Xu, "A first look at inter-data center traffic characteristics via Yahoo! datasets," in *Proc. IEEE INFOCOM'11*, Shanghai, China, Apr. 2011, pp. 1620–1628.

[33] "Xilinx Virtex-II series FPGAs," accessed on Jun. 8, 2024. [Online]. Available: https://www.xilinx.com/publications/matrix/virtexmatrix.pdf

[34] "$P4_{16}$ Intel@ Tofino™ native architecture - public version," Apr. 2021. [Online]. Available: https://raw.githubusercontent.com/barefootnetworks/Open-Tofino/master/PUBLIC_Tofino-Native-Arch.pdf

[35] C. Semeria, "Supporting differentiated service classes: Queue scheduling disciplines," Juniper Networks, White Paper, Dec. 2001. [Online]. Available: http://users.jyu.fi/~timoh/kurssit/verkot/scheduling.pdf

[36] N. K. Sharma, C. Zhao, M. Liu, P. G. Kannan, C. Kim, A. Krishnamurthy, and A. Sivaraman, "Programmable calendar queues for high-speed packet scheduling," in *Proc. NSDI'20*, Santa Clara, CA, USA, Feb. 2020, pp. 685–700.

[37] V. Jacobson, "Congestion avoidance and control," in *Proc. SIGCOMM'88*, Stanford, CA, USA, Sep. 1988, pp. 314–329.

[38] V. Vasudevan, A. Phanishayee, H. Shah, E. Krevat, D. G. Andersen, G. R. Ganger, G. A. Gibson, and B. Mueller, "Safe and effective fine-grained TCP retransmissions for datacenter communication," in *Proc. SIGCOMM'09*, Barcelona, Spain, Aug. 2009, pp. 303–314.

[39] D. Zhuo, M. Ghobadi, R. Mahajan, K.-T. Förster, A. Krishnamurthy, and T. Anderson, "Understanding and mitigating packet corruption in data center networks," in *Proc. SIGCOMM'17*, Los Angeles, CA, USA, Aug. 2017, pp. 362–375.

[40] T. Yang, H. Zhang, J. Li, J. Gong, S. Uhlig, S. Chen, and X. Li, "Heavykeeper: An accurate algorithm for finding top-k elephant flows," *IEEE/ACM Trans. Netw.*, vol. 27, no. 5, pp. 1845–1858, 2019.

[41] L. Gu, Y. Tian, W. Chen, Z. Wei, C. Wang, and X. Zhang, "Per-flow network measurement with distributed sketch," *IEEE/ACM Trans. Netw.*, vol. 32, no. 1, pp. 411–426, 2024.

[42] "ns-3: A discrete-event network simulator for internet systems," accessed on Oct. 15, 2023. [Online]. Available: https://www.nsnam.org/

[43] M. Alizadeh, S. Yang, M. Sharif, S. Katti, N. McKeown, B. Prabhakar, and S. Shenker, "pFabric: Minimal near-optimal datacenter transport," in *Proc. SIGCOMM'13*, Hong Kong, China, Aug. 2013, pp. 435–446.

[44] G. Cormode and S. Muthukrishnan, "An improved data stream summary: the count-min sketch and its applications," *J. of Algorithms*, vol. 55, no. 1, pp. 58–75, 2005.

[45] T. Henderson, S. Floyd, A. Gurtov, and Y. Nishida, "The NewReno modification to TCP's fast recovery algorithm," RFC 6582, Apr. 2012. [Online]. Available: https://www.rfc-editor.org/info/rfc6582

[46] "ns-3: Congestion control algorithms," accessed on Oct. 15, 2023. [Online]. Available: https://www.nsnam.org/doxygen/da/da5/group__congestion_ops.html

[47] L. Xu, K. Harfoush, and I. Rhee, "Binary increase congestion control (BIC) for fast long-distance networks," in *Proc. IEEE INFOCOM'04*, Hong Kong, China, Nov. 2004, pp. 2514–2524.

[48] S. Ha, I. Rhee, and L. Xu, "CUBIC: a new TCP-friendly high-speed TCP variant," *ACM Oper. Syst. Rev.*, vol. 42, no. 5, pp. 64–74, 2008.

[49] S. Floyd, "HighSpeed TCP for large congestion windows," RFC 3649, Dec. 2003. [Online]. Available: https://www.rfc-editor.org/info/rfc3649

[50] D. Leith and R. Shorten, "H-TCP: TCP for high-speed and long-distance networks," in *Proc. PFLDnet'04*, Argonne, IL, USA, Feb. 2004, pp. 1–16.

[51] C. Caini and R. Firrincieli, "TCP Hybla: a TCP enhancement for heterogeneous networks," *Int. J. Satell. Commun. Netw.*, vol. 22, no. 5, pp. 547–566, 2004.

[52] K. Fall and S. Floyd, "Simulation-based comparisons of Tahoe, Reno and SACK TCP," *SIGCOMM CCR*, vol. 26, no. 3, pp. 5–21, 1996.

[53] T. Kelly, "Scalable TCP: Improving performance in highspeed wide area networks," *SIGCOMM CCR*, vol. 33, no. 2, pp. 83–91, 2003.

[54] C. P. Fu and S. Liew, "TCP Veno: TCP enhancement for transmission over wireless access networks," *IEEE J. Sel. Areas Commun.*, vol. 21, no. 2, pp. 216–228, 2003.

[55] S. Mascolo, C. Casetti, M. Gerla, M. Y. Sanadidi, and R. Wang, "TCP Westwood: Bandwidth estimation for enhanced transport over wireless links," in *Proc. MobiCom'01*, Rome, Italy, Jul. 2001, pp. 287–297.

[56] S. Shalunov, G. Hazel, J. Iyengar, and M. Kuehlewind, "Low extra delay background transport (LEDBAT)," RFC 6817, Dec. 2012. [Online]. Available: https://www.rfc-editor.org/info/

[57] L. S. Brakmo, S. W. O'Malley, and L. L. Peterson, "TCP Vegas: new techniques for congestion detection and avoidance," in *Proc. SIGCOMM'94*, London, UK, Aug. 1994, pp. 24–35.

[58] S. Liu, T. Başar, and R. Srikant, "TCP-Illinois: a loss and delay-based congestion control algorithm for high-speed networks," *Perform. Eval*, vol. 65, no. 6-7, pp. 417–440, 2008.

[59] N. Cardwell, Y. Cheng, C. S. Gunn *et al.*, "BBR: Congestion-based congestion control," *ACM Queue*, vol. 14, no. 5, pp. 20–53, 2016.

[60] A. Baiocchi, A. P. Castellani, and F. Vacirca, "YeAH-TCP: Yet another highspeed TCP," in *Proc. PFLDnet'06*, Nara, Japan, Feb. 2006, pp. 37–42.

[61] V. Jain, T. R. Henderson, S. K. S., and M. P. Tahiliani, "Data Center TCP in ns-3: Implementation, validation and evaluation," in *Proc. Workshop on ns-3 (WNS3'20)*, Gaithersburg, MA, USA, Jun. 2020, pp. 65–72.

[62] S. Das, A. Rahbar, X. C. Wu, Z. Wang, W. Wang, A. Chen, and T. S. E. Ng, "Shufflecast: An optical, data-rate agnostic, and low-power multicast architecture for next-generation compute clusters," *IEEE/ACM Trans. Netw.*, vol. 30, no. 5, pp. 1970–1985, 2022.

[63] S. Vargaftik, K. Barabash, Y. Ben-Itzhak, O. Biran, I. Keslassy, D. Lorenz, and A. Orda, "Composite-path switching," in *Proc. CoNEXT'16*, Irvine, CA, USA, Dec. 2016, pp. 329–343.

[64] Z. Li and H. Shen, "Co-scheduler: A coflow-aware data-parallel job scheduler in hybrid electrical/optical datacenter networks," *IEEE/ACM Trans. Netw.*, vol. 30, no. 4, pp. 1599–1612, 2022.

[65] X. Wang, H. Shen, and H. Tian, "Scheduling coflows in hybrid optical-circuit and electrical-packet switches with performance guarantee," *IEEE/ACM Trans. Netw.*, vol. 32, no. 3, pp. 2299–2314, 2024.

[66] K. Nichols and V. Jacobson, "Controlling queue delay," *ACM Queue*, vol. 10, no. 5, 2012.

[67] R. Pan, P. Natarajan, C. Piglione, M. S. Prabhu, V. Subramanian, F. Baker, and B. VerSteeg, "PIE: A lightweight control scheme to address the bufferbloat problem," in *Proc. HPSR'13*, Taipei, China, Jul. 2013, pp. 148–155.

[68] K. D. Schepper, O. Bondarenko, I.-J. Tsang, and B. Briscoe, "PI2: A linearized AQM for both classic and scalable TCP," in *Proc. CoNEXT'16*, Irvine, CA, USA, Dec. 2016, pp. 105–119.

[69] B. Turkovic and F. Kuipers, "P4air: Increasing fairness among competing congestion control algorithms," in *Proc. IEEE ICNP'20*, Madrid, Spain, Oct. 2020, pp. 1–12.

[70] W. Chen, Y. Tian, X. Yu, B. Zheng, and X. Zhang, "Enhancing fairness for approximate weighted fair queueing with a single queue," *IEEE/ACM Trans. Netw.*, 2024, early access, doi: 10.1109/TNET.2024.3399212.

[71] A. Kaufmann, T. Stamler, S. Peter, N. K. Sharma, A. Krishnamurthy, and T. Anderson, "TAS: TCP acceleration as an OS service," in *Proc. EuroSys'19*, Dresden, Germany, Mar. 2019, pp. 1–16.

[72] Y. Moon, S. Lee, M. A. Jamshed, and K. Park, "AccelTCP: Accelerating network applications with stateful TCP offloading," in *Proc. NSDI'20*, Santa Clara, CA, USA, Feb. 2020, pp. 77–92.

[73] M. Arghavani, H. Zhang, D. Eyers, and A. Arghavani, "SUSS: Improving TCP performance by speeding up slow-start," in *Proc. SIGCOMM'24*, Sydney, Australia, Aug. 2024, pp. 151–165.

Wei Chen received the bachelor's degree in computer science from University of Science and Technology of China (USTC), Hefei, China, in 2020. He is currently pursuing the Ph.D. degree with the School of Computer Science and Technology, USTC. His research interests include programmable networks and network traffic scheduling.

Ye Tian received the bachelor's degree in electronic engineering and the master's degree in computer science from University of Science and Technology of China (USTC), Hefei, China, in 2001 and 2004, respectively, and the Ph.D. degree from the Department of Computer Science and Engineering, The Chinese University of Hong Kong, Hong Kong, China, in 2007. He joined USTC in 2008 and is currently an Associate Professor with the School of Computer Science and Technology, USTC. His research interests include programmable networks, network traffic scheduling, and network measurement. He has published over 80 papers and co-authored a research monograph published by Springer. He is the winner of the Wilkes Best Paper Award of Oxford The Computer Journal in 2016. He is a member of the IEEE.

Xinming Zhang received the BE and ME degrees in electrical engineering from China University of Mining and Technology, Xuzhou, China, in 1985 and 1988, respectively, and the PhD degree in computer science and technology from the University of Science and Technology of China (USTC), Hefei, China, in 2001. Since 2002, he has been with the faculty of USTC, where he is currently a Professor with the School of Computer Science and Technology. From September 2005 to August 2006, he was a visiting Professor with the Department of Electrical Engineering and Computer Science, Korea Advanced Institute of Science and Technology, Daejeon, Korea. His research interest includes wireless networks, deep learning, and intelligent transportation. He has published more than 100 papers. He won the second prize of Science and Technology Award of Anhui Province of China in Natural Sciences in 2017. He won the awards of Top reviewers (1%) in Computer Science & Cross Field by Publons in 2019. He is a senior member of the IEEE.