Hop-On Hop-Off Routing

A Fast Tour across the Optical Data Center Network for Latency-Sensitive Flows

Jialong Li
Max Planck Institute for Informatics

Yiming Lei
Max Planck Institute for Informatics

Federico De Marchi
Saarland University

Raj Joshi
National University of Singapore

Balakrishnan Chandrasekaran
Vrije Universiteit Amsterdam

Yiting Xia
Max Planck Institute for Informatics

ABSTRACT

Optical data center networks show promise to serve as the next-generation cloud infrastructure especially with their cost and power benefits. The need to set up dedicated optical circuits between end-points before they can exchange data, however, delays latency-sensitive ("mice") flows. We find the state-of-the-art solution to reducing flow latency produces sub-optimal paths. To address this issue, we leverage programmable switches to realize Hop-On Hop-Off (HOHO) routing, where mice flows are forwarded along the minimal-latency paths. We prove the optimality and robustness of our algorithm and sketch an implementation on programmable switches. In our packet-level simulations, HOHO routing reduces the flow-completion times for mice flows by up to 35% and the average path length by 15% compared to the state-of-the-art solution.

CCS CONCEPTS

• **Networks** → **Data center networks**; **Programmable networks**; **Routing protocols**.

KEYWORDS

Optical data center networks, routing, programmable switches

ACM Reference Format:
Jialong Li, Yiming Lei, Federico De Marchi, Raj Joshi, Balakrishnan Chandrasekaran, and Yiting Xia. 2022. Hop-On Hop-Off Routing: A Fast Tour across the Optical Data Center Network for Latency-Sensitive Flows. In *6th Asia-Pacific Workshop on Networking (APNet 2022), July–2, 2022, Fuzhou, China.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3542637.3542647

1 INTRODUCTION

Data-center-network (DCN) designs have largely benefited from the Moore's law for networking—the bandwidth of electrical switches doubles every two years at the same cost and power [25]. As this bandwidth scaling is slowing down, the networking community has started exploring high-radix passive optical network interconnects, which have lower per-port cost and consume less power than electrical switches. The latest optical-DCN designs deliver up to 4-times the bandwidth [18] and consume only 23% – 26% power [2] of a cost-equivalent electrical DCN.

APNet '22, July 1–2, 2022, Fuzhou, China

ACM ISBN 978-1-4503-9748-3/22/07.
https://doi.org/10.1145/3542637.3542647

A typical optical DCN fabric comprises of a number of optical switches that interconnect electrical top-of-rack switches (ToRs) and end servers (refer Fig. 1). The fabric uses circuit switching to establish dedicated optical circuits that are time-shared amongst the different ToR pairs. The delays incurred in establishing the circuits, however, substantially affect latency-sensitive traffic (or "mice" flows) that then use these circuits. In this paper, we focus on minimizing the impact of these delays on latency-sensitive traffic.

Prior attempts at solving this problem either use an electrical-optical hybrid network and send mice flows over the "always-on" electrical network [4, 26], or reduce the circuit-establishment delays using novel optical switching hardware [2]. The electrical-optical dual-fabric, however, doubles the deployment and maintenance costs of DCNs, while the latter requires extensive customizations to commercial network devices and the standard network stack, e.g., to adapt to the 1 ns optical switching speed in Sirius [2]. In contrast to such prior work, we propose a simple solution that leverages programmable switches: a routing algorithm with the specific objective of accelerating mice flows.

The idea of leveraging routing to accelerate latency-sensitive flows has been used in prior work, albeit within a narrow scope. Opera pursued a meticulous co-design of the optical network topology and routing for guaranteeing that mice flows *always* have (multi-hop) optical paths available via intermediate ToRs [18]. Similar to prior designs, Opera assumes, however, that packets must be buffered on end servers, as optical switches are bufferless. As soon as an optical path is available, packets hop on that path and ride it until the destination. They cannot hop off at intermediate ToRs even if a different optical path later offers an earlier arrival time (at the destination). Routing in Opera is, hence, sub-optimal: It searches for **non-stop paths**, rather than **the fastest paths**. *We offer support for packets to "hop-off" at ToRs by rethinking packet buffering on ToRs.*

Buffering at ToRs was deemed impossible due to the limited packet buffer on switches, the difficulty in synchronizing switches to coordinate with optical circuit configurations, and the lack of processing logic for scheduling packet transmissions at precise times. Recent programmable switches offer rich functionalities to clear these technical obstacles. Switches can, for example, provide temporal buffering for a small number of packets [5, 21], be time-synchronized at nanosecond-level precision [12], and provide time-based scheduled packet transmission via calendar queues [24].

We exploit the recent technological innovations in programmable switches to realize a novel routing algorithm for minimizing the delays experienced by latency-sensitive flows and summarize our contributions as follows: (a) we present a Hop-On Hop-Off (HOHO)

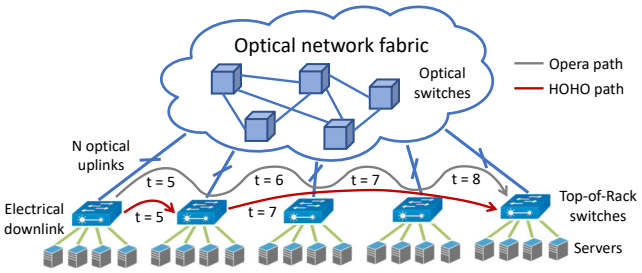

Figure 1: Illustration of an optical DCN where ToRs connect to an optical network fabric. If it takes one unit of time to traverse one hop, a packet leaving the source ToR at $t = 5$ (the notation shows departure times) reaches the target at $t = 9$ in Opera, which only supports non-stop paths. In HOHO, in contrast, the same packet waits at the first hop from $t = 6$ to $t = 7$ to choose the best path and arrive earlier at $t = 8$.

routing algorithm that provides the fastest paths—packets can "hop on" and "hop off" at intermediate ToRs to select the best optical paths that minimize their arrival time at the destination; (b) we prove the optimality and robustness of the HOHO routing algorithm, and sketch its implementation on programmable switches, including the time synchronization, routing lookup, and packet buffering mechanisms; (c) in our packet-level simulations with real DCN traffic, HOHO routing reduces the flow completion times (FCTs) of latency-sensitive flows by up to 35% and reduces the average path length by 15% compared to Opera. HOHO routing uses at most 7 queues per egress port and a packet buffer of about 3.24 MB, which is far below the capacity limit of commercial switch ASICs [7, 22].

2 BACKGROUND AND OVERVIEW

An optical DCN fabric (Fig. 1) uses circuit switching to construct dedicated optical circuits between different ToR pairs. The optical switches continuously change the circuits to interconnect different ToR pairs, and each circuit lasts for a fixed interval of time, called a *time slice*. The ToRs connected by a circuit have exclusive access to it (i.e., no contention with other ToRs) during the time slice. With today's mainstream optical switches, circuit establishment incurs delays varying between several nanoseconds to tens of microseconds [1, 2, 19]. A sequence of time slices, each connecting some subset of ToR pairs, constitutes an *optical schedule*. The switch repeats the schedule continuously and guarantees that each ToR pair is assigned at least one circuit per repetition (or *cycle*).

The goal of HOHO routing is to forward a packet from a source to a destination ToR via the *fastest path*. We define the fastest path in an optical DCN as the path that requires the minimum number of time slices. Depending on when a packet arrives at a ToR (i.e., its *arrival time slice*) and the optical schedule, the fastest path may "hop" through intermediate ToRs. HOHO routing consists of an offline routing algorithm (§3), which computes the fastest paths, and a runtime system (§4), which orchestrates packet forwarding along these paths. The HOHO routing algorithm is agnostic to optical DCN architectures and is general to a wide range of time slice durations. Given a cyclic optical schedule, the offline routing algorithm first computes the fastest paths for all source-destination ToR pairs, for all corresponding arrival time slices. The full paths are then converted into next-hop lookup tables for ease of implementation in the switch dataplane. Logically, for each ToR, there is a next-hop lookup table per arrival time slice. At run time, when a ToR receives a packet within a particular time slice, it looks up in the next-hop lookup table corresponding to that (arrival) time slice to get the egress port and the *send time slice* within which to transmit the packet. If the send time slice is later than the arrival time slice, the switch temporarily buffers the packet until the *send slice*, i.e., the time slice when the packet should be transmitted.

In designing HOHO routing, we assume that packets always arrive at the beginning of a time slice and that there is no queuing delay at the ToRs. These assumptions effectively decouple the design of the static routing algorithm and the runtime on-switch system. In practice, if breaking these assumptions renders a packet to miss the planned send time slice to reach the next hop, we perform run-time adjustments to match it to the next time slice. We prove that this mechanism finds the next optimal path (§3.2).

The runtime system on ToRs (§4) detects upcoming *slice-miss events* based on a fair estimation of the packet egress time from the packet arrival time and the queuing delay. The cost function in the HOHO routing algorithm only takes transmission latency as the cost at the moment. To reduce slice-miss events, we could revise this cost function, for example, to include queueing delays. We then need a reasonable measurement or estimation of queuing delays across different paths network-wide, which can be collected through network telemetry. Since the tradeoff between performance improvement and system complexity is, debatably, unclear, we leave this discussion to future work.

3 HOP-ON HOP-OFF ROUTING

In this section, we describe the design of the HOHO routing algorithm and prove three critical properties of the algorithm. The proofs provide evidence that it is a sound decision to implement the algorithm as next-hop lookup with run-time adjustments on the ToR system (§2).

3.1 Algorithm Design

The (optical) circuits in an optical DCN are analogous to "buses": the circuits (buses) transport packets (people) from a source ToR to a destination ToR. The time slices of circuits are simply "deadlines" to get on the "buses". The earliest time to arrive at a destination ToR is (as illustrated in Fig. 2) completely determined by the earliest time slice of the last-hop ToR, i.e., when the "bus" from the last "stop" departs for the destination (e.g., `t=5` in Fig. 2). To find the fastest path, we must find the earliest "bus" at the last hop to the destination (*step 1*). We then plan an "itinerary" from the source ToR to the last-hop ToR that satisfies the "deadlines" for making all the "transitions" (*step 2*): arriving at the next-hop ToR either earlier than the time slice to hop onward (e.g., waiting for the next "bus") or in the same time slice (e.g., being on time for the next "bus"). In Fig. 2, a packet arriving from S→B in the `t=3` slice to reach D must, for instance, wait at B for 2 time units, whereas one from G→B in the `t=5` slice is "on time" to use the B→D circuit. Once we choose the last-hop ToR in step 1, the arrival time does not

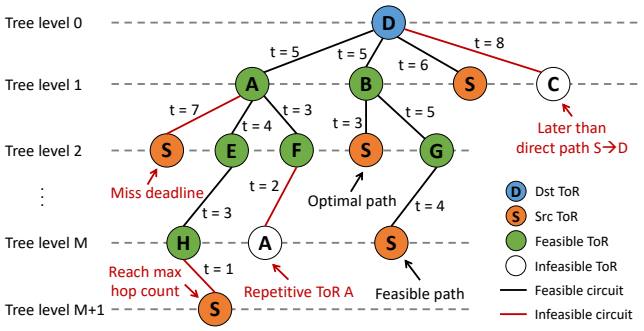

Figure 2: Illustration of the backtracking algorithm for HOHO routing (Alg. 1). We denote the time slices of optical circuits as absolute values, with the arrival time slice being $t = 0$. The destination calls Routing to find the earliest last hops (A and B with $t = 5$), which then call Subpath to find the shortest feasible path through them from the source (S→B→D). We prune out paths violating various constraints (see explanations in red) from this backtracking search.

change no matter how complicated the "itinerary" is in step 2, albeit we prefer a shorter path with fewer "transitions". Following this intuition, we design a *backtracking* algorithm (Alg. 1) for HOHO routing that comprises two procedures: Routing and Subpath, which correspond to steps 1 and 2, respectively.

For a packet that arrives at the source ToR in a particular time slice, the Routing procedure finds the fastest optical path to the destination ToR. It finds the last-hop ToR that provides the earliest arrival at the destination ToR, by sorting the time slices of all candidate ToRs connecting to this destination (*line 2*). For each candidate ToR, it calls the Subpath procedure to find a feasible sub-path from the source ToR (*line 20*). The procedure exits on finding the first valid path (*line 6*) or when the search ends (*line 11*). Since each ToR pair is guaranteed a circuit in the optical schedule (refer §2), Routing will always find a path—the direct path (S→D in Fig. 2) in the worst case, if no faster path exists. When multiple fastest paths (via A and B in Fig. 2) exist, the shortest path is chosen (*lines 9-10*).

The Subpath procedure finds a feasible sub-path from the source ToR to an intermediate ToR recursively. The procedure can terminate in two ways: (i) when it fails (*lines 13-14*) to find a path of length at most the maximum hop count, e.g., S→H→E→A→D in Fig. 2, or (ii) when it finds a connection from the source ToR and its time slice can make the "deadline" for the next-hop transmission (*lines 15-16*). In Fig. 2, for instance, S→B→D meets this condition, as the time slice `t=3` for S→B is earlier than the time slice `t=5` for B→D, while S→A→D violates this condition. No matter how many hops are traversed, the path must start from the source ToR. So, a path is found if and only if the source ToR is directly connected to the current intermediate ToR. Otherwise, Subpath calls itself to search onward to other intermediate ToRs not already in the sub-path and finds feasible sub-paths that constantly meet "deadlines" (*lines 18-22*). If Subpath find multiple feasible sub-paths, we select the shortest one (*line 23*). In Fig. 2, A→F→A→D is filtered for repetitive "A"s, and S→B→D is chosen ultimately because it is shorter than the other feasible path S→G→B→D.

Algorithm 1 Hop-On Hop-Off Routing Algorithm

Require:
$M \leftarrow$ max hop count
$src, dst \leftarrow$ source ToR, destination ToR
$S_0 \leftarrow$ the packet arrival time slice at src
$S_{(s,d)} \leftarrow$ the earliest time slice when ToR s and ToR d are connected, where $S_{(s,d)} \geq S_0$ must hold
▷ Find the fastest path per ToR pair per time slice
1: **procedure** Routing(src, dst, S_0)
2: Sort all ToRs by $S_{(tor,dst)}$ in ascending order
3: $path = \emptyset$, $min\_time = S_{(ToR[0],dst)}$, $min\_hop = \infty$
4: **for** each tor in ToRs **do**
5: **if** $S_{(tor,dst)} > min\_time$ **and** $path \neq \emptyset$ **then**
6: **return** $path$
7: $min\_time = S_{(tor,dst)}$
8: $path' =$ Subpath($src, tor, S_{(tor,dst)}, 1, \{dst\}$)
9: **if** $path' \neq \emptyset$ **and** $hop(path') < min\_hop$ **then**
10: $path = path'$, $min\_hop = hop(path')$
11: **return** $path$
▷ Find the shortest feasible subpath through an intermediate ToR
12: **procedure** Subpath($src, tor, S, level, subpath$)
13: **if** $level > M$ **then**
14: **return** $\emptyset$
15: **if** $S_{(src,tor)} \leq S$ **then**
16: **return** $src + subpath$
17: $feasible = \{\}$
18: **for** each tor' in ToRs **and** tor' not in $subpath$ **do**
19: **if** $S_{(tor',tor)} \leq S$ **then**
20: $p =$ Subpath($src, tor', S_{(tor',tor)}, level+1, tor+subpath$)
21: **if** $p \neq \emptyset$ **then**
22: $p \rightarrow feasible$
23: **return** $shortest(feasible)$ **or** $\emptyset$

3.2 Algorithm Properties

Below, we discuss and prove HOHO's three key properties.

Property 1: The HOHO routing algorithm is optimal: the chosen path is the shortest that leads to the minimal latency.

Proof. Let p be the selected path whose last-hop ToR to dst is r and path length is l. The time slice of the optical connection between r and dst is s. If there exists a better path $\hat{p}$ from src to dst with the last-hop ToR $\hat{r}$ at slice $\hat{s}$ and the path length is $\hat{l}$, then either $s > \hat{s}$, or $s = \hat{s}$ and $l > \hat{l}$. We prove by contradiction:

Case I: $s > \hat{s}$. In Routing, last-hop ToRs are traversed by their time slices to dst ascendingly. So, $\hat{p}$ must be found earlier than p, which is a contradiction.

Case II: $s = \hat{s}$ and $l > \hat{l}$. When Routing breaks the tie on the same time slice, $\hat{p}$ would overwrite p and be chosen (*lines 9-10*), which is a contradiction. □

HOHO routing produces full paths, including every hop along the way, but the routing lookup on each intermediate ToR is based only on the immediate next hop. Now, we prove that this implementation preserves the optimal paths.

Property 2: Per-hop lookups yield the optimal path.

Proof. Let p be the selected path whose first-hop ToR from src is r, last-hop ToR to dst is r', the optical connection between src

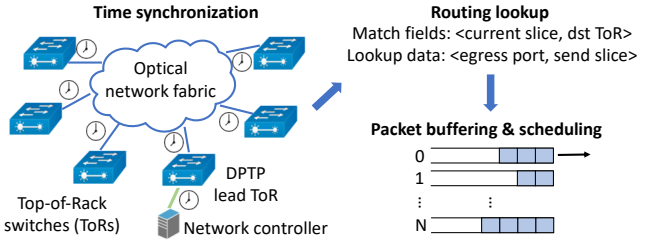

Figure 3: We use DPTP [12] for time synchronization, implement HOHO routing lookup using a custom match-action table, and use calendar queues [24] to achieve packet buffering and scheduled transmission.

and r is at time slice s, the connection between r' and dst is at slice s', and the path length is l. The residual path from r to dst is $p' = p - src$, the arrival time at r is s, and the path length is $l' = l - 1$. We prove p' is an optimal path for Routing(r, dst, s).

If there exists a better path $\hat{p'}$ than p' from r to dst at slice s, whose last-hop ToR to dst is $\hat{r'}$, the optical connection between $\hat{r'}$ to dst is $\hat{s'}$, and the path length is $\hat{l'}$, then $s' > \hat{s'}$, or $s' = \hat{s'}$ and $l' > \hat{l'}$. For either case, because $\hat{p'}$ starts at slice s where src and r are connected, there must be a path $\hat{p} = src + \hat{p'}$ from src to dst, which arrives at dst at slice $\hat{s'}$, and the path length is $\hat{l} = \hat{l'} + 1$. Comparing $\hat{p}$ to p, we have $s' > \hat{s'}$, or $s' = \hat{s'}$ and $l > \hat{l}$. So, $\hat{p}$ is better than p, which contradicts Property 1 that the chosen path p is optimal.

Now that p' is optimal, since Routing selects a single path out of the feasible paths, Routing(r, dst, s) may return a different optimal path $\hat{p'}$ equivalent to p', that is $s' = \hat{s'}$ and $l' = \hat{l'}$. Then for the full paths p and $\hat{p}$ from src, $s' = \hat{s'}$ and $l = \hat{l}$. So, $\hat{p}$ is also optimal.

Repeating the above proof hop by hop until dst, we have hop-wise lookups produce the optimal path. □

If a packet misses its planned time slice, the switch system adjusts at runtime to reroute the packet by the next available time slice (§2). We prove our runtime adjustment is robust to find the next optimal path starting from the current ToR.

Property 3: Rerouting after missing a planned send time slice gives the next optimal path.

Proof. Let p be the optimal path from src to dst, R be the set of intermediate ToRs along p, and S be the time slices set for ToR connections of adjacent hops. Assume S_i is missed at R_i and the current time slice is S_c ($S_c > S_i$). By per-hop lookup, we get a path p' from R_i to dst at slice S_c. According to Property 2, p' is optimal w.r.t. the current time slice S_c. □

4 SWITCH SYSTEM SKETCH

In this section, we outline the system design to demonstrate the feasibility of realizing HOHO routing in practice.

For the HOHO routing scheme (§3) to work, each ToR requires three key functionalities. **(i) Time synchronization.** For an arriving packet, the next hop (egress port) determined by HOHO routing depends on the packet's arrival time slice. Therefore, each ToR switch would need to keep track of the current time slice by time synchronizing with the optical network fabric. **(ii) HOHO Routing lookup.** Each ToR needs to implement a routing table, which can match on the packet's arrival time slice and the destination ToR, and look up the *egress port* (next hop) and the send slice. **(iii) Packet buffering and time-scheduled transmission.** A ToR also needs to implement time-scheduled packet transmission such that each packet can be transmitted precisely within the send slice determined by HOHO routing.

Time synchronization. A controller machine usually reconfigures circuits on an optical DCN fabric as per the optical schedule. To synchronize the ToRs with the optical schedule, we plan to use the DPTP time synchronization protocol [12]. DPTP supports both switch-to-host and switch-to-switch synchronization in the dataplane, at the precision of tens of nanoseconds. The ToR connected to the network controller is designated as the *lead ToR* (see Fig. 3) and synchronizes with the network controller. During the bootstrapping stage of the optical DCN fabric, we make a circuit connection between the lead ToR and every other ToR individually to synchronize them one by one. After the initial synchronization, the optical fabric becomes operational and thereafter DPTP synchronization is performed with the lead ToR once every optical schedule cycle to correct for the clock drifts.

HOHO Routing lookup. For a given optical schedule, the HOHO routing algorithm is run offline per time slice per source-destination ToR pair. The output paths are then encoded as static routes for next-hop lookups in a HOHO routing table on each ToR. The HOHO routing table is a simple match-action table where the match fields are the packet's arrival time slice and the destination ToR while the lookup (action) data returned consists of the egress port and the send slice when the packet should be transmitted to the next hop. Note that HOHO routing generates per arrival time slice routing tables assuming that the packets always arrive at the beginning of a time slice and therefore can finish transmission in the same time slice. In practice, however, this is not always the case due to the pipeline processing and queuing delays on the switch as well as the serialization and propagation delays on the fiber. Fortunately, the next generation programmable switches such as Intel Tofino2 provide queuing delay information in the ingress pipeline [14], while the remaining delays are deterministic given a packet size. Therefore, appropriate correction can be applied to the arrival time slice of a packet such that the HOHO routing lookup is performed based on the time slice in which the packet would complete its transmission.

Packet buffering and time-scheduled transmission. If the send slice determined by the HOHO routing lookup is later than the current time slice, the packet must be buffered temporarily for time-scheduled transmission. To this end, we propose to build on top of the calendar queues framework [24] where a "calendar day" corresponds to a time slice and is mapped to a FIFO queue. The *rank* for a packet is simply the difference between the send slice and the current time slice, i.e., how far in the future the packet's transmission is scheduled. We perform queue rotation each time the current time slice is updated, which in turn is updated in a time-driven manner using the on-chip packet generator [11]. We keep track of the queue corresponding to the current time slice using a SRAM-based register, which is updated per queue rotation.

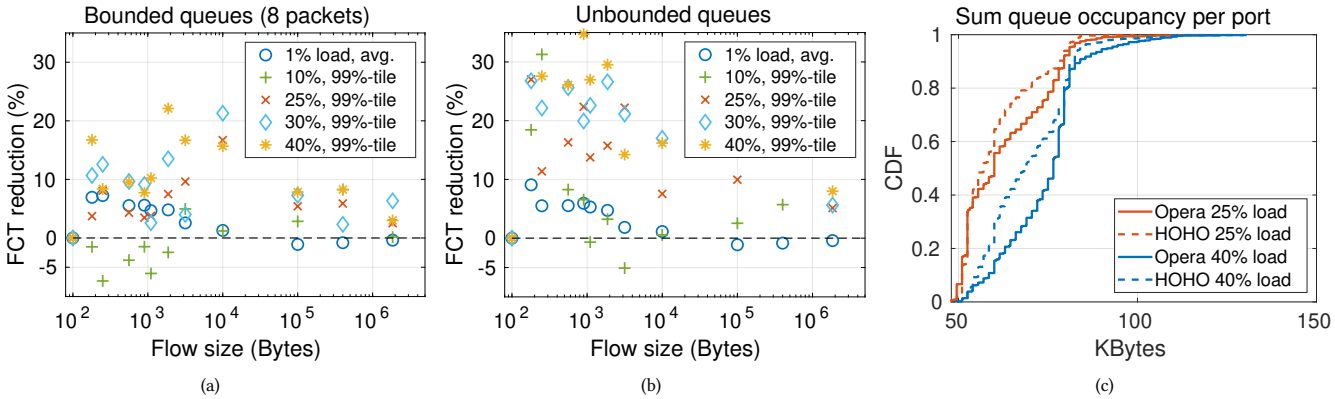

Figure 4: Performance comparison with Opera. (a) FCT reduction with 8 packets per queue. (b) FCT reduction with unbounded queues. (c) Queue occupancy distribution with unbounded queues.

Since packets are always dequeued from the queue corresponding to the current time slice, an enqueued packet would be dequeued during its send slice.

With Intel Tofino2, it is now possible to escalate queue priorities dynamically from the switch dataplane through queue pausing [14].

5 EVALUATION

We now compare the performance of HOHO routing with Opera, the latest optical DCN design optimized for mice flows. We conducted packet-level simulations using the Opera simulator [18] and extended the simulator with the routing algorithm (§3) and the runtime switch system (§4) for evaluating mice flows. For elephant flows, we retained the original direct-path routing and on-NIC packet buffering. To perform a fair comparison, we reused the experimental setup and flow classification heuristics in the Opera paper [18]. We simulated a network comprising 108 ToRs and 648 servers. Each ToR has 6 10 Gbps downlinks to servers and 6 10 Gbps uplinks to the optical DCN fabric containing 6 108-port optical switches. We used the data-mining traffic workload from Microsoft [8], scaled to a value between 1% and 40% load to match the link utilizations in production DCNs [23]. Opera uses the congestion control protocol (CCP) from NDP [9] with a cap of 8 packets per queue, and we use this configuration as such. We mostly stick to Opera's time slices of 60.5 µs, but vary the time slices later to demonstrate the flexibility of our solution.

Low latency. Fig. 4(a) shows that flow completion times (FCTs) for mice flows in HOHO routing are substantially smaller (up to 23%) than those in Opera. The improvements are also substantial for high traffic loads. Since HOHO routing is more effective than Opera at finding shorter paths, it alleviates packet queuing per ToR especially for heavy loads. The FCT reduction is amortized over the longer FCT for larger flows, which is consistent with our design purpose of improving mice flows. HOHO routing does not degrade the performance of elephant flows, albeit we omit elephant flows in Fig. 4(a) due to the neglible the difference in FCTs between HOHO routing and Opera for these flows.

Robustness to congestion. Opera typically generates long paths and it relies on NDP for congestion control. We removed the queue occupancy cap of 8 packets in NDP to mimic the scenario where such advanced CCPs are unavailable. Per Fig. 4(b), HOHO routing reduces the FCT by up to 35% in such scenarios. When we scrutinized the flows in Opera, we observed many packet drops: As queues build up, the queuing delay causes packets to miss the scheduled send slice and be sent over non-existent circuits. With shorter paths than Opera, HOHO routing reduces the queue occupancy in the first place. Furthermore, the runtime adjustment on the switch system (§4) detects the upcoming slice miss and postpones the packet to a later send slice rather than dropping it. This robustness to misses also demonstrates the benefit of per-hop lookups (§3.2).

Low queue occupancy. While Opera uses two separate queues for elephant and mice flows, HOHO uses N calendar queues as needed for mice flows and another queue for elephant flows. Fig. 4(c) plots the occupancy across all queues for mice flows (labelled "sum queue occupancy") per port in the unbounded queues setting. In general, HOHO routing and Opera both consume manageable queue space. HOHO routing has a noticeable reduction in the queue occupancy, regardless of the need to buffer packets during path transitions. In particular, the median queue occupancy in HOHO is 5% lower at 25% load and 10% lower at 40% load. These improvements stem from HOHO's short paths and highlight HOHO's potential to serve high traffic loads.

Flexible time slice duration. In Opera, as packets must stick to fixed (and usually long) paths, the time slice duration is lower-bounded by the worst-case one-way delay along the multi-hop paths. The paths in Opera can have up to 5 hops, and each hop queues up to 8 packets. The propagation and queuing delays require a minimal time slice duration of 60.5 µs, although the trend in optical switching delays indicates a shift from microseconds to nanoseconds. HOHO routing reduces the minimal slice duration to the one-hop delay, as packets can "hop off" optical paths freely. Considering empty queues and optical switching delays, the one-hop delay and thus the minimal achievable time slice duration in our 10 Gbps setup is 2.269 µs. However, we suggest using 12.1 µs time slices in our 10 Gbps setting to consider the queuing delay of 8 packets per queue as in Opera.

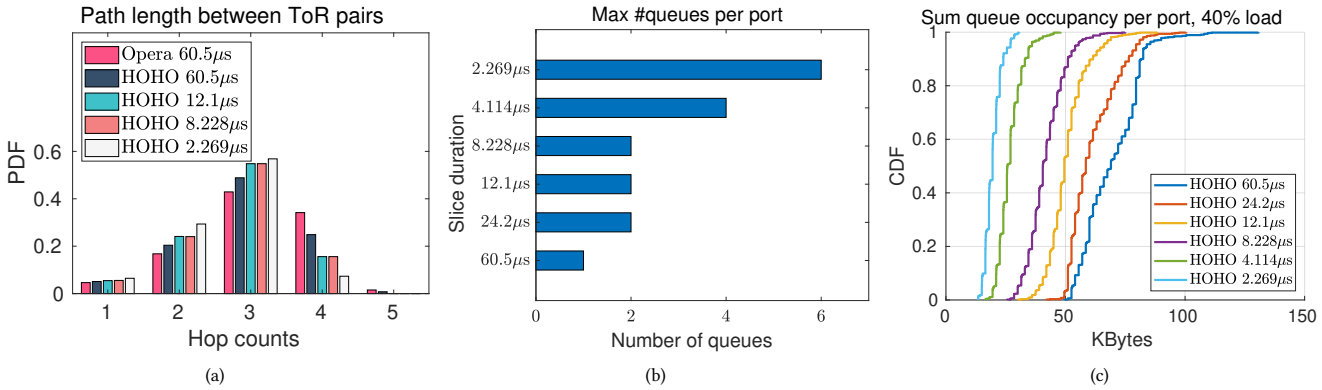

Figure 5: HOHO routing under varying time slot durations. (a) Path length distribution compared with Opera. (b) Maximum number of queues per port. (c) Queue occupancy with unbounded queues.

Short paths. Fig. 5(a) shows that HOHO routing yields shorter paths (i.e., fewer hops) than Opera. The advantage becomes more prominent as the time slice duration decreases, since for short time slices the benefits of waiting at a ToR for the next slice far outweigh the transmission delay over an extra hop. Notably, our suggested slice duration of 12.1 µs retains 99.99% of paths under 4 hops and 84.39% paths under 3 hops. It reduces the average path length from 3.11 in Opera to 2.80, and a slice duration of 2.269 µs reduces it further to 2.65.

Practically feasible. HOHO routing uses N calendar queues per port to buffer mice flows. Whether commercial switches have sufficient physical queues and packet buffer to hold the buffered packets is naturally a concern. Fig. 5(b) alleviates this concern by showing the maximum number of queues needed under different slice durations. HOHO routing needs 7 queues per port at most (including one queue for elephant flows), which is well below the capacity of today's commercial switch ASICs [6]. We plot in Fig. 5(c) the aggregate occupancy of all queues for mice flows under the highest (40% traffic) load with unbounded queues (as in Fig. 4). Long time slices results in higher queue occupancy, as it takes longer to clear packets for future slices. For the suggested 12.1 µs time slices, the average queue occupancy is 50.6 KB per port. With a high-end 128-port ToR with half ports connected to the optical fabric, we need a total buffer of size 3.24 MB. In the extreme case of 60.5 µs slices, the average queue occupancy is 69.4 KB per port, requiring 4.44 MB of total buffer. These sizes are significantly below the buffer size of commercial switch ASICs [22].

6 RELATED WORK

Several optical DCN architectures have been proposed to improve traditional electrical DCNs, e.g., to increase the network bandwidth, enable flexible topologies, and reduce deployment cost and power consumption [2–4, 16, 18–20, 26]. Few of these proposals aim at optimizing the flow latency [2, 4, 18, 26], and HOHO routing is a direct improvement over them. Unlike Opera [18], HOHO allows packets to "hop off" from the current path and choose a better one, resulting in shorter and faster paths than Opera in our simulations. We designed HOHO for optical-only DCNs and to be compatible with different optical hardware. It is thus easier and more economical to deploy HOHO than electrical-optical hybrid networks like Helios [4] and c-Through [26], and Sirius [2], which requires sweeping changes from optical transceivers to the network stack.

Our unconventional approach of packet buffering on ToRs is motivated by the recent successes of in-network computing [7, 10, 13, 15, 17, 27–30]. Particularly, we leverage DPTP [12] to synchronize ToRs with the optical network controller, and we leverage calendar queues [24] for temporal buffering and time-scheduled packet transmission. Once fully realized, HOHO routing will join this family of tools to enable radical network designs in the future.

7 CONCLUDING REMARKS

The delays incurred in setting up circuits in an optical DCN poses a significant barrier for supporting today's ultra-low latency cloud applications, including disaggregated computing, machine-learning inference, and cloud gaming. To address this issue, we presented the design of the Hop-On Hop-Off (HOHO) routing algorithm, sketched an implementation on programmable switches, and demonstrated the design's benefits through packet-level simulations. By showing the feasibility of and potential behind a radical (long deemed infeasible) idea—buffering packets on ToRs for optical DCNs—this paper opens up two new vistas of research. First, it urges the networking community to fundamentally alter the way we approach optical DCN designs and creates opportunities for novel architectures. Second, since HOHO routing's design is agnostic to optical hardware structure and time-slice durations, it can serve as a universal routing algorithm for comparative evaluations of different optical DCN architectures.

ACKNOWLEDGMENTS

We sincerely thank the authors of the Opera paper [18] for sharing their simulator implementation. This work would not have been possible without it. We are also grateful to the anonymous reviewers for their insightful feedback.

REFERENCES

[1] Agiltron. 2022. Optical Switches. https://agiltron.com/category/optical-switches/nanospeed-fiber-optical-switches/. [accessed on March 11, 2022].

[2] Hitesh Ballani, Paolo Costa, Raphael Behrendt, Daniel Cletheroe, Istvan Haller, Krzysztof Jozwik, Fotini Karinou, Sophie Lange, Kai Shi, Benn Thomsen, et al. 2020. Sirius: A flat datacenter network with nanosecond optical switching. In *Proceedings of SIGCOMM*.

[3] Kai Chen, Ankit Singla, Atul Singh, Kishore Ramachandran, Lei Xu, Yueping Zhang, Xitao Wen, and Yan Chen. 2012. OSA: An optical switching architecture for data center networks with unprecedented flexibility. In *Proceedings of NSDI*.

[4] Nathan Farrington, George Porter, Sivasankar Radhakrishnan, Hamid Hajabdolali Bazzaz, Vikram Subramanya, Yeshaiahu Fainman, George Papen, and Amin Vahdat. 2010. Helios: A hybrid electrical/optical switch architecture for modular data centers. In *Proceedings of SIGCOMM*.

[5] Swati Goswami, Nodir Kodirov, Craig Mustard, Ivan Beschastnikh, and Margo Seltzer. 2020. Parking packet payload with P4. In *Proceedings of CoNEXT*.

[6] Prateesh Goyal, Preey Shah, Naveen Kr Sharma, Mohammad Alizadeh, and Thomas E Anderson. 2019. Backpressure flow control. In *Proceedings of Workshop on Buffer Sizing*.

[7] Prateesh Goyal, Preey Shah, Kevin Zhao, Georgios Nikolaidis, Mohammad Alizadeh, and Thomas E. Anderson. 2022. Backpressure flow control. In *Proceedings of NSDI*.

[8] Albert Greenberg, James R Hamilton, Navendu Jain, Srikanth Kandula, Changhoon Kim, Parantap Lahiri, David A Maltz, Parveen Patel, and Sudipta Sengupta. 2009. VL2: A scalable and flexible data center network. In *Proceedings of SIGCOMM*.

[9] Mark Handley, Costin Raiciu, Alexandru Agache, Andrei Voinescu, Andrew W Moore, Gianni Antichi, and Marcin Wójcik. 2017. Re-architecting datacenter networks and stacks for low latency and high performance. In *Proceedings of SIGCOMM*.

[10] Theo Jepsen, Ali Fattaholmanan, Masoud Moshref, Nate Foster, Antonio Carzaniga, and Robert Soulé. 2020. Forwarding and routing with packet subscriptions. In *Proceedings of CoNext*.

[11] Raj Joshi, Ben Leong, and Mun Choon Chan. 2019. Timertasks: Towards time-driven execution in programmable dataplanes. In *Proceedings of SIGCOMM Posters and Demos*.

[12] Pravein Govindan Kannan, Raj Joshi, and Mun Choon Chan. 2019. Precise time-synchronization in the data-plane using programmable switching ASICs. In *Proceedings of SOSR*.

[13] Jonatan Langlet, Ran Ben-Basat, Sivaramakrishnan Ramanathan, Gabriele Oliaro, Michael Mitzenmacher, Minlan Yu, and Gianni Antichi. 2021. Zero-CPU collection with direct telemetry access. In *Proceedings of HotNets*.

[14] Jeongkeun Lee. 2020. Advanced congestion & flow control with programmable switches. In *P4 Expert Roundtable Series*. https://bit.ly/3J8x7fw

[15] Bojie Li, Gefei Zuo, Wei Bai, and Lintao Zhang. 2021. 1Pipe: Scalable total order communication in data center networks. In *Proceedings of SIGCOMM*.

[16] Yunpeng J. Liu, Peter X. Gao, Bernard Wong, and Srinivasan Keshav. 2014. Quartz: A new design element for low-latency DCNs. *SIGCOMM CCR* (2014).

[17] Zaoxing Liu, Hun Namkung, Georgios Nikolaidis, Jeongkeun Lee, Changhoon Kim, Xin Jin, Vladimir Braverman, Minlan Yu, and Vyas Sekar. 2021. Jaqen: A high-performance switch-native approach for detecting and mitigating volumetric DDoS attacks with programmable switches. In *Proceedings of USENIX Security*.

[18] William M Mellette, Rajdeep Das, Yibo Guo, Rob McGuinness, Alex C Snoeren, and George Porter. 2020. Expanding across time to deliver bandwidth efficiency and low latency. In *Proceedings of NSDI*.

[19] William M Mellette, Rob McGuinness, Arjun Roy, Alex Forencich, George Papen, Alex C Snoeren, and George Porter. 2017. Rotornet: A scalable, low-complexity, optical datacenter network. In *Proceedings of SIGCOMM*.

[20] George Porter, Richard Strong, Nathan Farrington, Alex Forencich, Pang Chen-Sun, Tajana Rosing, Yeshaiahu Fainman, George Papen, and Amin Vahdat. 2013. Integrating microsecond circuit switching into the data center. *SIGCOMM CCR* (2013).

[21] Ting Qu, Raj Joshi, Mun Choon Chan, Ben Leong, Deke Guo, and Zhong Liu. 2019. SQR: In-network packet loss recovery from link failures for highly reliable datacenter networks. In *Proceedings of ICNP*.

[22] Ting Qu, Raj Joshi, Mun Choon Chan, Ben Leong, Deke Guo, and Zhong Liu. 2019. SQR: In-network packet loss recovery from link failures for highly reliable datacenter networks. In *Proceedings of ICNP*.

[23] Arjun Roy, Hongyi Zeng, Jasmeet Bagga, George Porter, and Alex C Snoeren. 2015. Inside the social network's (datacenter) network. In *Proceedings of SIGCOMM*.

[24] Naveen Kr Sharma, Chenxingyu Zhao, Ming Liu, Pravein G Kannan, Changhoon Kim, Arvind Krishnamurthy, and Anirudh Sivaraman. 2020. Programmable calendar queues for high-speed packet scheduling. In *Proceedings of NSDI*.

[25] Arjun Singh, Joon Ong, Amit Agarwal, Glen Anderson, Ashby Armistead, Roy Bannon, Seb Boving, Gaurav Desai, Bob Felderman, Paulie Germano, et al. 2015. Jupiter rising: A decade of clos topologies and centralized control in Google's datacenter network. *SIGCOMM CCR* (2015).

[26] Guohui Wang, David G Andersen, Michael Kaminsky, Konstantina Papagiannaki, TS Eugene Ng, Michael Kozuch, and Michael Ryan. 2010. c-Through: Part-time optics in data centers. In *Proceedings of SIGCOMM*.

[27] Zhuolong Yu, Chuheng Hu, Jingfeng Wu, Xiao Sun, Vladimir Braverman, Mosharaf Chowdhury, Zhenhua Liu, and Xin Jin. 2021. Programmable packet scheduling with a single queue. In *Proceedings of SIGCOMM*.

[28] Yifan Yuan, Omar Alama, Jiawei Fei, Jacob Nelson, Dan R. K. Ports, Amedeo Sapio, Marco Canini, and Nam Sung Kim. 2022. Unlocking the power of inline floating-point operations on programmable switches. In *Proceedings of NSDI*.

[29] Chaoliang Zeng, Layong Luo, Teng Zhang, Zilong Wang, Luyang Li, Wenchen Han, Nan Chen, Lebing Wan, Lichao Liu, Zhipeng Ding, Xiongfei Geng, Tao Feng, Feng Ning, Kai Chen, and Chuanxiong Guo. 2022. Tiara: A scalable and efficient hardware acceleration architecture for stateful layer-4 load balancing. In *Proceedings of NSDI*.

[30] Kaiyuan Zhang, Danyang Zhuo, and Arvind Krishnamurthy. 2020. Gallium: Automated software middlebox offloading to programmable switches. In *Proceedings of SIGCOMM*.