Title
Integrating Microsecond Circuit Switching into the Data Center

Authors
Porter, George
Strong, Richard
Farrington, Nathan
et al.

Publication Date
2013-02-13

Peer reviewed

Integrating Microsecond Circuit Switching into the Data Center

George Porter, Richard Strong, Nathan Farrington*
Alex Forencich, Pang-Chen Sun, Tajana Rosing
Yeshaiahu Fainman, George Papen, Amin Vahdat†
University of California, San Diego *Facebook, Inc. †UCSD and Google, Inc.

Abstract

Recent proposals have employed optical circuit switching (OCS) to reduce the cost of data center networks. However, the relatively slow switching times (10–100 ms) assumed by these approaches, and the accompanying latencies of their control planes, has limited its use to only the largest data center networks with highly aggregated and constrained workloads. As faster switch technologies become available, designing a control plane capable of supporting them becomes a key challenge.

In this paper, we design and implement an OCS prototype capable of switching in 11.5 μs, and we use this prototype to expose a set of challenges that arise when supporting switching at microsecond time scales. In response, we propose a microsecond-latency control plane based on a circuit scheduling approach we call Traffic Matrix Scheduling (TMS) that proactively communicates circuit assignments to communicating entities so that circuit bandwidth can be used efficiently.

Categories and Subject Descriptors

C.2.1 [**Computer-Communication Networks**]: Network Architecture and Design—*circuit-switching networks, packet-switching networks, network topology*

General Terms

Design, Experimentation, Measurement, Performance

Keywords

Data Center Networks, Optical Networks

1. INTRODUCTION

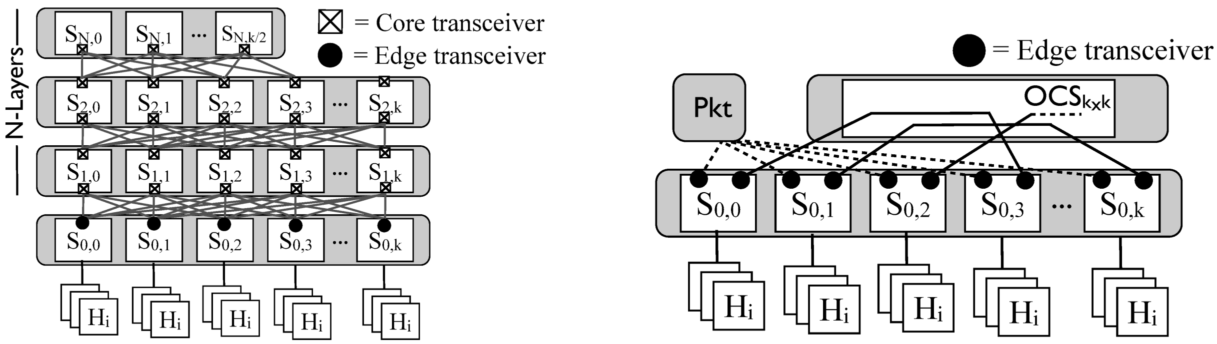
As the size and complexity of data center deployments grow, meeting their requisite bisection bandwidth needs is a challenge. Servers with 10 Gb/s link rates are common today, and 40 Gb/s NICs are already commercially available. At large scale, this translates into significant bisection bandwidth requirements. For a large data center with numerous, rapidly changing applications, supporting as close to full bisection bandwidth as practical is important, since ultimately application performance, and hence overall server utilization, may suffer if insufficient bandwidth is available. The result is that network complexity and expense is increasing.

To meet the required bandwidth demands, data center operators have adopted multi-layer network topologies [12] (e.g., folded Clos, or "FatTree" [1, 14]), shown in Figure 1(a). While these topologies scale to very high port counts, they are also a significant source of cost, due in part to the large amount of switches, optical transceivers, fibers, and power each of their layers requires. Recent efforts have proposed [6, 8, 24] using optical circuit switches (OCS) to deliver reconfigurable bandwidth throughout the network, reducing some of the expense of multi-layer scale-out networks, shown in Figure 1(b). A key challenge to adopting these proposals has been their slow reconfiguration time, driven largely by existing 3D-MEMS technology limitations. This reconfiguration time is dominated by two components: (1) the hardware switching time of the 3D-MEMS OCS (10–100 ms), and (2) the software/control plane overhead required to measure the communication patterns and calculate a new schedule (100ms to 1s). As a result, the resulting control plane is limited to supporting only highly aggregated traffic at the core of the network [8], or constrained applications with high traffic stability [24].

As optical switches become faster, deploying them more widely in data center networks requires a correspondingly faster control plane capable of efficiently utilizing short-lived circuits. The contribution of this paper is such a control plane. To gain experience with fast OCS switching, we start by designing and building a simple 24-port OCS prototype called Mordia,[1] which has a switch time of 11.5 μs. Mordia is built entirely with commercially available components, most notably 2D-based MEMS wavelength-selective switches (WSS). We use this prototype as a stand-in for future low-latency OCS devices.

Using the Mordia prototype as a starting point, we then identify a set of challenges involved in adopting ex-

[1] Microsecond Optical Research Data Center Interconnect Architecture



(a) A FatTree network topology

(b) A Hybrid network topology

Figure 1: A comparison of a scale-out, multi-layered FatTree network and a Hybrid electrical/optical network design. In the FatTree topology (a) each layer of switching incurs additional cost in electronics, core transceivers, fiber cabling, and power. In contrast, the Hybrid topology (b) requires only a single "layer" assuming that the OCS reconfiguration speed is sufficiently fast.

isting OCS control planes to microsecond-latency switching. In response to these challenges, we propose a new circuit scheduling approach called Traffic Matrix Scheduling (TMS). Instead of measuring long-lived, prevailing conditions and configuring the topology in response, TMS instead leverages application information and short-term demand estimates to compute short-term circuit schedules. TMS chooses these schedules to rapidly multiplex circuits across a set of end points, making use of the fast circuit switch time to reduce buffering and network delay. To obtain high circuit utilization, the computed schedules are communicated to top-of-rack switches (TOR) connected to Mordia, which adjust the transmission of packets into the network to coincide with the scheduled switch reconfigurations, with full knowledge of when bandwidth will be most available to a particular destination. In this way, both short and long flows can be offloaded into the OCS.

As a result, TMS can achieve 65% of the bandwidth of an identical link rate electronic packet switch (EPS) with circuits as short as 61μs duration, and 95% of EPS performance with 300-μs circuits using commodity hardware. Taken together, our work suggests that continuing to push down the reconfiguration time of optical switches and reducing the software and control overheads holds the potential to radically lower the cost for delivering high bisection bandwidth in the data center.

2. MOTIVATION: REDUCING NETWORK COST VIA FASTER SWITCHING

We have argued that decreasing the end-to-end reconfiguration time of optical circuit switching can reduce the cost and complexity of data center networks. We now examine some of the sources of these costs, and motivate the need for low-latency circuit switching.

Speed	Radix	Depth	# Nodes (x1000)	# Core Ports (x1000)
10G	48	5	498	3,484
	96	3	28	83
40G	16	7	33	360
		9	524	7,864
	24	5	16	109
		7	560	6,159

Table 1: The complexity of sample multi-layer, fully-provisioned, scale-out network topologies. Small-radix switches and link redundancy require more layers, and thus more switches and optical transceivers, which drives up their cost.

2.1 Multi-layer Switching Networks

While multi-layer switching topologies like FatTrees have been shown to support very large bisection bandwidths, they are a significant source of cost. We now argue that this cost is likely to increase, and that to stem this cost increase requires pushing optical circuit switching to a lower layer of the topology — which requires a likewise decrease in OCS switching latency.

Multi-layer packet-switched topologies are very flexible — any node can communicate with any other node on demand. However, they must be provisioned for worst-case communication patterns, which can require as many as five to nine layers in the largest networks, with each subsequent layer less utilized than the next in the common case. Each of these layers adds substantial cost in terms of the switch hardware, optical transceivers, fibers, and power.

Consider a scale-out data center network supporting M servers partitioned into racks (e.g., 20 to 40 servers per rack), and assume that the network is a FatTree. In general, an N-level FatTree built from k-radix switches can support $k^N/2^{N-1}$ servers, with each layer of switching requiring $k^{N-1}/2^{N-2}$ switches (though layer N itself requires half this amount). Therefore, the choice of the number of layers in the network is determined by the number of hosts and the radix k of each switch. Given a particular data center, it is straightforward to determine the number of layers needed to interconnect each of the servers.

There are two trends that impact the cost of the network by increasing the number of necessary layers of switching: fault tolerance and high link rates. We consider each in turn:

Fault tolerance: While a FatTree network can survive link failures by relying on its multi-path topology, doing so incurs a network-wide reconvergence. This can be highly disruptive at large scale, and so redundant links are often used to survive such failures. Dual link redundancy, for instance, effectively cuts the radix of the switch in half since each logical link now requires two switch ports.

High link rates: For mature link technologies like 10 Gb/s Ethernet, high-radix switches are widely available commercially: 10 Gb/s switches with 64 or even 96 ports are becoming commodity. In contrast, newer generations of switches based on 40 Gb/s have much lower radices, for example 16 to 24 ports per switch. Hence, as data center operators build out new networks based on increasingly faster link rates, it will not always be possible to use high radix switches as the fundamental building block. This constraint will necessitate additional switching layers and, thus, additional cost and complexity. Table 1 shows the number of core network ports (ports used to connect one layer of switching to an adjacent layer) for a set of data center sizes and switch radices. Note that since this table shows fully-provisioned topologies, it serves as an upper bound to what might be built in practice since the network might be only partially provisioned depending on the number of nodes that need to be supported.

In summary, at large scale several switching layers are likely necessary to deliver scalable bandwidth to a large number of servers. Each layer of switching in the data center network adds additional cost, wiring, and complexity. This cost is driven primarily from three sources: the switches, optical transceivers, and fiber links. Since each layer in a fully provisioned FatTree network consists of $k^{N-1}/2^{N-2}$ switches, these switches constitute a considerable source of cost and motivate increased adoption of OCS switching.

2.2 OCS Model

We now describe a simple model of an OCS suitable for supporting a greater share of overall network traffic than previous proposals. This model is similar to that assumed by previous hybrid network designs [6, 8, 24], with a key difference: orders of magnitude faster switching speed.

We consider a model consisting of an N-port optical circuit switch, with a reconfiguration latency of $O(10)$ μs. Each input port can be mapped to any output port, and these mappings can be changed arbitrarily (with the constraint that only one input port can map to any given output port). The OCS does not buffer packets, and indeed does not interpret the bits in packets either — the mapping of input ports to output ports is entirely controlled by an external scheduler. This scheduler is responsible for determining the time-varying mapping of input ports to output ports and programming the switch accordingly.

For this reason, we assume that TORs attached to the OCS support per-destination flow control, meaning that packets for destination D are only admitted to a switch input port when that input port connects to D. Packets to destinations other than D are queued in the edge TOR during this time. Furthermore, during the OCS reconfiguration period, all packets are queued in the TOR. Since the OCS cannot buffer packets, the TOR must be synchronized to only transmit packets at the appropriate times. This queueing can lead to significant delay, especially for small flows that are particularly sensitive to the observed round-trip time. In these cases, packets can be sent to a packet switch in the spirit of other hybrid network proposals. In this work, we focus on the OCS and its control plane in isolation, focusing particularly on reducing the end-to-end reconfiguration latency. In this way, our work is complementary to other work in designing hybrid networks.

3. MICROSECOND SCHEDULING

A key challenge in supporting microsecond-latency OCS switches is effectively making use of short-lived circuits. In this section, we describe our approach to circuit scheduling, called Traffic Matrix Scheduling (TMS). For now, we assume that the network-wide traffic demand is known and return to the issue of estimating demand at the end of the section.

3.1 Overview

Existing approaches that integrate OCS hardware into the data center amortize the long switching time (tens of milliseconds) of previous generation optical technology by reconfiguring the OCS only once every few 100s of milliseconds or even several seconds. The substantial interval between reconfigurations affords their underlying control loops the opportunity to estimate demand, calculate an optimal OCS configuration, and communicate it across the network every time the switch is repositioned.

Previous hybrid networks perform what we call hotspot scheduling (HSS). HSS (a) measures the inter-rack traffic matrix, (b) estimates the traffic demand matrix, (c) identifies hotspots, and (d) uses a centralized scheduler to establish physical circuits between racks to offload only the identified hotspot traffic onto the circuit-switched network. The remaining traffic is routed over the packet-switched network. Because of the substantial delay between reconfigurations, HSS can employ complex methods and algorithms to estimate demand and identify hotspots. Errors in identifying hotspots, however, can lead to significant losses in efficiency. If a selected hotspot does not generate sufficient traffic to saturate a circuit, then the remaining capacity goes to waste for the (non-trivial) duration of the current configuration.

When the OCS can be reconfigured on the order of 10s of μs, however, it is possible to route most or even all traffic demand over circuits. In contrast to HSS, we propose an approach called "Traffic Matrix Switching" (TMS) that estimates demand and calculates a short-term *schedule* that moves the OCS rapidly through a sequence of configurations to service predicted demand. By rapidly time-sharing circuits across many destinations at microsecond time scales, TMS is able to make more effective use of the circuit bandwidth (and reducing to hotspot scheduling when demand is extremely concentrated). The key insight is that by sending the upcoming schedule to both the OCS and the TORs, they can effectively make use of each circuit as it becomes available. Moreover, while the current schedule is being carried out, the control loop can enter its next iteration. In this way, the running time of the control plane is decoupled from the switch speed. In particular, the control plane only needs to recompute schedules fast enough to keep up with shifts in the underlying traffic patterns, rather than the circuit switch operation.

3.2 Example

For instance, consider the situation of eight racks running Hadoop and generating a perfectly uniform all-to-all communication pattern. Figure 2(a) shows the racks physically connected to the same core-circuit switch; Figure 2(b) shows the logical connectivity, and Figure 2(c) shows the inter-rack traffic demand matrix with sources as rows, destinations as columns, and values as fractions of the total link rate. The diagonal is not zero because hosts send to other hosts in the same rack. Although this intra-rack traffic does not transit the core circuit switch, it is still accounted for in the traffic demand matrix. This matrix is the desired transmission rate of the hosts, and it is the responsibility of the network to satisfy this demand.

The Gantt chart in Figure 2(d) shows a circuit switch schedule that partitions time into eight equal-duration

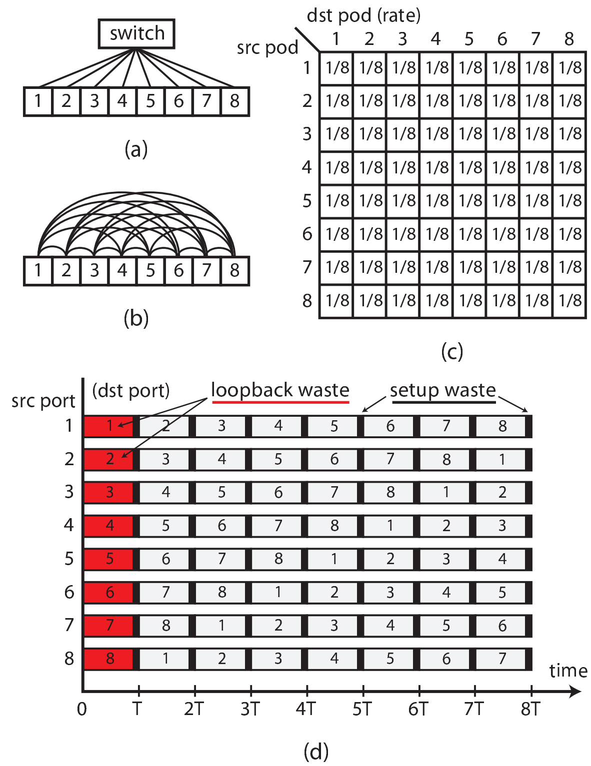


Figure 2: Eight racks running Hadoop with an all-to-all communication pattern: (a) physical topology, (b) logical topology, (c) inter-rack traffic demand matrix, (d) circuit switch schedule. (Figure reproduced with permission from [7].)

time slots. Over the course of the schedule, each source port will connect to each destination port for exactly 1/8 of the total time. It thus implements the logical full mesh topology in Figure 2(b) and allows all of the traffic to be routed. The schedule then repeats. A circuit switch schedule has two sources of waste. First, loopback traffic does not leave the rack and transit the circuit switch, so any circuit switch loopback assignments are wasted, such as the assignment from $t = 0$ to $t = T$. Second, the circuit switch takes a non-negligible amount of time to switch and setup new circuits (t_{setup}), which we represent as black bars at the end of each time slot. No traffic can transit the circuit switch during this time. Reducing loopback waste requires careful scheduling, whereas reducing setup waste requires faster switching. Finally, note that although this example produces a repeating schedule, TMS can generate arbitrary time-varying circuit assignments as we describe below.

3.3 Schedule computation

The TMS algorithm is divided into two phases as shown in Figure 3. In phase 1, the traffic demand matrix (TDM) is scaled into a bandwidth allocation matrix (BAM). A TDM represents the amount of traffic,

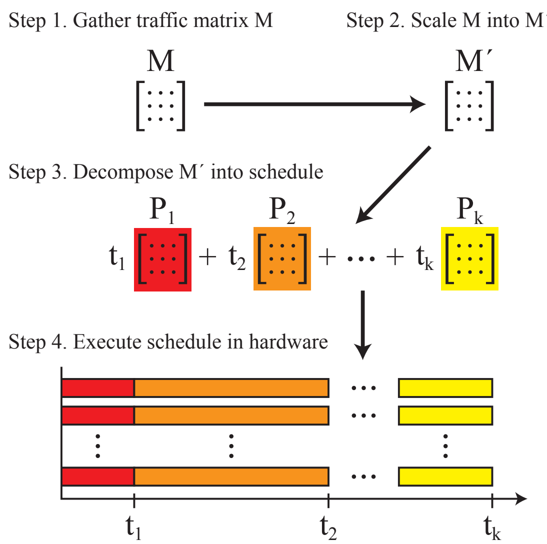


Figure 3: Steps of the traffic matrix scheduling algorithm. (Figure reproduced with permission from [7].)

in units of circuit line rate, that the hosts in a source rack wish to transmit to the hosts in a destination rack. A BAM, on the other hand, represents the fraction of circuit bandwidth the switch should allocate between each input-output port pair in an ideal schedule. In general, the TDM may not be admissable (i.e., the total demand is greater than the network capacity). In practice, though, the network is rarely driven to full utilization, so we need to scale "up" the TDM to arrive at a BAM. If no rack wishes to send more than its link rate (its row sum is less than or equal to 1) and no rack wishes to receive more than its link rate (its column sum is less than or equal to 1), then we say that the TDM is both admissible and doubly substochastic. The goal of scaling the TDM is to compute a doubly stochastic BDM where its row sums and column sums are all exactly equal to 1 — meaning the circuits would be fully utilized. By scaling the TDM into a BAM, we simultaneously preserve the relative demands of the senders and receivers while satisfying the constraints of the circuit switch. Several matrix scaling algorithms can be used for this purpose. Sinkhorn's algorithm [17] is particularly attractive because it works even when the originally TDM is not admissible (i.e., the network is over driven).

In phase 2, the BAM is decomposed into a circuit switch schedule, which is a convex combination of permutation matrices that sum to the original BAM,

$$\text{BAM} = \sum_{i}^{k} c_i P_i \tag{1}$$

where $0 \leq i \leq k$, and $k = N^2 - 2N + 2$. Each permutation matrix, P_i, represents a circuit switch assignment, and each scalar coefficient, c_i, represents a time slot duration as a fraction of the total schedule duration. A variety of matrix decomposition algorithms exist. We employ an algorithm originally due to Birkhoff-von Neumann (BvN) [5, 23] that can decompose any doubly stochastic matrix, implying we can always compute a perfect schedule given a BAM. Improved versions of the classic BvN algorithm have running times between $O(n \log^2 n)$ and $O(n^2)$ [10].

3.4 Longest time-slot first scheduling

While BvN always produces a schedule that eventually serves all input-output pairs according to their demand, sometimes it may be better not to schedule all traffic over the circuit switch and to simply schedule only the longest time slots. The reason is that the BvN decomposition algorithm generates time slots of different lengths, some of which can be quite short (e.g., less than 1% of the entire schedule). With such a short time slot, it is likely that the OCS switching time (t_{setup}) would dominate any bandwidth delivered during those small slots. In these cases, it is better to route that traffic over the packet-switched network.

The greatest benefit comes from scheduling the first n time slots, where n is chosen based on both the minimum required duty cycle, D, as well as the maximum allowed schedule length, T_{schedule}. We extend our definition of D to variable-length time slots as follows

$$T_{\text{setup}} = n t_{\text{setup}} \tag{2}$$

$$D = \frac{T_{\text{stable}}}{T_{\text{setup}} + T_{\text{stable}}} = \frac{T_{\text{stable}}}{T_{\text{schedule}}} \tag{3}$$

where $n \leq k$ is the number of time slots in the schedule, T_{schedule} is the duration of the entire schedule period, and t_{stable} is the average time that the circuit is established. This definition allows us to choose to schedule only the first n time slots. Traffic that is not scheduled over the circuit-switched network instead transits the packet-switched network. Using the same randomly generated TDM used in [7], Table 3.4 shows the tradeoffs in choosing the right number of time slots for the schedule.

As n increases, an increasing fraction of the total traffic routes over the circuit-switched network. In the limit when $n = k$, all traffic routes over the circuit-switched network. However, the duty cycle decreases with increasing n. Because T_{schedule} is held constant, T_{stable} must decrease as T_{setup} increases. For example, if the minimum required duty cycle was 95%, then by setting $n = 5$, 80.6% of the total traffic would be routed over circuit switches. Alternatively, at the cost of increased host buffering, we could increase T_{schedule} to increase n to 6 or 7 while keeping the duty cycle at 95%.

n	**Circuit**	**Packet**	**D**
0	0%	100.0%	N/A
1	39.4%	60.6%	100.0%
2	53.8%	46.2%	98.0%
3	63.8%	36.2%	97.0%
4	72.7%	27.3%	96.0%
5	80.6%	19.4%	95.0%
6	87.3%	12.7%	94.0%
7	92.3%	7.7%	93.0%
8	96.6%	3.4%	92.0%
9	99.3%	0.7%	91.0%
10	100.0%	0%	90.0%

Table 2: Example of tradeoffs between the number of schedule time slots (n), the amount of traffic sent over the optical-circuit switched network (Circuit) vs. the packet-switched network (Packet), and the duty cycle (D) for a randomly generated TDM. $t_{setup} = 10\ \mu$s, $T_{schedule} = 1$ ms. (Used with permission from [7].)

3.5 Demand estimation

Traffic matrix scheduling, just like hotspot scheduling, requires an estimate of the network-wide demand. There are several potential sources of this information. First, packet counters in the TORs can be polled to determine the traffic matrix, and from that the demand matrix can be computed using techniques presented in Hedera [2]. This method would likely introduce significant delays, given the latency of polling and running the demand estimator. A second potential approach, if the network is centrally controlled, is to rely on OpenFlow [13] network controllers to provide a snapshot of the overall traffic demand. Third, an approach similar to that taken by c-Through [24] may be adopted: A central controller, or even each TOR, can query individual end hosts and retrieve the TCP send buffer sizes of active connections. Asynchronously sending this information to the TORs can further reduce the latency of collecting the measurements. Finally, application controllers, such as the Hadoop JobTracker [15], can provide hints as to future demands. Our prototype implementation does not implement demand estimation.

4. ANALYSIS

The throughput of a network that uses circuit switching is constrained by the network's *duty cycle*, and its feasibility is constrained by the amount of buffering required. We consider these issues in turn.

4.1 Duty Cycle and Effective Link Rate

In a circuit-switched network, there is a finite reconfiguration time or setup time t_{setup} during which no data can be sent. If the link data rate is R_{link}, then the effective data rate R of each circuit is

$$R = DR_{\text{link}}, \quad \text{where } D = \frac{t_{\text{stable}}}{t_{\text{setup}} + t_{\text{stable}}} \tag{4}$$

is the duty cycle and t_{stable} is the time that the circuit is "open" and can carry traffic. For example, if R_{link} is 10 Gb/s and D is 90%, which is representative of the Mordia OCS, then the effective link rate $R_{\text{effective}}$ is 9 Gb/s.

The duty cycle can be increased by reducing the setup time or increasing the duration that a circuit is open. Reducing setup time t_{setup} depends on switch technology. The duration t_{stable} that a circuit is open is controllable. However, having circuits that are open for a long period of time affects the amount of buffering that is required at the host, as we discuss below.

Interestingly, since the Mordia OCS is rate agnostic, it is possible to increase overall delivered bandwidth by using faster optical transceivers to increase the link rate while simultaneously reducing the duty cycle for the circuit-switched portion of the network.

time slot →

VOQ ↓																	
1	0	0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7
2	0	1	0	1	2	3	4	5	6	7	0	1	2	3	4	5	6
3	0	1	2	0	1	2	3	4	5	6	7	0	1	2	3	4	5
4	0	1	2	3	0	1	2	3	4	5	6	7	0	1	2	3	4
5	0	1	2	3	4	0	1	2	3	4	5	6	7	0	1	2	3
6	0	1	2	3	4	5	0	1	2	3	4	5	6	7	0	1	2
7	0	1	2	3	4	5	6	0	1	2	3	4	5	6	7	0	1
8	0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7	0
total	0	7	13	18	22	25	27	28	28	28	28	28	28	28	28	28	28

Figure 4: Virtual output queue (VOQ) buffer occupancies for a TOR from cold start. (Figure reproduced with permission from [7].)

4.2 Buffer Requirements

Buffering is required at the source in a circuit-switched network because there is not always a circuit established between a particular source and destination. In this section, we analyze these buffering requirements.

Assume that each TOR connected to the Mordia switch maintains a set of N virtual output queues (VOQs) [20], one for every possible circuit destination. Strict VOQ is not required, but the TOR must maintain at least one set of queues for each possible destination. When a circuit is established, all traffic destined for that particular destination is drained from the respective queue. Figure 4 shows the buffer occupancies of these VOQs of a TOR from a cold start, in units of the slot time T (without loss of generality, we assume uniform slot times here). In less than one complete scheduling period a TOR has filled its VOQs to the steady-state level. A particular queue fills at a rate dictated by the traffic matrix until a circuit is established to the appropriate destination. The queue is drained over the duration that the circuit is open.

For an all-to-all workload with N TORs, an effective link rate of R bits per second, and a slot duration of T seconds, the buffering required by each host is:

$$B \;=\; R(N-1)T \qquad \text{(bits)} \tag{5}$$

Examining (5) shows why millisecond switching times are simply too large to support traffic matrix scheduling. For example if $R = 9$ Gb/s, $N = 64$, and $T = 100$ ms, then B is 7.1 GB of buffering per OCS port, which is not currently practical. Given that R and N are both likely to increase in future data centers, the only way to make traffic matrix scheduling practical is to decrease the slot time T by using, e.g., microsecond switching. Setting $T = 100$ μs yields $B = 7.1$ MB of buffering per port, which is currently practical. This is a key reason why traffic matrix scheduling requires microsecond circuit switching.

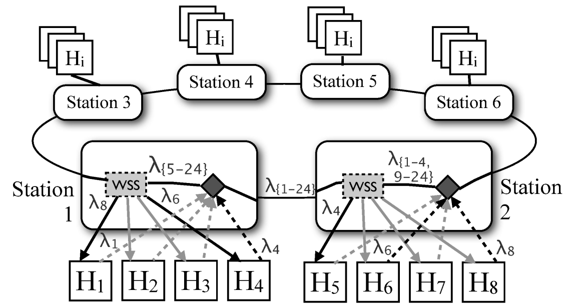


Figure 5: The Mordia OCS prototype, which consists of a ring conveying N wavelengths through six stations. Each source TOR transmits on its own wavelength, and each station forwards a subset of four wavelengths to the TORs attached to it. This prototype supports an arbitrary reconfigurable mapping of source ports to destination ports with a switch time of 11.5 μs.

5. IMPLEMENTATION

To evaluate our design, we chose to implement it in a testbed environment. This implementation effort consists of two primary tasks: (1) selecting an OCS capable of switching at O(10) μs, and (2) modifying TORs to support flow control on a per-destination basis at microsecond timescales. Unfortunately, as we discuss in Section 8, the only practical commercially available OCSes that can switch in sub-100 μs timescales have small port counts (e.g., 4 or 8 ports). To evaluate at the scale of a TOR (e.g., 24–48 hosts), we instead build our own prototype OCS supporting 24 ports based on commercial wavelength-selective switches, described in Section 5.1. Instead of building our own TORs with our flow control requirements, we instead emulate them using commodity Linux servers, as described in Section 5.2.

5.1 Mordia Prototype

The Mordia prototype is a 24-port OCS that supports arbitrary reconfiguration of the input-to-output port mappings. We first describe the underlying technology we leveraged in building the OCS, then describe its design.

5.1.1 Technology

Unlike previous data center OCS designs [8, 24], we chose not to use 3D-MEMS based switches due to their high switch time. The maximum achievable speed of a 3D-MEMS space switch depends on the number of ports. Large port count switches require precise analog control of the 2-axis orientation of relatively large mirrors. Since the mirror response time depends on the size and angular range, there is in general a design tradeoff between the switch port count, insertion loss, and switching speed. Commercial 3D-MEMS switches support reconfiguration times in the 10s of milliseconds range [18].

Another type of optical circuit switch is a *wavelength-selective switch* (WSS). It takes as input a fiber with N wavelengths in it, and it can be configured to carry any subset of those N wavelengths to M output ports. Typically a WSS switch has an extra "bypass" port that carries the remaining $N - M$ frequencies. We call this type of WSS switch a 1xM switch, and in our prototype, $M = 4$. Our switch does not have a bypass port, and so we implement the bypass functionality external to the WSS using additional optical components.

The internal switching elements used in a wavelength-selective switch can be constructed using liquid crystal technology or MEMS [9]. Most MEMS WSSes use analog tilt to address multiple outputs, but at least one commercial WSS has been built using binary MEMS-based switches [19]. Binary MEMS switching technology uses only two positions for each mirror moving between two mechanically stopped angles and also uses much smaller mirrors with respect to a 3D-MEMS space switch. A similar binary MEMS switch is used for commercial projection televisions. The binary switching of small mirror elements results in an achievable switching speed that is several orders of magnitude faster than a commercial 3D-MEMS switch.

In general, there is a tradeoff between 3D-MEMS, which offers high port count at relatively slow reconfiguration time, and 2D-MEMS, which offers microsecond switching time at small port counts (e.g., 1×4 or 1×8). The key idea in the Mordia OCS prototype is to harness six 1×4 switches with bypass ports to build a single $24x24$-port switch. We now briefly summarize the operation of the data path.

5.1.2 Data Plane

The Mordia OCS prototype is physically constructed as a unidirectional ring of $N = 24$ individual wavelengths carried in a single optical fiber. Each wavelength is an individual channel connecting an input port to an output port, and each input port is assigned its own specific wavelength that is not used by any other input port. An output port can tune to receive any of the wavelengths in the ring, and deliver packets from any of the input ports. Consequently, this architecture supports circuit unicast, circuit multicast, circuit broadcast, and also circuit loopback, in which traffic from each port transits the entire ring before returning back to the source. We note that although the data plane is physically a ring, any host can send to any other host, and the input-to-output mapping can be configured arbitrarily (an example of which is shown in Figure 5).

Wavelengths are dropped and added from/to the ring at six *stations*. A station is an interconnection point for TORs to receive and transmit packets from/to the Mordia prototype. To receive packets, the input containing all N wavelengths enters the WSS to be wavelength multiplexed. The WSS selects four of these wavelengths, and routes one of each to the four WSS output ports, and onto the four TORs at that station. To transmit packets, each station adds four wavelengths to the ring, identical to the four wavelengths the station initially drops. To enable this scheme, each station has a commercial 1×4-port WSS.

5.1.3 TORs

Each TOR connects to the OCS via one or more optical uplinks. Each TOR internally maintains $N - 1$ queues of outgoing packets, one for each of the $N - 1$ OCS output ports. The TOR participates in a control plane, which is used to inform each TOR of the short-term schedule of impending circuit configurations. In this way, the TORs know which circuits will be established in the near future, and can use that foreknowledge to make efficient use of circuits once they are established.

Initially, the TOR does not send any packets into the network, and simply waits to become synchronized with the Mordia OCS. This synchronization is necessary since the OCS cannot buffer any packets, and so the TOR must drain packets from the appropriate queue in sync with the OCS's circuit establishment. Synchronization consists of two steps: (1) receiving a schedule from the scheduler via an out-of-band channel (e.g., an Ethernet-based management port on the TOR), and (2) determining the current state of the OCS. Step 2 can be accomplished by having the TOR monitor the link up and down events and matching their timings with the schedule received in Step 1. Given the duration of circuit reconfiguration is always 11.5 μs, the scheduler can artificially extend one reconfiguration delay periodically to serve as a synchronization point. The delay must exceed the error of its measurement and any variation in reconfiguration times to be detectable (i.e., must be greater than 1 μs in our case). Adding this extra delay incurs negligible overhead since it is done infrequently (e.g., every second).

We use the terminology *day* to refer to a period when a circuit is established and packets can transit a circuit, and we say that *night* is when the switch is being reconfigured, and no light (and hence no packets) are transiting the circuit. The length of a single schedule is called a *week*, and the week lengths can vary from week-to-week. When the OCS is undergoing reconfiguration, each TOR port detects a link down event, and night begins. Once the reconfiguration is complete, the link comes back up and the next "day" begins.

During normal-time operation, any data received by the TOR from its connected hosts is simply buffered internally into the appropriate queue based on the destination. The mapping of the packet destination and the queue number is topology-specific, and is configured out-of-band via the control plane at initialization time and whenever the topology changes. When the TOR detects that day i has started, it begins draining packets from queue i into the OCS. When it detects night time (link down), it re-buffers the packet it was transmitting (since that packet likely was 'runted' mid-transmission), and stops sending any packets into the network.

5.1.4 Data plane example

Figure 5 shows an overview of the prototype's data path. In this example, there are a three circuits established: one from H_6 to H_4, one from H_8 to H_1, and one from H_4 to H_5. Consider the circuit from H_4 to H_5. H_4 has a transceiver with its own frequency, shown in the Figure as λ_4. This signal is introduced into the ring by an optical mux, shown as a black diamond, and transits to the next station, along with the other $N - 1$ frequencies. The WSS switch in Station 2 is configured to forward λ_4 to its first output port, which corresponds to H_5. In this way, the signal from H_4 terminates at H_5. The $N - 4$ signals that the WSS is not configured to map to local hosts bypass the WSS, which is shown as $\lambda_{\{1-4,9-24\}}$. These are re-integrated with the signals from hosts H_5 through H_8 originating in Station 2, and sent back into the ring. A lower-bound on the end-to-end reconfiguration time of such a network is gated on the switching speed of the individual WSS switches, which we evaluate in Section 6.1.

5.1.5 Implementation details

The implementation of the hardware for the Mordia prototype consists of four rack-mounted sliding trays.

Three of these trays contain the components for the six stations with each tray housing the components for two stations. The fourth tray contains power supplies and an FPGA control board that implements the scheduler. This board is based on a Xilinx Spartan-6 XC6SLX45 FPGA device. Each tray contains two wavelength-selective switches, which are 1×4 Nistica Full Fledge 100 switches. Although these switches can be programmed arbitrarily, the signaling path to do so has not yet been optimized for low latency. Thus we asked the vendor to modify the WSS switches to enable low-latency operation by supporting a single signaling pin to step the switch forward through a programmable schedule. As a result, although our prototype only supports weighted round-robin schedules, those schedules can be reprogrammed on a week-to-week basis. This limitation is not fundamental, but rather one of engineering expediency.

5.2 Emulating TORs with Commodity Servers

To construct our prototype, we use commodity servers to emulate each of the TORs. Although the Mordia OCS supports 24 ports, our transceiver vendor was not able to meet specifications on one of those transceivers, leaving us with 23 usable ports in total. Each of our 23 servers is an HP DL 380G6 with two Intel E5520 4-core CPUs, 24 GB of memory, and a dual-port Myricom 10G-PCIE-8B 10 Gb/s NIC. One port on each server contains a DWDM 10 Gbps transceiver, taken from the following ITU-T DWDM laser channels: 15-18, 23-26, 31-34, 39-42, 47-50, and 55-58. Each server runs Linux 2.6.32.

5.2.1 *Explicit synchronization and control*

Each of the emulated TOR must transmit packets from the appropriate queue in sync with the OCS with microsecond precision. The source code to our NIC firmware is not publicly available, and so we cannot detect link up and down events in real time and cannot implement the synchronization approach presented in Section 5.1.3. Instead, we have modified our prototype to include a separate synchronization channel between the scheduler and the servers that the scheduler uses to notify the servers when the switch is being reconfigured. Ethernet NICs do not typically provide much direct control over the scheduling of packet transmissions. Thus we have implemented a Linux kernel module to carry out these tasks. We now describe how we modified the Linux networking stack on each server to support circuit scheduling, and to remain synchronized with the OCS.

We modify the OS in three key ways. First, we adapt the Ethernet NIC driver to listen for synchronization packets from the scheduler so that the host knows the current state of the OCS. Second, we modify the NIC driver to ignore the “link-down” events that occur when

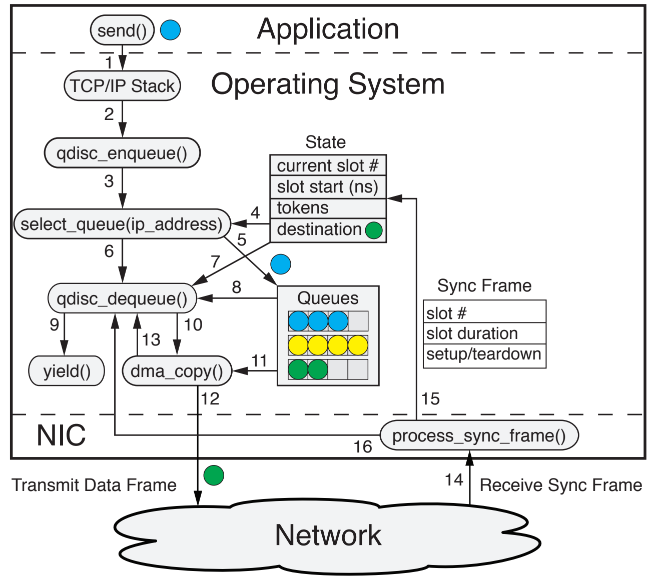


Figure 6: A software implementation of multi-queue support in Linux using commodity Ethernet NICs. Sync frames coordinate state between each emulated TOR (server) and the scheduler, so that each Qdisc knows when to transmit Ethernet frames.

the OCS is reconfiguring. Third, we add a custom queuing discipline (Qdisc) that drains packets from queues based on the configuration of the OCS.

Synchronization packets: The OCS FPGA controller transmits synchronization packets to a separate 10G packet-switched network which connects to a second Ethernet port on each server. These packets are sent before and after reconfiguration so that all connected devices know the state of the OCS. The packets include the slot number, the slot duration, and whether the circuits are being setup or torn down. A map is maintained between the slot number and each destination. The Ethernet NIC driver also maintains a data structure with a set of per-circuit tokens to control the data transmission time and rate.

Link-down events: Since the OCS is switching rapidly, the host NIC ports attached to the OCS experience numerous link-up and link-down events. When Linux receives a link-down event, it normally disables the interface and resets and closes any open sockets and TCP connections. To prevent these resets, we disable the link-down and link-up calls in the NIC driver. Our NIC vendor believes that with access to the firmware source code, synchronization could be implemented entirely in hardware, obviating the need for the out-of-band synchronization channel.

Mordia Qdisc (Figure 6): When a user’s application sends data, that data transits the TCP/IP stack (1) and is encapsulated into a sequence of Ethernet frames.

The kernel enqueues these frames into our custom Qdisc (2), which then selects (3) one of multiple virtual output queues (VOQs) based on the packet's IP address and the queue-to-destination map (4). The Ethernet frame is enqueued (5) and the qdisc_dequeue function is scheduled (6) using a softirq. The qdisc_dequeue function reads the current communication slot number (7) and checks the queue length (8). If there is no frame to transmit, control is yielded back to the OS (9). If there is a frame to transmit, the frame is DMA copied to the Ethernet NIC (10–12). The total number of packets sent directly corresponds to the number of tokens accumulated in the Ethernet NIC's data structure to control the timing and the rate. The qdisc_dequeue function is then scheduled again (13) until VOQ is empty and control is yielded back to the OS (9). When the next sync frame arrives (14), it is processed, and the scheduling state is updated (15). Then the qdisc_dequeue function is scheduled with a softirq in case there are frames enqueued that can now be transmitted (16). Given that all the packets are only transmitted during the times that the slot is active, the code for receiving packets did not need to be modified.

6. EVALUATION

Our evaluation seeks to answer the following research questions:

1. What is the baseline end-to-end reconfiguration time of the Mordia OCS as seen by TORs?
2. How closely can the control plane keep TOR devices synchronized with the OCS?
3. What is the overall throughput and circuit utilization delivered by Mordia?
4. How well does TCP perform over the OCS portion of the hybrid network?

We evaluate each of these questions below.

6.1 End-to-end reconfiguration time

We know from other work [8] that the raw switching speed of the underlying OCS does not determine the end-to-end switching speed, since additional time is required for reinitializing the optics and software overheads. In this section, we empirically measure the OCS switching speed as perceived at a packet level by the devices connected to it. This fundamental switching speed gates the expected performance we expect to see in the remainder of our evaluation.

We first connect 23 emulated TORs (which we refer to as hosts) to the OCS prototype (with host i connected to port i). Host 1 transmits fixed-size Ethernet frames at line rate, ignoring synchronization packets. Host 1 transmits continuously, even during gaps. Hosts 2 through 23 capture the first 22 octets of each frame using tcpdump. Each frame contains an incrementing sequence number so we can detect loss. After each experiment, we merge the pcap files from each host.

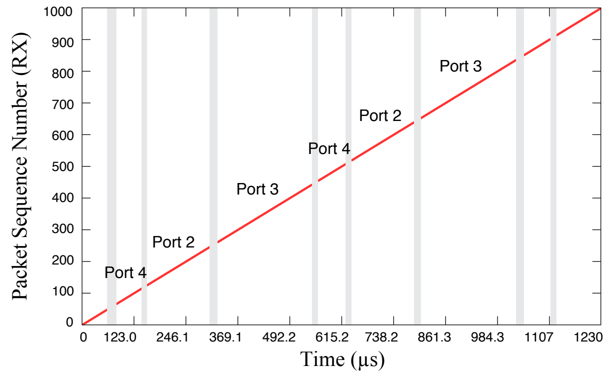


Figure 7: Variable-length slots.

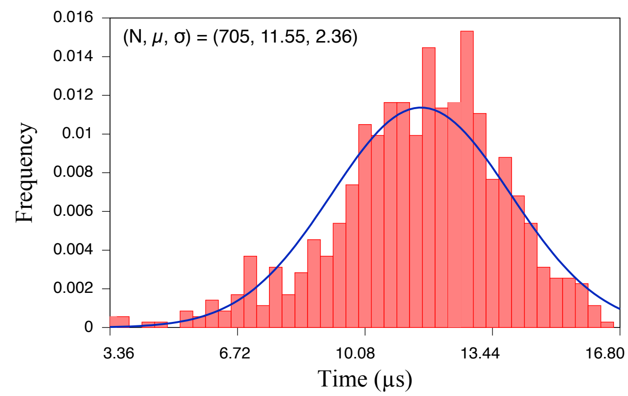


Figure 8: Histogram of t_{setup} using 705 samples. A normal curve is fitted to the data.

Figure 7 shows an experiment with regular-sized Ethernet frames (1500 bytes) and variable-length slots (80 μs, 160 μs, and 240 μs). The x-axis is time and the y-axis is packet sequence number. The OCS was programmed with a round-robin schedule, meaning that during slot k, input port i was connected to output port $i + k \mod 23$. The slot durations vary in size, which allocates bandwidth proportionally to different hosts based on the slot length. Gaps are highlighted as gray vertical strips, and frames transmitted during gaps are lost. The remaining frames are received by the other hosts. The last packet transmitted during a slot often gets dropped by the receiving NIC because it is cut off in the middle by the OCS.

From a merged pcap trace of approximately one million packets, we extracted 705 gaps in packet transmission. The length of each gap is a measurement of t_{setup}. Figure 8 shows the resulting histogram. The data fits a normal distribution with a mean of 11.55 μs and a standard deviation of 2.36 μs. Note that this packet capture

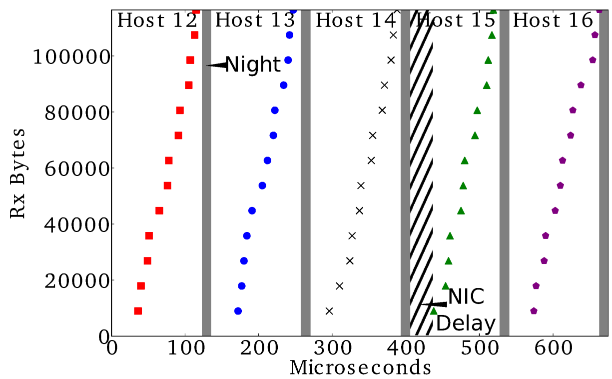


Figure 9: Host 1's Qdisc receiving UDP packets from Hosts 12–16 as it cycles through circuits connecting it to 22 other hosts.

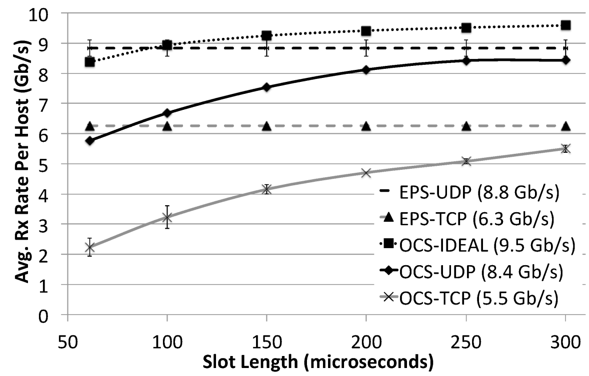


Figure 10: Throughput delivered over the OCS. The fundamental difference between ideal and observed is due to the OCS duty cycle. Further deviations are due to our system artifacts, namely the lack of segmentation offloading in the NIC, NIC-induced delay, and synchronization packet jitter. The minimum slot length is 61 μs. The legend shows the maximum average receive rate for each switch and protocol combination.

is collected across several machines, and so some of the variance shown in Figure 8 is due to clock skew across nodes.

With $T = 106$ μs and $t_{\text{setup}} = 11.5$ μs, the duty cycle is equal to 89.15%. Therefore we expect to have captured 89.15% of the transmitted Ethernet frames. From 997,917 transmitted packets, we captured 871,731 packets, yielding a measured duty cycle of 87.35%. In other words, there are approximately 18,000 additional missing packets. We attribute these additional missing packets primarily to periods where the sender or receiver was interrupted by the non-real-time Linux kernel.

Summary: These experiments show that the end-to-end reconfiguration latency of the Mordia OCS, including the time to re-establish the link at the optical component level, is on average 11.5 μs. Further, the FPGA-based scheduler is able establish variable-length days and control the OCS with high precision.

6.2 Emulated TOR Software

The previous section demonstrates that a single host can utilize 87.35% of a circuit with an 89.15% duty cycle. However, this measurement does not account for host synchronization at the OCS ports. Figure 9 captures Host 1 receiving 8,966 octet UDP packets from Hosts 12–16 via the Qdisc described in Section 5.2.1 for a day and night of 123.5 μs and 11.5 μs, respectively. The transmission time and sequence number of each packet is determined from a tcpdump trace that runs on only Host 1.

First, we note that it takes on average 33.2 μs for the first packet of each circuit to reach Host 1. The tcpdump trace indicates that it takes less than 3 μs to process the synchronization frame and transmit the first packet. When sending a 9000-octet frame (7.2 μs), the packet spends at least 23 μs in the NIC before being transmitted. To prevent this "NIC delay" from causing packet loss in excess of 0.5%, the OS Qdisc must stop queuing packets for transmission 23 μs early. The result is that the Qdisc cannot use 23 μs of the slot due to the behavior of the underlying NIC hardware. Vattikonda et al. [22] have shown how the use of hardware NIC priority flow control (PFC) pause frames can be used to enable fine-grained scheduling of circuits on (all-electrical) data center networks, and this technique could be applied to Mordia.

Summary: The Linux hosts used as emulated TORs are able to drain packets into the network during the appropriate "day," which can be as small as 61 μs. However, jitter in receiving synchronization packets results in a 0.5% overall loss rate, and there is a 23 μs delay after each switch reconfiguration before the NIC begins sending packets to the OCS. These overheads are specific to our use of commodity hosts as TORs.

6.3 Throughput

We generated all-to-all TCP and UDP traffic between 23 hosts and measured the throughput both over a traditional electrical packet switch (EPS, used as a baseline) as well as our Mordia OCS prototype including emulated TOR switches (shown in Figure 10). The throughput over the EPS serves as an upper bound on the potential performance over the OCS. In addition, throughput over the OCS is fundamentally limited by the OCS duty cycle.

Starting from baseline, the EPS can deliver UDP traffic (EPS-UDP) at an average receive rate of 8.83 Gb/s to each of the 23 hosts. We find that as the number of hosts increases from 2 to 23, the average receive rate degrades from 9.81 Gb/s, which we attribute to the kernel and NIC. The EPS is capable of delivering TCP traffic at 8.69 Gb/s, which is within 1.6% of UDP traffic. However, this throughout relies on TCP segmentation offloading (TSO) support in the NIC, which is incompatible with our Mordia kernel module. The happens because, when circuits are reconfigured, any packets in flight are 'runted' by the link going down, and we lose control over the transmission of packets when relying on TSO. Consequently, Mordia requires disabling TSO support. On an all-electrical packet network, TCP without TSO support is limited to 6.26 Gb/s (EPS-TCP), which we use as an upper bound on the performance we expect to see over the OCS.

Figure 10 shows the raw bandwidth available to each host (calculated as the duty cycle) from the OCS as OCS-IDEAL. It is important to remember that this line does not account for the 23.5 μs *NIC delay* which acts to reduce measured duty cycle even more. For the experiments, we varied the OCS slot duration between 61–300 μs to observe the effect of different duty cycles (due to the programming time of our WSSs, the smallest slot duration we support is 61 μs). The OCS's UDP throughput (OCS-UDP) ranges from 5.726-8.429 Gb/s, or within 4.6% of EPS-UDP. The major reasons for the discrepancy are duty cycle, NIC delay, the OS's delay in handling a softirq, and synchronization jitter (see discussion below).

TCP throughput on the OCS (OCS-TCP) ranges from 2.231–5.501 Gb/s, or within 12.1% of EPS-TCP for large slot durations. TCP throughput suffers from all of the issues of UDP throughput, but suffers from two addition issues. First, TCP traffic cannot use TSO to offload, and so the TCP/IP stack becomes CPU-bound handling the required 506 connections. Second, the observed 0.5% loss rate invokes congestion control, which decreases throughput. However, TCP does show an upward trend in bandwidth with increasing duty cycle. We attempted the kernel bypass techniques from Vattikonda et al. [22] to eliminate NIC delay and the softirq variance, but found that it was difficult to minimize loss rate below 5% due to interruptions from the OS that would cause packets to be sent to the wrong host during reconfiguration.

We mention above that synchronization jitter causes a reduction in throughput, and is also responsible for packet loss. Synchronization packets are generated in hardware to minimize latency and jitter, but the OS can add jitter in how its devices receive the packets and schedule softirqs. To measure this jitter, we set the day and night to 106 μs and 6 μs, respectively, and capture 1,922,507 synchronization packets across 3 random hosts. We compute the difference in timestamps between each packet and expect to see packets arriving with timestamp deltas of either 6±1 μs or 100±1 μs.

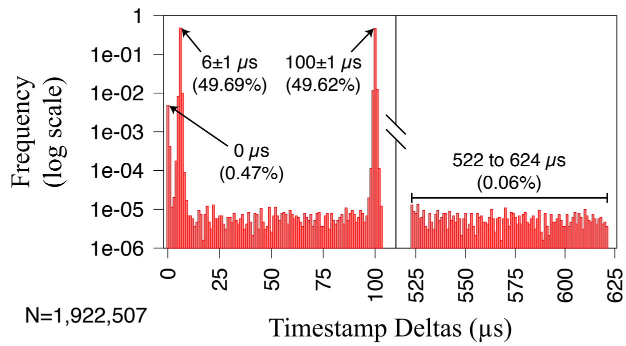


Figure 11: Synchronization jitter as seen by our software TOR switches' OS.

We found that 99.31% of synchronization packets arrive at their expected times, 0.47% of packets arrive with timestamp deltas of zero, and 0.06% packets arrive with timestamp deltas between 522 μs and 624 μs (see Figure 11). The remaining 0.16% of packets arrive between 7–99 μs. We also point out that the 0.53% of synchronization packets that arrive at unexpected times is very close to our measured loss rate. Our attempts to detect bad synchronization events in the Qdisc did not change the loss rate measurably. Firmware changes in the NIC could be used to entirely avoid the need for these synchronization packets by directly measuring the link up/down events.

Summary: Despite the non-realtime behavior inherent in emulating TORs with commodity PCs, we are able to achieve 95.4% of the bandwidth of a comparable EPS with UDP traffic, and 87.9% of an EPS, sending non-TSO TCP traffic. We are encouraged by these results, which we consider to be lower bounds of what would be possible with more precise control over the TOR.

7. SCALABILITY

Supporting large-scale data centers requires an OCS that can scale to many ports. We briefly consider these scalability implications.

WDM: The Mordia prototype we built uses a single ring with 24 wavelength channels in the C-band to create a 24×24-port OCS. Since the C-band contains 44 DWDM channels, it is straightforward to scale the prototype to 44 ports. Increasing the number of wavelengths on a single ring beyond 44 is more difficult. Mordia happens to rely on 100 GHz spacing, but we could have used 50 GHz, 25 GHz, or even 12.5 GHz spacing. Each smaller increment doubles the number of channels. SFP+ modules with lasers on the 50 GHz grid are commercially available, meaning that it is straightforward to scale to 88 ports. However, the technology to support 10G, 40G, and 100G Ethernet over narrower DWDM channels might not yet be commercially avail-

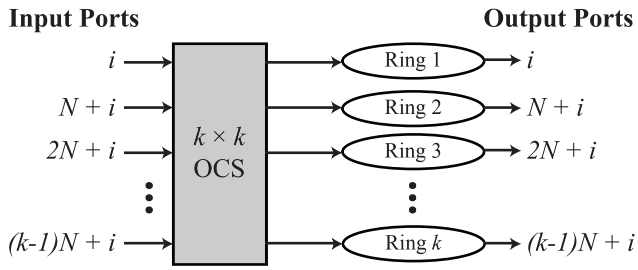


Figure 12: Multiple independent rings can be stacked to increase the total port count. Each of the k rings has N ports. Every input port $jN+i$, where $j \in \{0..k-1\}$ and $i \in \{1..N\}$, is bundled together into a $k \times k$ ring-selection OCS before being sent to its default ring. This approach allows an input normally destined for one ring to arrive at a different ring.

able or might be cost prohibitive in a data center environment. An alternative could be to keep the 100 GHz spacing but to extend into the L-band. This would allow a doubling of the number of channels, but would make amplification more difficult. Thus the use of WDM provides a small level of scalability up to a couple of hundred ports.

Discrete: Another way to scale beyond 88 ports is to use multiple stacked rings, with each ring reusing the same wavelength channels, as shown in Figure 12. For example, an 8×8 ring-selection OCS would allow the construction of a $8 \times 88 = 704$-port OCS. It is important that all inputs assigned to the same wavelength channel be connected to the same ring-selection OCS, or else there could be a collision within a particular ring. The ring-selection OCS is only used for the input ports; the output ports directly connect to the individual rings.

While the single-ring architecture is fully non-blocking, the stacked-ring architecture is blocking, meaning that not all input-output port mappings are possible. Fundamentally the challenge comes from reusing a finite number of wavelength channels across a larger number of switch ports. One possible solution to this problem is to introduce another degree of freedom by using tunable lasers that can transmit on any wavelength channel rather than on a specific channel. This should restore the fully non-blocking property of the OCS at the cost of additional optical and algorithmic complexity. Even without tunable lasers, the blocking version of the stacked-ring OCS can still potentially be useful for realistic communication patterns.

Integrated: Finally, it is possible to build an integrated scale-out OCS by interconnecting smaller OCS switches in a multi-stage topology on a single board, using waveguides instead of discrete fibers. This approach greatly reduces loss, since the couplers used to connect the switch to the fibers can be a significant source of loss. Multi-stage, integrated OCSes have been built [3], but rely on slower 3D-MEMS technology.

8. RELATED WORK

Optical switching technologies: Realistic optical switches that can be used in practice require a limited overall insertion loss and crosstalk, and must also be compatible with commercial fiber optic transceivers. Subject to these constraints, the performance of a switch is characterized by the switch speed and port count. Optical switches based on electro-optic modulation or semiconductor amplification can provide nanosecond switching speeds, but intrinsic crosstalk and insertion loss limit their port count. Analog (3D) MEMs beam steering switches can have high port counts (e.g., 1000 [4]), but are limited in switching speed on the order of milliseconds. Digital MEMs tilt mirror devices are a "middle-ground". They have a lower port count than analog MEMs switches, but have a switching speed on the order of a microsecond [9] and a sufficiently low insertion loss to permit constructing larger port-count OCSes by composition.

"Hotspot Schedulers": Mordia is complementary to work such as Helios [8], c-Through [24], Flyways [11], and OSA [6], which explored the potential of deploying optical circuit switch technology in a data center environment. Such systems to date have all been examples of hotspot schedulers. A hotspot scheduler observes network traffic over time, detects hotspots, and then changes the network topology (e.g., optically [6,8,24] or wirelessly [11]) such that more network capacity is allocated to traffic matrix hotspots and overall throughput is maximized.

Optical Burst Switching: Optical Burst Switching [16,21] is a research area exploring alternate ways of scheduling optical links through the Internet. Previous and current techniques require the optical circuits to be setup manually on human timescales. The result is low link utilization. OBS introduces statistical multiplexing where a queue of packets with the same source and destination are assembled into a burst (a much larger packet) and sent through the network together. Like OBS, the Mordia architecture has a separate control plane and data plane.

TDMA: Time division multiple access is often used in wireless networks to share the channel capacity among multiple senders and receivers. It is also used by a few wired networks such ITU-T G.hn "HomeGrid" LANs and "FlexRay" automotive networks. Its applicability to data center packet-switched Ethernet networks was studied in [22].

9. CONCLUSIONS

In this paper, we have presented the design and implementation of the Mordia OCS architecture, and have evaluated it on a 24-port prototype. A key contribution of this work is a control plane that supports an end-to-end reconfiguration time 2–3 orders of magnitude smaller than previous approaches based on a novel circuit scheduling approach called Traffic Matrix Scheduling. While Mordia is only one piece in a larger effort, we are encouraged by this initial experience building an operational hardware/software network that supports microsecond switching.

10. ACKNOWLEDGEMENTS

We received support from the National Science Foundation through CIAN NSF ERC under grant #EEC-0812072. Portions of this work were funded through a Google Focused Research Award, as well as through Ericsson and Microsoft. We acknowledge the support of the Multiscale Systems Center, one of six research centers funded under the Focus Center Research Program (FCRP), a Semiconductor Research Corporation program. We'd also like to thank Alex Snoeren and Geoff Voelker for providing invaluable feedback to this work.

11. REFERENCES

[1] M. Al-Fares, A. Loukissas, and A. Vahdat. A Scalable, Commodity, Data Center Network Architecture. In *ACM SIGCOMM '08*.

[2] M. Al-Fares, S. Radhakrishnan, B. Raghavan, N. Huang, and A. Vahdat. Hedera: Dynamic Flow Scheduling for Data Center Networks. *In NSDI'10*.

[3] W. Anderson, J. Jackel, G.-K. Chang, H. Dai, W. Xin, M. Goodman, C. Allyn, M. Alvarez, O. Clarke, A. Gottlieb, F. Kleytman, J. Morreale, V. Nichols, A. Tzathas, R. Vora, L. Mercer, H. Dardy, E. Renaud, L. Williard, J. Perreault, R. McFarland, and T. Gibbons. The MONET Project—A Final Report. *Lightwave Technology, Journal of*, 18(12):1988 –2009, dec 2000.

[4] D. Beaver, S. Kumar, H. C. Li, J. Sobel, and P. Vajgel. Finding a needle in haystack: Facebook's photo storage. *In OSDI'10*.

[5] G. Birkhoff. Tres Observaciones Sobre el Algebra Lineal. *Univ. Nac. Tucumán Rev. Ser. A*, 5:147–151, 1946.

[6] K. Chen, A. Singla, A. Singh, K. Ramachandran, L. Xu, Y. Zhang, and X. Wen. OSA: An Optical Switching Architecture for Data Center Networks and Unprecedented Flexibility. In *NSDI '12*.

[7] N. Farrington, G. Porter, Y. Fainman, G. Papen, and A. Vahdat. Hunting mice with microsecond circuit switches. In *Proceedings of the 11th ACM Workshop on Hot Topics in Networks*, HotNets-XI, pages 115–120, New York, NY, USA, 2012. ACM.

[8] 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 *ACM SIGCOMM '10*.

[9] J. E. Ford, V. A. Aksyuk, D. J. Bishop, and J. A.Walker. Wavelength add-drop switching using tilting micromirrors. *IEEE Journal of Lightwave Technology*, 17:904–911, 1999.

[10] A. Goel, M. Kapralov, and S. Khanna. Perfect matchings in o(n log n) time in regular bipartite graphs. In *Proceedings of the 42nd ACM symposium on Theory of computing*, STOC '10, pages 39–46, New York, NY, USA, 2010. ACM.

[11] D. Halperin, S. Kandula, J. Padhye, P. Bahl, and D. Wetherall. Augmenting Data Center Networks with Multi-gigabit Wireless Links. In *ACM SIGCOMM '11*.

[12] U. Hoelzle and L. A. Barroso. *The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines*. Morgan and Claypool Publishers, 1st edition, 2009.

[13] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner. Openflow: enabling innovation in campus networks. *SIGCOMM Comput. Commun. Rev.*, 38(2):69–74, Mar. 2008.

[14] R. N. Mysore, A. Pamporis, N. Farrington, N. Huang, P. Miri, S. Radhakrishnan, Subramanya, and A. Vahdat. PortLand: A Scalable, Fault-Tolerant Layer 2 Data Center Network Fabric. In *ACM SIGCOMM '09*.

[15] Hadoop: Open source implementation of map reduce. `http://hadoop.apache.org/`.

[16] C. Qiao and M. Yoo. Optical Burst Switching (OBS) - A New Paradigm for an Optical Internet. *Journal of High Speed Networks*, 8(1):69, 1999.

[17] R. Sinkhorn. A Relationship Between Arbitrary Positive Matrices and Doubly Stochastic Matrices. *The Annals of Mathematical Statistics*, 35(2):876–879, 1964.

[18] Glimmerglass 80x80 mems switch. `http://www.glimmerglass.com/products/technology/`.

[19] T. A. Strasser and J. L. Wagener. Wavelength-Selective Switches for ROADM Applications. *IEEE Journal of Selected Topics in Quantum Electronics*, 16:1150–1157, 2010.

[20] Y. Tamir and G. L. Frazier. High-performance multi-queue buffers for vlsi communications switches. In *Proceedings of the 15th Annual International Symposium on Computer architecture*, ISCA '88, pages 343–354, Los Alamitos, CA, USA, 1988. IEEE Computer Society Press.

[21] J. S. Turner. Terabit Burst Switching. *Journal of High Speed Networks*, 8(1):3–16, 1999.

[22] B. C. Vattikonda, G. Porter, A. Vahdat, and A. C. Snoeren. Practical TDMA for Datacenter Ethernet. *In EuroSys '12*.

[23] J. von Neumann. A Certain Zero-sum Two-person Game Equivalent to the Optimal Assignment Problem. *Contributions to the Theory of Games*, 2:5–12, 1953.

[24] 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 *ACM SIGCOMM '10*.