

Realizing RotorNet: Toward Practical Microsecond-scale Optical Networking

William M. Mellette
max@infocusnetworks.com
inFocus Networks
San Diego, CA, USA

Alex Forencich, Rukshani Athapathu
Alex C. Snoeren, George Papen, George Porter
University of California, San Diego
La Jolla, CA, USA

ABSTRACT

We describe our experience building and deploying a demand-oblivious optically-switched network based on the RotorNet and Opera architectures. We detail the design, manufacture, deployment, and end-to-end operation of a 128-port optical rotor switch along with supporting NIC hardware and host software. Using this prototype, we assess yield, synchronization, and interoperability with commodity hardware and software at a scale of practical relevance. We provide the first real-world measurements of Linux TCP throughput and host-to-host latency in an operational RotorNet, achieving 98% of link rate with 99th-percentile ping times faster than commodity packet-switching hardware. In the process, we uncover unexpected challenges with link-level dropouts and devise a novel and flexible way to address them. Our deployment experience demonstrates the feasibility of our implementation approach and identifies opportunities for future exploration.

CCS CONCEPTS

• **Networks** → **Network architectures**; **Network components**; **Data center networks**; • **Hardware** → **Networking hardware**;

KEYWORDS

Circuit-switching, Optical networking, Datacenter networks

ACM Reference Format:

William M. Mellette, Alex Forencich, Rukshani Athapathu, and Alex C. Snoeren, George Papen, George Porter. 2024. Realizing RotorNet: Toward Practical Microsecond-scale Optical Networking. In *ACM SIGCOMM 2024 Conference (ACM SIGCOMM '24)*, August 4–8, 2024, Sydney, NSW, Australia. ACM, New York, NY, USA, 23 pages. <https://doi.org/10.1145/3651890.3672273>

1 INTRODUCTION

Optical circuit switching has been instrumental in meeting the cost, power, and operational requirements of today’s production cloud [44] and machine learning [25] systems. Production deployments described in the open literature use relatively slow optical switching hardware, with reconfiguration times on the order of

milliseconds [52]. There has been extensive research on fast switching (microseconds to nanoseconds), which has produced evidence of further potential benefits [5, 6, 11, 14, 19, 21, 43, 61], but we are unaware of any commercial deployments of these systems.

Moreover, the practicality of these research proposals for production systems remains unclear, in large part because they are (understandably) limited to very small-scale hardware prototypes. While today’s production systems employ optical circuit switches (OCSes) with over 100 ports [25, 44], most research prototypes have fewer than ten ports and often use custom “one-off” transceiver technology. The potential for unanticipated issues when these prototype systems are scaled to a full deployment makes it difficult to assess the actual system cost and practicality. Yield, reliability, scalability, and hardware and software implementation and interoperability loom as potentially significant challenges.

In addition to the hardware complexity inherent in fast switches, many proposals to integrate them into network fabrics require collecting network-wide traffic demand information [14, 20, 29, 55] to compute a unified schedule [8, 30], which may be infeasible at scale. As a result, *demand-oblivious* networks have attracted significant interest [3, 5, 27, 45, 47, 57]: despite the fact that they sacrifice at most a factor of two in bandwidth, they are attractive in the context of circuit switching due to (1) their simple, distributed control plane and (2) their amenability to relaxed hardware requirements. While researchers have shown the feasibility of their hardware designs and demonstrated the potential performance gains of their network architectures through simulation and even emulation, to date, none of these proposals have been evaluated in practice, using production-scale hardware switches and commodity end hosts.

Two of the most promising demand-oblivious network designs, RotorNet [36] and its extension, Opera [34], rely on a novel switching technology known as a *rotor switch* to construct an all-optical datacenter network. While our prior work on RotorNet and Opera outlined a vision for a simple, scalable, high-bandwidth, low-latency network, it failed to address many practical concerns because large-scale rotor switching hardware compatible with commodity link technology simply did not exist. Specifically, while our initial work mocked up an 8-port switch, the device itself was actually a MEMS-based selector switch [37]. Similarly, the Opera network design was prototyped using a Tofino-based electrical packet switch as a stand-in for the rotor switch. Moreover, both systems employed custom network transport protocols (RotorLB and NDP [22], respectively) at the end hosts, preventing straightforward end-to-end performance testing using standard tools.

In this paper, we provide the first real-world evaluation of RotorNet and Opera using an actual rotor switch [33]. We discuss

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

ACM SIGCOMM '24, August 4–8, 2024, Sydney, NSW, Australia

© 2024 Copyright held by the owner/author(s). Publication rights licensed to the Association for Computing Machinery.

ACM ISBN 979-8-4007-0614-1/24/08...\$15.00

<https://doi.org/10.1145/3651890.3672273>

our experience manufacturing, deploying, and performing system-level testing of a 128-port optical rotor switch connecting a rack of commodity servers using commercial transceivers and an unmodified Linux networking stack. In addition to demonstrating high-performance operation with end hosts running standard Linux, our deployment highlights the robustness and flexibility of the RotorNet/Opera approach. While our switch has 100% port yield on all 256 single-mode fibers, the rotor itself exhibits minor manufacturing defects. We develop and deploy a novel end-host-based masking technique—akin to bad-sector mapping in commercial hard drives—that allows TCP transfers to extract 99.5% of the achievable link capacity despite the imperfections, demonstrating the practicality of the rotor concept. We also show how Opera’s expander-graph-based path selection allows deployments to make full use of potentially sub-optimal matchings installed in the rotor switch.

By taking RotorNet and Opera to the next level of practicality, we hope to provide insight into their potential for adoption as well as share techniques and lessons learned that apply more generally to optical networking. Our contributions include:

- Articulating the benefits of a network design capable of delivering both logical (§ 2.1) and physical transparency (§ 2.2) to help guide the design of future optically-switched networks;
- Providing a detailed description of the design, fabrication, and characterization of a 3U rack-mounted 128-port 7-microsecond optical rotor switch compatible with commodity off-the-shelf, unamplified 10 Gb/s datacom transceiver modules (§ 4);
- Describing and validating a method of forming compact, high-density, low-loss, passive optical transforms (matchings) using structured micro-optics (§ 4.2.1);
- Characterizing temporal link error bursts caused by manufacturing defects in the rotor switch and developing and validating a NIC-based approach to masking those error events using precise timing information (§ 4.4);
- Demonstrating lossless TCP traffic running end-to-end through an unmodified Linux stack, FPGA NIC, and rotor switch, with as little as 2% throughput overhead due to switch reconfigurations and masked error bursts (§ 5.2); and
- Demonstrating lossless end-to-end ICMP ping latencies through the OS, FPGA NIC, and rotor switch comparable to using a commodity packet switch and NICs, with minimal latency degradation and no packet drops even with interference from contending link-rate transfers (§ 5.3).

This work does not raise any ethical issues.

2 BACKGROUND

Datacenter network fabrics are under constant pressure to deliver higher link speeds to a larger number of end points. Physical and manufacturing constraints are driving operators away from electrical links connected by silicon-based packet switches [38] to optical links connected by circuit switches. These so-called *lightwave* fabrics achieve significant cost and power savings by pushing electronic modulation, detection, buffering, and scheduling to the network edge in an attempt to offer a “transparent” network core. Transparency, however, comes in different flavors, which we term

Switch Technology	Transparency		Switch Arch.
	Physical	Logical	
SOA [45]	✗ (λ crosstalk)	✓ (ns)	xbar
Tunable laser & AWGR [5]	✗ (Fixed λ plan)	✓ (ns)	xbar
MEMs [52]	✓ (Space switch)	✗ (ms)	xbar
Rotor	✓ (Space switch)	✓ (μ s)	non-xbar

Table 1: Summary of optical switching technologies. λ means “optical wavelength channel;” ns, μ s, and ms are time units; xbar means crossbar.

logical and *physical*, each with their own ramifications. While the ideal lightwave fabric would be transparent both logically and physically, achieving this goal is difficult in practice. Hence, most existing designs pursue one at the expense of the other. RotorNet [36] (and its extension, Opera [34]), on the other hand, provides a unique middle ground.

2.1 Logical transparency

For lightwave fabrics to have widespread practical utility, they must be able to approximate the performance of packet-switched networks for the application(s) running over the fabric. We refer to this as *logical transparency* and, generally speaking, this capability relates to how quickly the optical switch (and the rest of the network) can reconfigure its connectivity. Unlike packet switches—which deliver seamless multiplexing of packets—optical switches incur a technology-dependent delay in order to change connections. The slower the reconfiguration, the less frequently connections can be switched, and the lower the logical transparency.

Different assumptions about application traffic requirements have led to different design points based on disparate optical switching technologies. Initial academic approaches focused on hybrid fabrics that attempt to hide the optical switching latency by opportunistically employing circuit-switched paths in combination with traditional packet-switch fabrics [15, 29, 55]. The inherent complexity of multiple, disparate backbones has limited adoption of hybrid approaches, driving a continued quest to decrease switching time. Practitioners have found millisecond-scale switching technologies [52] sufficient for traffic engineering [44] and use cases with predictable workload dynamics [25], but they remain impractical for widespread use in the general datacenter. Researchers have shown that optical wavelength-switching technologies can approach even the most aggressive “holy grail” target of per-packet switching (i.e., nanosecond scale) [5, 45] but they sacrifice transparency along another dimension.

2.2 Physical transparency

A *physically transparent* lightwave fabric is agnostic to the link technology and data rate used to send and receive data traffic. A high degree of physical transparency is required in production datacenter [44] and wide-area [24] fabrics to allow compatibility with multiple generations of link technologies. This flexibility is critical because it allows amortization of the (significant) total cost of ownership of a lightwave fabric across many years.

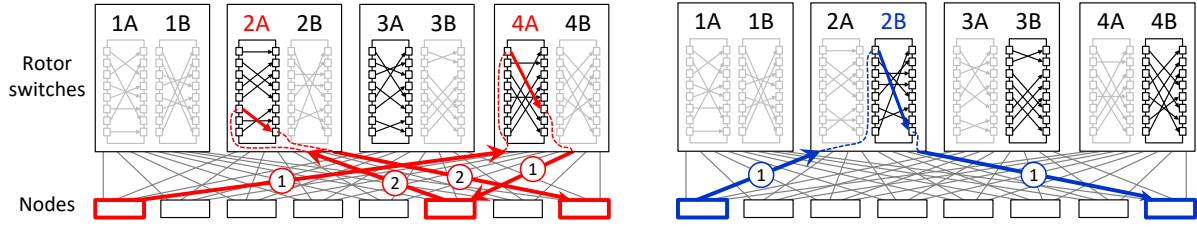


Figure 1: An example Opera topology with eight nodes and four rotor switches (reproduced from [34, Fig. 5]). Two different paths from node 1 to node 8 are highlighted: (left) a two-hop path in red, and (right) a one-hop path in blue. Each direct inter-node connection (blue) is implemented only once per matching cycle, while multi-hop paths (red) are available between each node-pair at all times, enabling low-latency forwarding.

New generations of optical links increase the overall data rate by adding additional wavelengths and/or by increasing the modulation rate (and therefore spectral bandwidth) of each wavelength channel. While traditional space-switched technologies like MEMs [52] are unperturbed by such changes, wavelength-switched networks require some form of “elastic grid” wavelength planning to accommodate changing spectral bandwidths¹. Existing research proposals for wavelength-switched datacenter networks [5, 45] lack such functionality, requiring either the insertion of additional devices into the optical path (along with the cost, power, and link budget implications), or an extension or a complete rebuild of the network with each link technology upgrade.

2.3 An alternative tradeoff

RotorNet represents a deliberate balance between physical and logical transparency informed by the limitations of existing optical switching technologies and their associated economic implications. Rather than trade off one form of transparency against the other, our prior work identified an alternative concession [37]: by restricting the degree of configurability of an optical switch, a *selector switch* can increase reconfiguration speed (i.e., logical transparency) by orders of magnitude without restricting spectral transmission properties (physical transparency).

The RotorNet and Opera network architectures permit a further optimization: they only require optical switches to repetitively cycle through their restricted connectivity. This additional restriction leads to a fundamentally different and specialized type of optical switch, known as a *rotor switch* [36], and we discuss its design further in § 4.2.1. It is worth pointing out that while the configurability restrictions of rotor switching reduce logical transparency at the level of a single switch, the RotorNet and Opera architectures restore logical transparency at the overall network level via their topology and routing design (see § 3.1).

Table 1 contrasts rotor switching with other optical switching technologies. Rotor switching is 1000× faster than MEMS while remaining physically transparent (see § 4) and its moderate hardware-level logical transparency can be enhanced at the system level by using multi-hop paths to reduce latency as in Opera [34] (see § 5.3).

¹This functionality typically requires a specific kind of wavelength-selective switch (WSS) and optical amplification to overcome the loss in the WSS [14] leading to high system cost and complexity, relegating its use to systems where the cost can be amortized over link distances that are much longer compared to datacenter links.

3 ARCHITECTURE

We begin this section with a brief review of the RotorNet and Opera designs. We then highlight the data forwarding and control subsystems we developed and implemented to realize a complete end-to-end system, including NIC-based time synchronization, routing, and forwarding.

3.1 RotorNet and Opera

RotorNet [36] addresses two of the primary challenges facing optically-switched networks: (1) optical circuit scheduling complexity and (2) OCS hardware scalability.

Scheduling: Because optical switches cannot inspect packets to make routing decisions, their connectivity must be set via another mechanism. Doing so using a centralized scheduler based on overall network traffic demand is challenging at large scale and high speed [4, 30, 39]. Instead, RotorNet employs a fixed, cyclic connectivity schedule at the physical OCS level and a version of Valiant load balancing [53] called RotorLB to route traffic over the predetermined (but time-varying) optical paths. This provides 50-100% of ideal throughput regardless of traffic skew while avoiding the complexity of a centralized scheduler.

OCS scalability: As mentioned in § 2.3, RotorNet relaxes OCS hardware requirements because it does not require arbitrary, any-to-any connectivity at each optical switch. This can be leveraged to design rotor switches with faster switching speed, larger radix, and lower cost than conventional OCSes, as discussed in more detail in § 4.2.1.

As originally conceived, RotorNet does not directly support low-latency traffic delivery. Opera [34] is an extension to RotorNet that retains bandwidth-efficient delivery of “bulk” traffic while adding support for low-latency delivery of a limited amount of latency-sensitive traffic. Built using the same rotor switch hardware, Opera leverages RotorNet’s parallel rotor switches and specifies a particular set of connection patterns (or *matchings*) within each rotor such that at all times the total set of active parallel connections forms an expander graph. Expander graphs have many potential short paths between all source/destination pairs [26], and Opera uses this property to guarantee the existence of low-latency, multi-hop “cut through” routes at every instant within the matching cycle. This design enables latency-sensitive traffic to be immediately sent to

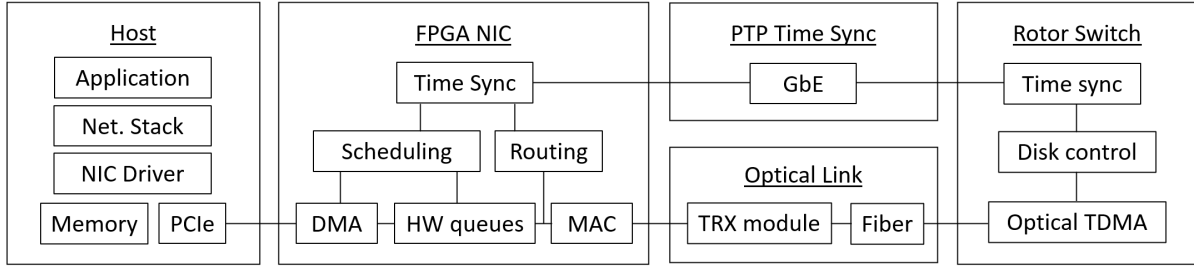


Figure 2: The end-to-end architecture of our rotor network implementation, capable of communicating user space application data between nodes with zero packet loss using commodity protocols and transceiver modules paired with custom NICs and rotor switch.

any destination at any time without waiting for a direct connection. Figure 1 reproduces an example from the original proposal in which eight nodes are connected using four rotor switches, each of which cycle through two distinct matchings. At any moment in time, each node is connected to four destinations *directly* (one through each switch) while the remainder can be reached by a (likely short) *indirect* path.

3.2 Synchronization and traffic admission

Both RotorNet and Opera can serve as the backbone of a multi-layered network; they can directly connect individual servers or electrically-aggregated sets of computing units (e.g. a traditional server rack or a dense cluster of accelerators [40]). In either case, each node in the optical network requires multiple distinct uplinks, one per rotor switch, and all nodes must be time-synchronized with the rotors so data traffic can be sent during the appropriate connections as they become available through the switches. The rotor switch’s microsecond-scale reconfiguration speed means synchronization is readily achievable even at scale.

In this paper, we focus on a direct-to-host network as it allows us to implement required edge functionality in host NICs. Figure 2 depicts the data and control flow (i.e., time sync, scheduling, and routing) between the hardware components in our deployment. We found that running the Precision Time Protocol (PTP) over the built-in lower speed GbE server management ports realizes sub-microsecond clock synchronization across the entire network.

To enable time-synchronous traffic admission from each node, we use an FPGA-based NIC running Corundum [18], a 100-Gb/s-capable, open-source, FPGA prototyping platform (see Appendix B). Corundum has an internal PTP clock module that is synchronized with the host’s management NIC via PCIe, as well as an application block for custom packet-processing logic. This timing information is used along with knowledge of the predetermined rotor schedule to precisely control the flow of traffic into the optical switch. Bulk traffic is transferred via DMA over the PCIe bus for injection into the network at precise times under the control of the NIC’s schedulers, while maintaining shallow hardware queues in the NIC to enforce tightly synchronized admission control. Custom logic implements time-based routing and forwarding within the application block on the FPGA. Other than running PTP client software, the host operating system and its applications do not need need to

be synchronized with the network. Details of how the rotor switch is synchronized to the hosts are provided in § 4.3.

3.3 Synchronous routing

We leverage the fact that rotor switches reconfigure in a predetermined sequence to precompute multi-hop routes offline and load hop-by-hop routing tables into the NIC at each node. The routing tables contain $N \cdot T$ elements: a next-hop for each of the N destinations, with T versions for the T topologies in the Opera architecture. (Note: At small scales $T = N$, but assuming the number of node uplinks has even a weak proportionality to the number of nodes, the size of the routing tables can be proportional to N by overlapping rotor reconfigurations to reduce the number of topologies [34].) For example, server racks or multi-accelerator nodes commonly have multiple Tb/s of uplink bandwidth, enabling 10s to 100s of physical uplinks depending on packaging specifics. This can enable N/T ratios of 20 or more.

In our implementation, rather than simply indexing the routing table based on a packet’s destination IP address (like a conventional network), the NIC uses the current PTP-synchronized time to derive a secondary table index. This allows us to specify the outgoing NIC port of each packet to match the current optical network topology, ensuring packets go through the correct rotor switch on their next hop.

We also consider and address the corner case of packet routing across topology transitions. To avoid packets being sent out a NIC port into a reconfiguring rotor (which would drop the packets), we exclude the ports with impending reconfigurations from the routing calculation for a given topology. This allows time for packets experiencing extra queuing delays to drain from the NIC’s hardware transmit queues before the corresponding rotor reconfigures.

3.4 Forwarding

There are two options for forwarding traffic through the network: (1) a low-latency, but less bandwidth-efficient service based on Opera’s multi-hop “cut-through” routing and (2) a higher-latency, but more bandwidth-efficient service based on one- or two-hop “store and forward” routing as in RotorLB. Packets are marked depending on their latency requirements, which can be done on a per-socket basis using standard Linux tools.

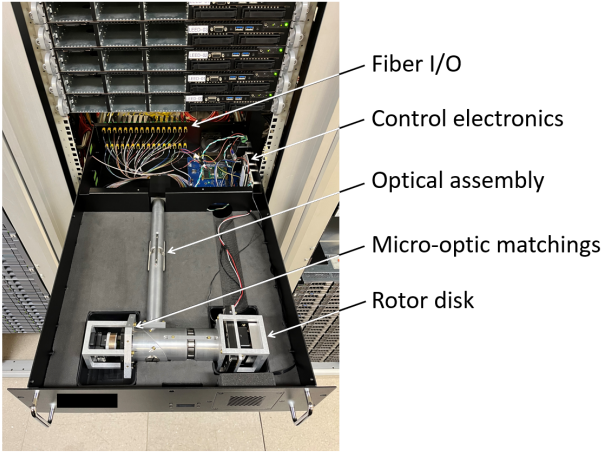


Figure 3: 3U rotor switch prototype installed in a machine room server rack with lid removed to show optics and electronics. The thermally-tolerant optical assembly floats between machined foam inserts for isolation from mechanical shock and vibration.

3.4.1 Cut-through forwarding. Packets marked as latency-sensitive are strictly prioritized in the NIC. Using FPGA-based NICs, we designed a mechanism to divert indirected latency-sensitive packets to the appropriate output NIC ports without requiring the packets to traverse the PCIe bus back to the host. This is critical for enabling low-latency delivery. As we show in § 5.3, the added latency of our cut-through forwarding is nearly imperceptible when probed with ping, and actually reduces tail latency compared to a commodity NIC. Appendix B provides additional details on our NIC implementation.

3.4.2 Store and forward. Packets marked as latency-insensitive are forwarded by the NIC at lower priority. In cases where these packets are being indirected through the NIC at an intermediate node (i.e., using RotorLB), there are two options: (1) if the NIC has a sufficient quantity of on-board RAM with sufficient memory bandwidth, such as high-bandwidth memory (HBM), these packets can be stored on the NIC itself to avoid traversing the PCIe bus back to the host, or (2) they can be sent to the host for storage in system memory. Commercially-available FPGA NICs with sufficient HBM exist [7, 48], but we do not have access to these devices at time of writing so indirect forwarding of bulk traffic is not currently supported.

4 ROTOR SWITCH

In this section, we delve into the design considerations, scalability, reliability, topology properties, and link-level performance of our 128-port rotor switch prototype.

4.1 Overview

We explicitly designed the rotor switch (Figure 3) to satisfy a number of practical deployment considerations, with a primary focus on compatibility with standard, unamplified commodity datacom transceiver modules. Minimizing transceiver changes is critical

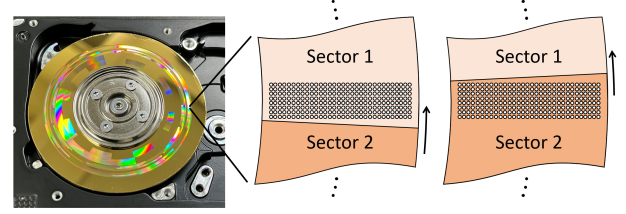


Figure 4: The rotor switch reconfigures by scanning the boundary between diffractive disk sectors across the spatial extent of the 256-element optical signal array (not to scale). A rectangular optical signal array reduces reconfiguration delay by reducing this scan time.

from a cost standpoint since these devices already account for the majority of networking cost in commercial data centers [32].

While we achieved 100% port yield in our prototype 128-port rotor switch, manufacturing imperfections caused short, temporally-localized signal drop-outs during operation (see § 4.4). Improved fabrication processes could address this issue, but we also developed a novel approach to mask these defects as described below in § 4.5.

4.2 Design

As discussed in § 2, our goal in rotor switch design is to balance physical transparency (which requires low levels of signal attenuation and other signal impairments) with logical transparency (which requires fast reconfiguration speed), while also considering manufacturability, reliability, and cost. The low-level hardware details of our switch are discussed elsewhere [33] and in Appendix A; here we focus on the aspects relevant to the overall RotorNet/Opera system.

4.2.1 Co-designed performance benefits. Traditional optical crossbar switches have a performance-scalability tradeoff because they must provide random access to an exponential set ($N!$) of input-output port matchings. This tradeoff manifests in technology-dependent ways; for MEMS crossconnect switches, the reconfiguration speed has an inverse proportionality to the switch radix because the MEMS micromirrors must perform both switching and spatial routing functions [35]. Due to the unique characteristics of the RotorNet/Opera architecture, each rotor switch only needs to (1) implement a small number ($\ll N$) of *predetermined* matchings and (2) switch between those matchings *sequentially*. These two differences allow us to *decouple* switching from routing, and are the fundamental reasons rotor switches can simultaneously achieve good physical and logical transparency, even for large radices. We expand on this claim below, discussing both the rotary switching engine and the matching implementation.

Rotary disk scalability and reliability: Conventional optical switches require at least one individually-articulated element per optical signal, which limits scalability due to the cost, complexity, and yield constraints of producing a large array of such switching elements. For example, MEMs switches use a pair of two-dimensional N -element arrays of two-axis tilting mirrors to “route” optical signals through the switch.

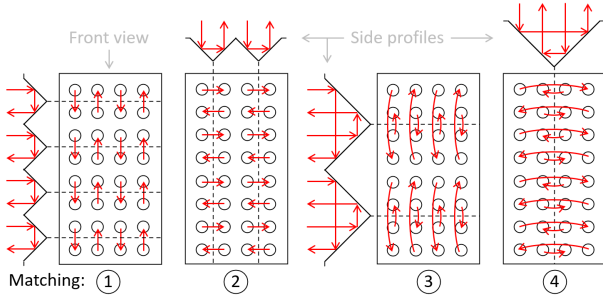


Figure 5: Physical layout of four micro-optic matchings for a scaled-down, 16-port rotor switch. Each matching shows a frontal view of how the 32-element 2-D array of optical signals is rearranged (rectangles), as well as cross-sectional profiles of the micro-optics accomplishing each transform (triangles). (A full-scale layout appears in Appendix A.)

The rotor switch could use a similar micromirror array, or even a single, large micromirror [37] to switch signals. However, we found that switching all signals together using a rotating disk patterned with diffractive sectors resulted in faster, simpler, and lower cost switching. Figure 4 shows the switching action as the disk scans from diffractive sector 1 to sector 2 across the 256-signal array (128 in and 128 out). Using a rectangular signal array (8×32 as shown in Fig. 4), we achieved 7- μ s reconfiguration for a 128-port switch, which is $1,000\times$ faster than MEMS or piezo switches (the only other switching technologies demonstrated to scale to a comparable port count with low insertion loss [42, 52]). Scaling up the radix of the rotor switch to thousands of ports can be realized by adding more signals to the optical array—no additional active devices or control signals are required. Using an optimized aspect ratio for the optical signal array, we anticipate being able to scale the rotor switch to thousands of ports while maintaining the reconfiguration speed [33].

The primary failure mode of the rotor switch is the rotary disk spindle. We employ a commodity hard drive spindle in our design, which has been cost and performance-engineered over decades for high reliability. The mean time to failure of an enterprise-grade hard drive unit is about 2 M hours [51], translating to an annual failure rate of about 0.04%. However, the overwhelming majority of failures are due to read-head crashes, whereas spindle failures are exceedingly rare [10] (our switch has no read head). Moreover, the overall network has sufficient path diversity to gracefully handle multiple rotor switch failures [34].

Micro-optic matchings and topology: Prior work has proposed implementing optical matchings using manual fiber patch panels [62], robotic patch panels [56], MEMS switches [15, 52], and even ceiling-mounted “disco-ball” reflectors [20]. Because our rotor switch design switches a dense array of optical signals in free space (as discussed above), we are able to employ a unique, high-density, and low-loss matching technology: simple, regularly-patterned, micro-optic structures. This technology was proposed over three decades ago for multi-stage interconnects [23]; here, we repurpose it for time-division interconnections. Figures 5 and 6 show a scaled-down version of the physical and logical structure of the matchings,

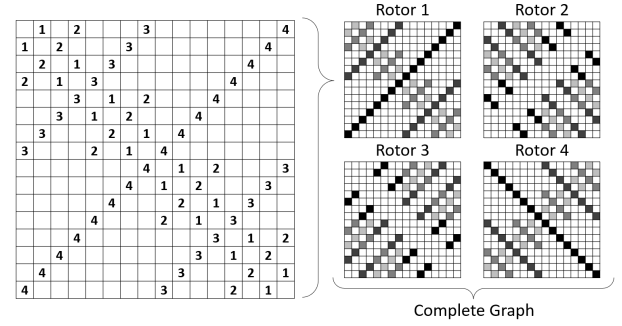


Figure 6: (left) The adjacency-matrix representation of the four matchings in Figure 5, which covers 1/4 of the 16-node graph. (right) By permuting the input cabling to four 16-port rotor switches each containing identical internal micro-optic matchings, the connectivity of all four rotors can be made completely orthogonal. Taken together, the four rotors then cover the complete graph. This example demonstrates how structured, replicated micro-optic matchings can cost-effectively implement the overall RotorNet/Opera topology.

which are fabricated by cutting microscopic grooves into a metal substrate (see Appendix A).

By permuting the fiber cabling structure between the rotor switch bulkheads and the internal fiber array that interfaces with the matchings, we can realize sets of completely orthogonal (or disjoint) matching patterns (Fig. 6 (right)). For a RotorNet/Opera design with multiple rotor switches, there exists a set of cable permutations that allow us to cover the entire fully-connected input-output node graph using the same set of micro-optic structures internal to each rotor switch. This cable permutation feature provides significant cost savings for multi-rotor networks because only one set of micro-optics is needed for all the switches and the manufacturing cost of permuting the fibers is negligible.

4.2.2 Design hardening, manufacturability, and cost. A major goal in fabricating the 128-port rotor switch was to move beyond laboratory benchtop prototypes relying on exotic and costly one-off components, and deliver a practical, rack-mountable switch based on scalable and cost-effective manufacturing processes.

We chose the switch’s optomechanical materials and geometry to tolerate a wide range of thermal conditions (it was deployed and operated in a hot/cold aisle machine room with large temperature gradients). We also designed in tolerance to mechanical strain, shock, and vibration through material selection and packaging (the optical components float between shaped foam inserts for mechanical isolation from vibration from server fans coupled through the rack rails). The switch is equipped with a processor running Linux, which runs the rotor control firmware and is accessible via remote access over a GbE management interface.

Figure 3 shows the packaged switch prototype, including optomechanical assembly, control electronics, Ethernet management interfaces, and 32 8-fiber MPO bulkhead connectors inside a 3 RU rack-mount enclosure. The switch was rack-mounted along with servers, packet switches, and other machine room hardware above and below the switch in the same rack in a hot/cold aisle machine

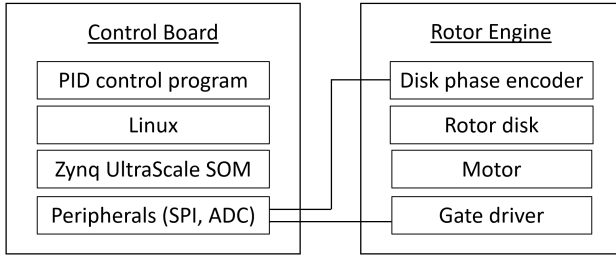


Figure 7: The control scheme used to synchronize the mechanical frequency and phase of the rotor switch’s disk. A PID control program running on Linux reads the disk position via the phase encoder, compares it to the reference, and adjusts the speed command to the motor’s gate driver.

room, subjecting the switch to vibration and temperature gradients during operation.

The switch subsystems leverage mature and high-volume manufacturing processes. The matchings were fabricated by diamond ruling; the imaging lenses were ground, polished, and coated from standard glass blanks; the optomechanics were fabricated through conventional machining; the rotor disk was produced by nanoimprint lithography; and the spindle was repurposed from a commodity hard drive. At high volume, the cost to produce these components is low enough that the highest per-port cost contribution is the fiber array. Because the rotor switch employs the same fiber array technology used in production optical switches [52], we expect the overall per-port cost to be comparable.

4.2.3 Partitioning for limited-scale deployment. Our rotor switch could be used to support a 128-node deployment, but this would require access to 128 nodes and the manufacture of additional rotor switches to operate in parallel. Our initial deployment targeted 16 nodes and required 4 parallel rotor switches (see § 5). To support this configuration, we partitioned our large switch into multiple “sub”-rotors by dividing the surface of the diffractive rotary disk into four staggered tracks, each intersecting a portion of the optical signal array. Rather than switching all 128 ports together, groups of 32 ports are switched independently. By offsetting the angular position of each track, we used the single physical rotor switch as four 32-port sub-rotor switches with their reconfigurations staggered in time.

4.3 Switch-Host Synchronization

The rotor switch needs to be synchronized to the hosts so injected traffic is carried over the intended optical paths. This requires aligning the rotation frequency and phase of the rotor switch’s motor (see § 4) to the frequency and phase of a common precision clock running on all the hosts. As discussed in § 3.2, we used the IEEE-1588 Precision Time Protocol (PTP) as the reference clock in our implementation. The servers and rotor switch are all synchronized using PTP over the same copper GbE network used for cluster management (see Fig 2). While far from optimal, we found PTP had a jitter of less than 1 μ s, making it sufficient for our needs. PTP is also commoditized to the point that it is supported on our servers’ stock

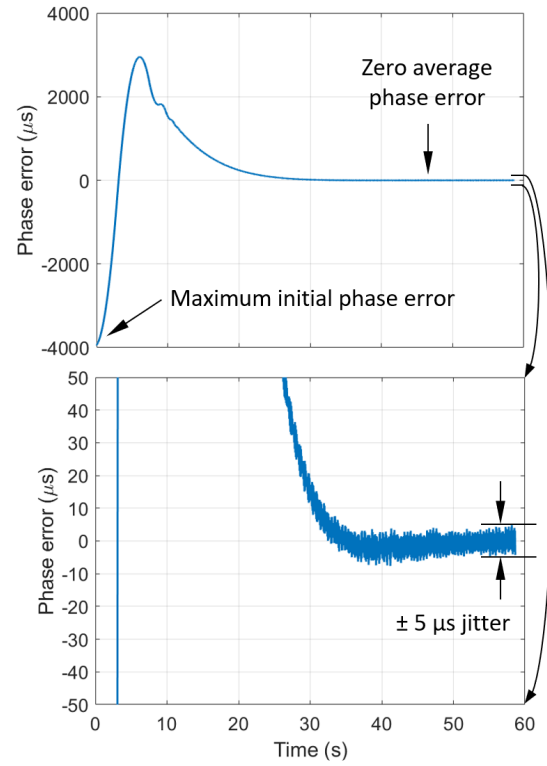


Figure 8: (Top) Phase locking performance of our rotor control loop, pulling in from a worst-case initialization point of 180° out-of-phase (@4 milliseconds) to the steady state. (Bottom) Zoomed in view showing a steady-state phase jitter of ± 5 microseconds.

GbE motherboard interfaces, meaning synchronization incurred no extra cost.

Figure 7 illustrates the scheme used to control the frequency and phase of the rotary disk. A custom motor controller based around a Xilinx Zynq XCZU4CG MPSoC on a Trenz TE0803 SoM and a Trinamic TMC4671 motor controller was used to drive the rotor switch (See Fig. B.2 in Appendix B). The controller runs a program based on proportional–integral–derivative (PID) control that minimizes the error between the rotor disk’s phase and a reference phase signal. Both phase signals use a pulse-per-revolution (PPR) format: the feedback PPR signal from the rotor is generated by an optical encoder that reads a PPR track physically printed on the disk, and the reference PPR signal is generated electronically based on PTP time.

Figure 8 shows the convergence of the control loop after (a worst-case) startup and the measured jitter in steady state. We achieved a steady state rotor phase-lock accuracy of $\pm 5 \mu$ s, which was the limiting factor in timing jitter in the overall system (since PTP jitter was $< 1 \mu$ s). While phase lock accuracy could likely be improved using multi-PPR encoder feedback, the current performance is on the same order as the 7- μ s switch reconfiguration delay and the ~ 10 –50- μ s signal acquisition time.

By virtue of the distributed nature of PTP, our control scheme naturally scales to support systems with multiple rotor switches. We validated that two physically isolated rotors could be synchronized to within $\pm 6 \mu\text{s}$ relative to the reference time, which is nearly the same jitter as a single rotor. This provides evidence that large-scale systems composed of multiple rotor switches are practical to synchronize.

4.4 Link-level characterization

Next, we performed link-level testing of the fabricated rotor switch. Unlike traditional static networks, lightwave fabrics require optical links (1) reacquire the optical signal each time the OCS reconfigures and (2) have sufficient link power budget to accommodate the signal impairments of the OCS. Because commercially available optical modules are designed for static networks, neither of the above features are explicitly supported in existing datacom standards. We developed characterization methods [28], described in Appendix C, to determine which transceiver modules would work in the rotor network, and chose the best performing commercial 10 Gb/s-line-rate modules for use in our system. As discussed in Appendix C, 10 Gb/s operation is not a fundamental limit of our system, rather a consequence of existing commodity modules not supporting fast signal acquisition and sufficient link budget at higher serial data rates.

To test all 128 input and 128 output ports, we connected the rotor switch using 32 8-fiber cables to 32 commercial off-the-shelf QSFP28 PSM4 datacom optical transceiver modules, which were in turn installed in 16 dual-port FPGA NICs. Each transceiver has four transmit lasers and four receivers, for a total of 128 transmitters and 128 receivers. The FPGAs were used to send a 10-Gb/s pseudorandom bit sequence (PRBS) through all of the transmitters, and check for bit errors at each receiver. The error detection logic was synchronized to the rotor switch via PTP, which enabled the error counts to be binned in $2\text{-}\mu\text{s}$ -wide time bins, allowing us to gather high-resolution time-resolved bit error ratio (BER) data for all 512 unique time-division paths (128 ports $\times$ 4 matchings) in one experiment.

Figure 9 illustrates the results. The outcome was 100% yield on both ports and matchings, experimentally validating our claims regarding the simplicity and scalability of our rotor switch design. While all paths through the switch were able to transmit data, we noticed a number of temporally-localized signal dropouts—some expected and some unexpected. The expected dropouts are due to rotor reconfigurations (marked “R” in Fig. 9). We traced the unexpected “blips” in BER performance back to microscopic manufacturing defects in the rotary disk. Because we need a large-area disk and small-area optical signals to achieve fast, high-radix switching (Figure 4), our design is sensitive to a small number of tiny imperfections on the disk surface that get scanned across the optical signal array as the disk rotates.

A beneficial feature of the commodity datacom transceiver modules we used was that they were able to reacquire the optical signal on microsecond timescales after both reconfiguration and defect events despite not being explicitly designed to support this. Figure 10 shows histograms of the signal reacquisition time for both reconfigurations and defects. Note that the phase synchronization

accuracy of the disk itself is $\pm 5 \mu\text{s}$, meaning the underlying reacquisition time of the modules is about $10 \mu\text{s}$ less than indicated in the figure.

While the commercial optical modules worked surprisingly well at 10 Gb/s even with the rapid disk defect events and the insertion loss through the rotor switch, when we tried to operate the system at 25 Gb/s, about 25% of the 512 connections through the switch did not perform well. These nonperforming connections are likely the result of reduced transceiver link margin at the higher data rate, coupled with transmit power and receiver sensitivity variations between the modules. These issues are not fundamental; indeed, production lightwave fabrics have addressed this issue by proprietary customization of the transceiver design to increase link budget at higher data rates to accommodate the signal attenuation of the OCS [31].

4.5 Error masking

Fortunately, because both the spatial position of the defects and the disk speed is constant, the signal dropouts occur at known, regular intervals. We developed a semi-automated network initialization routine that collects and processes the time-resolved BER data, determines the location of all defects, and leverages the same guard-band mechanism we employ to pause frame transmission during reconfiguration events to pause transmission during defect dropouts as well. We found that running this routine once at network initialization was sufficient to prevent any link errors during subsequent operation, but also note that this functionality could be used to debug link quality on a link-by-link basis after network maintenance or to help diagnose grey failures. It should also be possible to monitor the link quality during operation via signals from the PHY logic and adjust accordingly.

As discussed in § 3.2, each host maintains tight control of when frames are emitted into the network. Each host NIC maintains a 4096-element bitmask (corresponding to one complete revolution of the switch disk) of $2\text{-}\mu\text{s}$ timeslots for each output port, where the bitmask indicates whether frames may be emitted during that time interval or not. We implemented a guard-band enforcement module on the NIC that prevents transmission during masked intervals. This module is synchronized to the rotor switch via PTP and controls the flow of data right before it enters the transmit MACs to be sent to the optical switch. As we show in § 5.2, this enables fully lossless data transmission through the switch with near-zero overhead.

5 DEPLOYMENT

Next we turn to addressing the question: is our prototype rotor switch capable of supporting the RotorNet and Opera networks at an end-to-end system level in practice?

5.1 Topology, schedule, and flexibility

To answer this question, we installed the rotor switch in a standard 19" rack in our machine room (shown in Figure 3) along with 17 servers (1 control node and 16 cluster nodes). Each of the 16 cluster servers has eight 10-Gb/s optical connections (i.e., 80 Gb/s per server) to the rotor switch via an FPGA-based NIC implemented

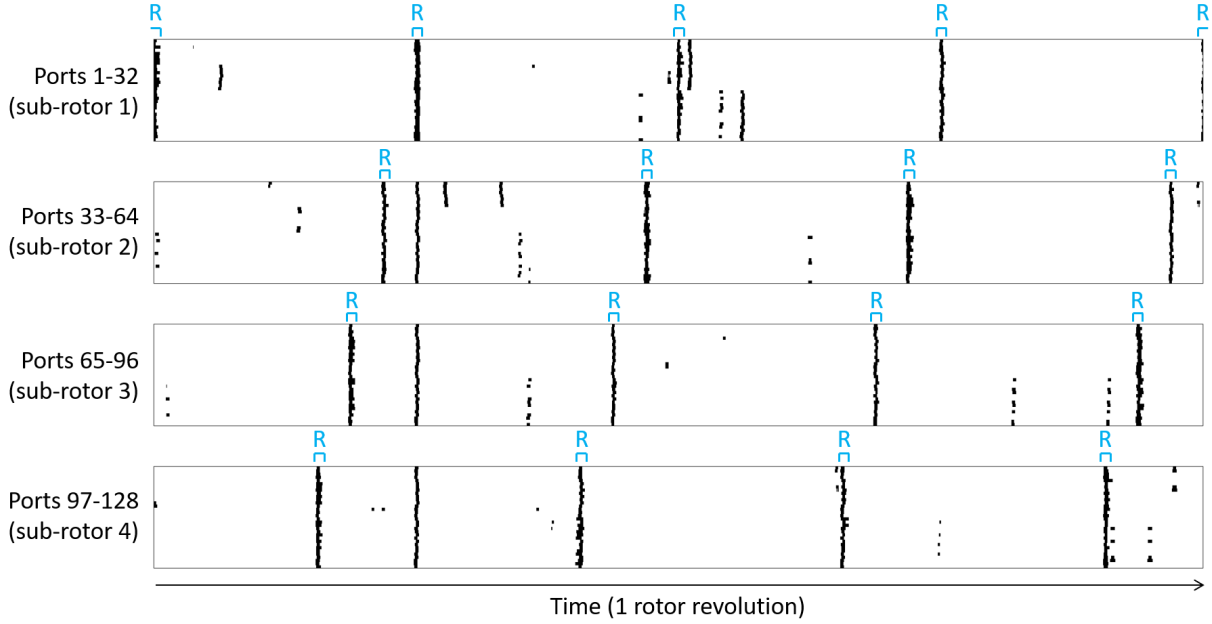


Figure 9: Time-resolved bit error rate performance at 10 Gb/s for all 512 paths through the rotor switch. Note that the figure overlays several thousand rotor revolutions and therefore gives a complete picture of timing jitter. For clarity, we use a binary plot: one or more bit errors in 75 million transmitted bits corresponds to a black data point; zero errors corresponds to a white data point (each data point is 2- μ s wide). Some of these error bands are expected due to switch reconfigurations (marked “R”), which are intentionally staggered between sub-rotors. All other errors, including the full “vertical line” errors on sub-rotors 2-4 are caused by defects in the rotary disk. We map and mask these defects at the link level to avoid packet corruption when operating the network.

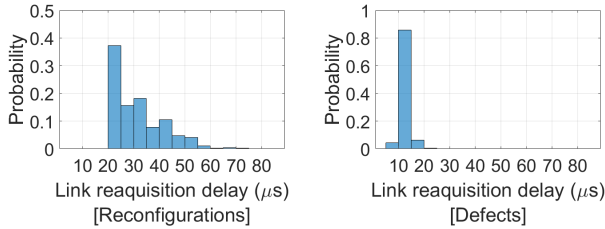


Figure 10: Link reacquisition delays for 10 Gb/s off-the-shelf optical datacom modules for (left) rotor reconfigurations and (right) rotor defects, including 10 μ s of rotor phase jitter. The sum of all the delays constitute an overall 2% overhead in data transmission capacity.

using the Corundum framework [18].² The FPGA NICs used in the servers are a mix of Alpha Data ADM-PCIE-9V3 (XCVU3P) and Silicom fb2CG@KU15P (XCKU15P). Since both QSFP28 ports on each NIC are used to connect to the optical switch, time synchronization must be done via the host system (see § 4.3). Figure 12 summarizes our deployment configuration.

²While Corundum currently supports links up to 100 Gb/s, eight separate 10-Gb/s links was preferred due to the deployment specifics.

Figure 11 shows the connectivity schedule of our testbed from the perspective of a server’s NIC. As discussed in § 4.2.3, we intentionally partitioned our 128-port rotor switch into four 32-port sub-rotors with staggered reconfigurations. This allows us to both demonstrate the feasibility of a 128-port rotor switch while employing the same hardware to implement multiple, lower-radix sub-rotors for a smaller-scale deployment of a full multi-rotor network. The partitioning means we “locked-in” a number of topological and scheduling design choices: (1) the degree of staggering between sub-rotors, (2) the matching sequence of each sub-rotor, and (3) the fiber cable permutations to each sub-rotor. In other words, these properties were physically encoded into the switch during manufacture.

Notably, these design choices were originally intended to support a 32-server deployment that would include a second rotor switch, rather than the current deployment of a single (partitioned) switch supporting 16 servers. Remarkably, by simply re-computing routes in software, we are able to fully leverage what could be considered a suboptimally-designed rotor switch in this alternate deployment scenario without any bandwidth waste or other compromise in performance. The surprising flexibility of what might be regarded as a rather rigid switch design ultimately stems from the properties of expander graphs. Because we designed the switch’s matchings following Opera’s design methodology, the resulting expander graph’s rich connectivity provided a large pool of low-latency routes capable of supporting a variety of possible deployment scenarios.

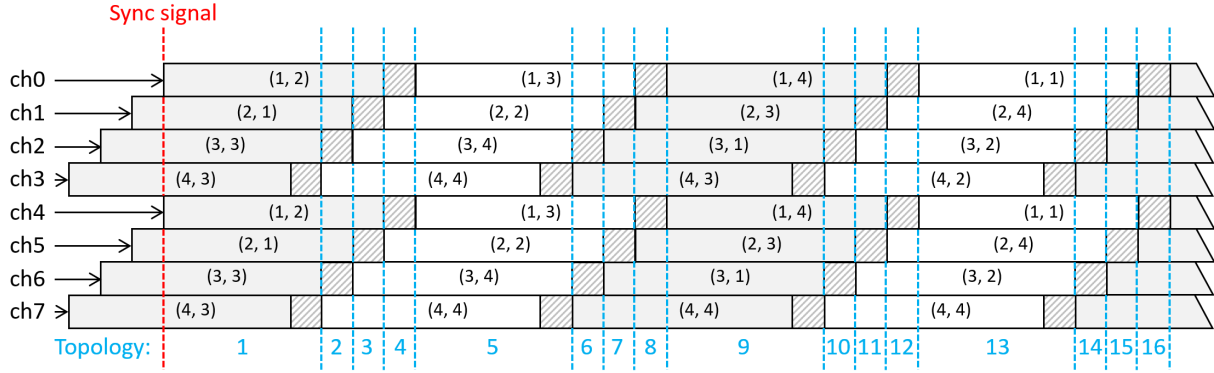


Figure 11: The TDMA connectivity of our testbed from the perspective of a NIC’s eight ports (ch0-7). We patterned multiple staggered tracks on the rotary disk to allow our 128-port rotor switch to be partitioned into four 32-port sub-rotors with reconfigurations that are staggered in time. Each matching is enumerated ($\langle$ sub-rotor-id $\rangle$, $\langle$ matching-id $\rangle$). The hatched regions preceding each reconfiguration indicate that the port is excluded from routing calculations for that topology to allow queues to drain before the impending reconfiguration (see § 3.3). The track staggering is not uniform because we designed the disk tracks for a 32-node deployment that would include a second rotor switch (which would fill in the gaps). Despite ultimately using the switch in a 16-node cluster without a second rotor, we suffered no loss in functionality or other operational issues.

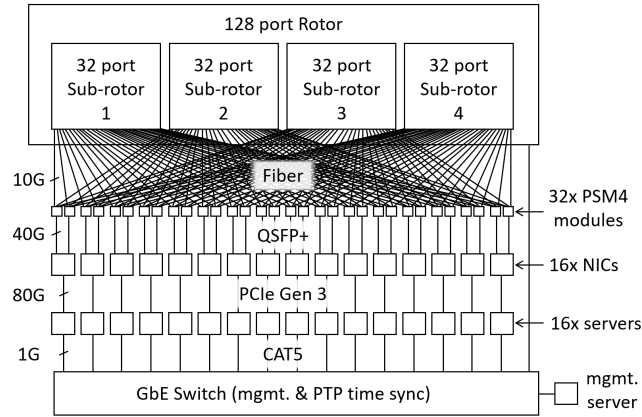


Figure 12: Deployment configuration: 16 cluster servers are connected via PCIe gen 3 x16 to 16 FPGA NICs. Each NIC is populated with two QSFP28 PSM4 optical modules, each of which has four PSM fiber connections to the four sub-rotors. Synchronization is done via PTP over a GbE network. The logical fiber breakout occurs inside the rotor switch enclosure; single 8-fiber ribbon cables between the modules and switch minimize cabling complexity.

5.2 Throughput

We first evaluated the ability to send end-to-end TCP traffic between remote Linux applications, with packets traversing the network stack, NIC driver, NIC, and rotor switch.

5.2.1 1-hop forwarding. We begin by considering the case of 1-hop forwarding, where traffic is only allowed admission into the network during the direct optical connection between a source and destination (which is 1/4 of the time for a single link on a single

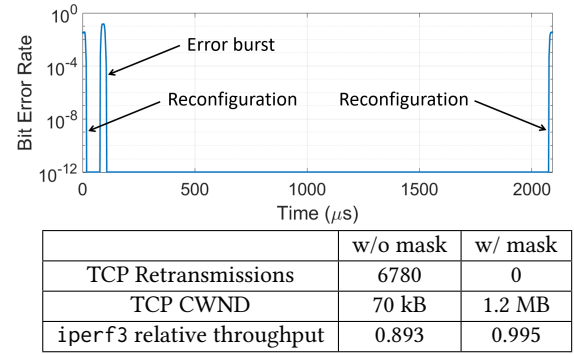


Figure 13: (top) BER timeseries for a rotor switch timeslot captured during network initialization. Left and right error bursts are due to reconfigurations. The mid-slot burst is due to a rotor disk defect. (bottom) Metrics for a 10-second iperf3 TCP transfer through this timeslot with and without masking the mid-slot error burst.

sub-rotor³). We enforce this by configuring the scheduling behavior of the FPGA NIC. We are immediately confronted with the issue of rotor defects: consider a timeslot with a burst of bit errors due to a rotor defect, as shown in Figure 13(top). We first simply attempt to send traffic through this timeslot without imposing an error mask: we start a TCP iperf3 server on one machine and an iperf3 client on the other and transmit for 10 seconds (over 1,000 rotor cycles). Unsurprisingly, we see a large number of retransmissions and a reduced congestion window as summarized in Figure 13(bottom). Despite the defect only occupying 2% of the total usable timeslot, throughput is reduced by 11% (relative to

³In the 16-node system, the 16 matchings are distributed across the 4 (sub)rotors, so each (sub)rotor contains 4 matchings.

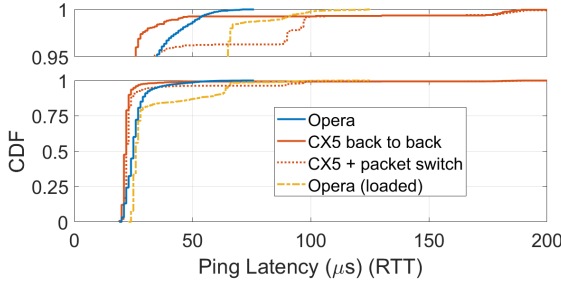


Figure 14: RTT latency statistics for kernel-to-kernel (ping) messages with detail on 95–100th percentiles for Opera and reference NICs.

the theoretically achievable value of $10 \text{ Gb/s} / 4 = 2.5 \text{ Gb/s}$ as a consequence of TCP backoff.

Next, we apply our error masking technique at the server’s NIC and repeat the same 10-second `iperf3` transfer. The NIC simply pauses data transmission during the problematic window, and standard Linux backpressure mechanisms prevent any software packet drops on the host during this brief interruption. (Note that this same standard Linux backpressure prevents packet drops during the other 3/4 of the timeslot when we cannot send 1-hop traffic.) As shown in Figure 13, we achieve zero TCP retransmissions, a large congestion window, and a throughput within 2% of the ideal 2.5 Gb/s (0.5% overhead when subtracting out the defect gap). Notably, none of this requires any modifications to the Linux application, TCP, or the Linux networking stack.

5.2.2 Multi-hop Opera forwarding. Next, we configure the NICs to allow traffic to traverse multi-hop paths (as in Opera), and repeat the throughput test. Now, instead of sending traffic only 1/4 of the time, we fully utilize the link—the NIC pauses data transmission briefly during reconfiguration events and rotor defects. A single source/destination flow traverses eight distinct paths (one direct path, six two-hop paths, and one three-hop path) employing all four rotors in the forward direction, and a similar—but distinct—set of paths on the return path. When traffic traverses multiple hops, the effect of defect masking along each leg of the path can accumulate, leading to a slightly higher overhead. The other potential issue is that packet reordering can occur due to path lengths changing as data is being transmitted. Remarkably, we achieve 9.3 Gb/s for the same 10-second TCP `iperf3` transfer (93% of link rate), with only a handful of retransmissions due to packet reordering. Such small amounts of reordering are commonly addressed with a reorder buffer, which we leave to future work.

5.3 Latency

Next, we evaluated traffic delivery from a latency perspective with multi-hop Opera forwarding enabled.

5.3.1 Unloaded latency. We begin by characterizing the latency between two servers in the absence of other traffic. First, we ran a ping flood between nodes spanning hundreds of rotor cycles to ensure we captured a set of latency statistics representative of the full set of network topologies (i.e., path lengths ranging from one

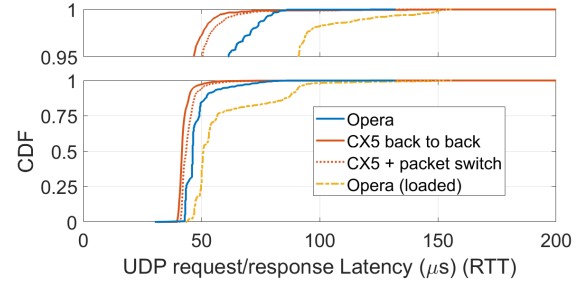


Figure 15: RTT latency statistics for userspace-to-userspace (UDP) request/response traffic.

to three hops). For reference, we repeated the experiment with commodity ConnectX-5 NICs connected (1) back-to-back and (2) through a packet switch to give practical upper bounds on performance. We found that Opera has comparable minimum and median latency with the commodity NIC, slightly worse 90th percentile latency due to packets taking multiple hops, and better latency above the 99th percentile, likely due to differences in packet batching behavior (see Figure 14). Next, because ping operates at the kernel level, we also tested the userspace-to-userspace latency using a simple UDP client/server program to understand the performance a typical sockets application would experience. Figure 15 shows the latency statistics for UDP again for both Opera and the commodity ConnectX-5 NICs.

5.3.2 Loaded latency. Next, we repeated the latency experiments for Opera under the presence of interfering cross traffic. We set up a line-rate TCP `iperf3` transfer between nodes *A* and *B* while measuring the latency of ICMP and UDP packets between nodes *C* and *B*. Because the network topology and routes constantly change, the paths taken by the bulk transfer and latency-sensitive packets vary, overlapping about 15% of the time on the $A \rightarrow B$ link when the shortest path between *C* and *B* traverses node *A* (i.e. $C \rightarrow A \rightarrow B$). The effect of interfering traffic in Opera is shown for ping and UDP in separate curves in Figure 14 and Figure 15, increasing the last ~15% of tail latencies by about 15 microseconds. Critically, no packets were dropped while merging traffic originating at *A* with indirect traffic using *A*’s NIC as an intermediate hop.

6 ONGOING WORK

While our rotor switch is fully operational, our implementation remains ongoing and we outline several areas of future work.

NIC: The current NIC design does not yet support the full feature set outlined in the original Opera and RotorNet proposals. In particular, it remains to add support for classifying and prioritizing the scheduling of latency-sensitive vs. bulk traffic originating at the host—as opposed to prioritizing cut-through Opera traffic, which is already supported. Also, as discussed previously, our FPGA-based NICs lack on-board high-bandwidth memory so we have not yet implemented RotorLB store-and-forward indirection for bulk traffic. Once we do, we will need to employ some form of flow/congestion control along multi-hop paths. Based on preliminary investigation,

link-level pause-frames provide a promising and straightforward path to supporting high utilization while minimizing queuing delays [54].

Host: Pending the availability of HBM at the NIC, we are considering performing RotorLB store-and-forward indirection using host memory. Our initial design and prototype exploration suggests the eXpress Data Path (XDP) [60] subsystem of the Linux kernel in combination with AF_XDP [2] as a promising route to enable this functionality, but considerable performance tuning remains.

Rotor switch: Full physical transparency requires extremely low signal attenuation, and our current prototype has a number of opportunities to reduce loss, which will enable operation at higher serial data rates. However, our prototype rotor switch has a wide spectral bandwidth (over 100 nm), meaning we can also increase the per-fiber data rate significantly by multiplexing a (potentially large) number of wavelengths onto each fiber. Indeed, challenges with increasing the serial line rate have driven commodity optical links to turn to wavelength (or space) parallelism to drive speed. It would also be interesting to test the prototype switch with a variety of different optical transceiver modules, including those specifically designed for optical switch compatibility.

ML applications: Given the recently reported benefits of light-wave fabrics for ML applications [25, 31], we are interested in understanding the potential benefit of using a variant of RotorNet for machine learning (ML) applications. For ML, only a limited set of collective communication patterns is needed support model training and inference [50, 56]. This limited set of patterns may be a good match to the limited connectivity and extensive scalability of RotorNet-like networks. As a specific example, our current prototype uses expander-like connection patterns for each network configuration [34]. Recently, these patterns have been shown to be nearly optimal for supporting the all-to-all collective communication pattern in directly-connected architectures [41]. Future work in this area includes determining the number and kind of communication patterns required for different ML models. Each of the required patterns could potentially be mapped into a network matching. Each matching could then be selected using a modified rotor switch or selector switch [37].

7 RELATED WORK

The benefits of lightwave fabrics based on optical circuit switching (OCS) in production datacenters have been described in several recent papers [25, 31, 44, 52]. These systems all employed a 128-port OCS based on MEMs optical switching technology [59]. This kind of optical switch reconfigures in tens of milliseconds, approximately 1000× slower than rotor switches. The reported production CapEx and OpEx savings have re-focused the systems research community on other potential system-level benefits of lightwave fabrics based on different OCS characteristics (switching speed, radix, blocking vs. non-blocking, etc.) compared to the MEMs devices used by the reported production systems.

7.1 Production lightwave fabrics

The initial use-case for OCSes in production datacenters was to enable “spine-free” topologies [44, 52]. These OCS-enabled networks

deliver 30% reduction in CapEx and 40% reduction in OpEx compared to the standard spine-full network using electrical packet switches. Using this directly-connected network with appropriate topology and traffic engineering provides 10% improvement in flow completion time and 30% increase in TCP throughput compared to a uniform mesh network.

A more recent use of OCSes to re-configure computing blocks within “superpods” designed for ML workloads provides up to 3× better system availability and model-dependent performance improvements up to 3.3× compared to a static network, despite the fact that the entire OCS network accounts for 6% of the total system cost [31].

7.2 Demand oblivious networks

Demand-oblivious scheduling based on Valiant load balancing has also been proposed for electrically switched networks [9]. Sirius [5] uses a demand-oblivious schedule that connects end points to each other in a round-robin fashion. Sirius demonstrated several configurations of fast wavelength switching using tunable wavelength sources and modulators. Fast optical switching on the order of 90 ns for one of the demonstrated configurations can yield less than 10% switching overhead for the reconfiguration period for 400-Gb/s packet-based traffic. Shoal [47] is a rack-level implementation of a demand-oblivious network that uses electrical circuit switches instead of OCSes. Due to the shorter reach of electrical signaling than optical fiber, it focused on performant energy-efficient rack-scale networks.

8 CONCLUSION

Demand-oblivious optical networks are attractive due to their simple, distributed control planes and amenability to simplified hardware designs. In this paper, we report on our experiences building such a network, including a 128-port optical rotor switch, a topology that supports high-bandwidth and low-latency packet delivery across multiple cluster configurations, NIC-based protocol implementations, and end-host support for an unmodified Linux platform. We have demonstrated the practicality of our rotor switch implementation with commodity optics and with low-cost manufacturing processes. Although the resulting switch had imperfections, we showed how to implement a masking feature in the NICs which enabled us to hide those imperfections from endhosts, resulting in less than a 2% throughput overhead for both switch reconfigurations and masking. The net result is a network that supports both TCP at near link rate and low-latency request/response performance that is comparable to a network based on commodity NICs and packet switches.

ACKNOWLEDGEMENTS

This work was supported in part by the Department of Energy “ARPA-e ENLITENED (DE-AR0000845)” as well as a grant from the California Energy Commission. We would like to thank the reviewers and our shepherd Yiting Xia.

REFERENCES

- [1] 2021. IEEE/ISO/IEC International Standard-Telecommunications and exchange between information technology systems — Requirements for local and metropolitan area networks — Part 3: Standard for Ethernet AMENDMENT 9: Physical

- layer specifications and management parameters for 25 Gb/s and 50 Gb/s passive optical networks. *ISO/IEC/IEEE 8802-3:2021/Amd.9:2021(E)* (2021), 1–270. <https://doi.org/10.1109/IEEESTD.2021.9650819>
- [2] AF_XDP 2024. AF XDP. https://www.kernel.org/doc/html/net/netfilter/af_xdp.html. (2024).
 - [3] Daniel Amir, Tegan Wilson, Vishal Shrivastav, Hakim Weatherspoon, Robert Kleinberg, and Rachit Agarwal. 2022. Optimal oblivious reconfigurable networks. In *Proceedings of the 54th Annual ACM SIGACT Symposium on Theory of Computing*. 1339–1352.
 - [4] Chen Avin and Stefan Schmid. 2019. Toward demand-aware networking: a theory for self-adjusting networks. *SIGCOMM Comput. Commun. Rev.* 48, 5 (jan 2019), 31–40. <https://doi.org/10.1145/3310165.3310170>
 - [5] 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 the Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication*. 782–797.
 - [6] Joshua L. Benjamin, Thomas Gerard, Domanic Lavery, Polina Bayvel, and Georgios Zervas. 2020. PULSE: Optical Circuit Switched Data Center Architecture Operating at Nanosecond Timescales. *Journal of Lightwave Technology* 38, 18 (2020), 4906–4921. <https://doi.org/10.1109/JLT.2020.2997664>
 - [7] Bittware 2024. Bittware 520N-MX FPGA NIC. <https://www.bittware.com/>. (2024).
 - [8] Shaileshh Bojja Venkatakrishnan, Mohammad Alizadeh, and Pramod Viswanath. 2016. Costly Circuits, Submodular Schedules and Approximate Carathéodory Theorems. In *Proceedings of the 2016 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Science (SIGMETRICS '16)*. 75–88.
 - [9] Cheng-Shang Chang, Duan-Shin Lee, and Yi-Shean Jou. 2002. Load balanced Birkhoff-von Neumann switches, part I: one-stage buffering. *Computer Communications* 25, 6 (2002), 611–622. [https://doi.org/10.1016/S0140-3664\(01\)00427-3](https://doi.org/10.1016/S0140-3664(01)00427-3)
 - [10] Amit Chattopadhyay. 2012. Hard Disk Drive Reliability Overview. https://indico.cern.ch/event/247864/contributions/1570376/attachments/426734/592321/HEPIX_October_2013_ver_6.pdf. (2012).
 - [11] Kari A. Clark, Daniel Cletheroe, Thomas Gerard, Istvan Haller, Krzysztof Jozwik, Kai Shi, Benn Thomsen, Hugh Williams, Georgios Zervas, Hitesh Ballani, Polina Bayvel, Paolo Costa, and Zhixin Liu. 2020. Synchronous subnanosecond clock and data recovery for optically switched data centres using clock phase caching. *Nature Electronics* 3, 7 (JUL 2020), 426–433. <https://doi.org/10.1038/s41928-020-0423-y>
 - [12] Gertjan Coudyzer, Peter Ossieur, Johan Bauwelinck, and Xin Yin. 2020. A 25Gbaud PAM-4 Linear Burst-Mode Receiver With Analog Gain- and Offset Control in 0.25 μm m SiGe-C BiCMOS. *IEEE Journal of Solid-State Circuits* 55, 8 (AUG 2020), 2206–2218. <https://doi.org/10.1109/JSSC.2020.2987680>
 - [13] Gertjan Coudyzer, Michiel Verplaetse, Borre Van Lombergen, Robert Borkowski, Thibaut Gurne, Michael Straub, Yannick Lefevre, Peter Ossieur, René Bonk, Werner Coomans, Jochen Maes, and Xin Yin. 2023. 100 Gbit/s PAM-4 Linear Burst-Mode Transimpedance Amplifier for Upstream Flexible Passive Optical Networks. *Journal of Lightwave Technology* 41, 12 (2023), 3652–3659. <https://doi.org/10.1109/JLT.2023.3262319>
 - [14] Nathan Farrington, Alex Forencich, George Porter, P.-C. Sun, Joseph E. Ford, Yeshaiah Fainman, George C. Papen, and Amin Vahdat. 2013. A Multiport Microsecond Optical Circuit Switch for Data Center Networking. *IEEE Photonics Technology Letters* 25, 16 (2013), 1589–1592. <https://doi.org/10.1109/LPT.2013.2270462>
 - [15] Nathan Farrington, George Porter, Sivasankar Radhakrishnan, Hamid Hajabdolali Bazzaz, Vikram Subramanya, Yeshaiah Fainman, George Papen, and Amin Vahdat. 2010. Helios: a hybrid electrical/optical switch architecture for modular data centers. In *Proceedings of the ACM SIGCOMM 2010 Conference*. 339–350.
 - [16] J.E. Ford, V.A. Aksyuk, D.J. Bishop, and J.A. Walker. 1999. Wavelength add-drop switching using tilting micromirrors. *Journal of Lightwave Technology* 17, 5 (1999), 904–911. <https://doi.org/10.1109/50.762910>
 - [17] Alex Forencich, Valerija Kamchevska, Nicolas Dupuis, Benjamin G. Lee, Christian W. Baks, George Papen, and Laurent Schares. 2020. A Dynamically-Reconfigurable Burst-Mode Link Using a Nanosecond Photonic Switch. *Journal of Lightwave Technology* 38, 6 (2020), 1330–1340. <https://doi.org/10.1109/JLT.2020.2970458>
 - [18] Alex Forencich, Alex C Snoeren, George Porter, and George Papen. 2020. Corundum: An open-source 100-Gbps NIC. In *2020 IEEE 28th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*. IEEE, 38–46.
 - [19] Mingye Fu, Guangyao Liu, Roberto Proietti, Yichi Zhang, and S. J. Ben Yoo. 2021. First Demonstration of Monolithic Silicon Photonic Integrated Circuit 32×32 Thin-CLOS AWGR for All-to-All Interconnections. In *2021 European Conference on Optical Communication (ECOC)*. 1–4. <https://doi.org/10.1109/ECOC52684.2021.9605973>
 - [20] Monia Ghobadi, Ratul Mahajan, Amar Phanishayee, Nikhil Devanur, Janardhan Kulkarni, Gireeja Ranade, Pierre-Alexandre Blanche, Houman Rastegarfar, Madeleine Glick, and Daniel Kilper. 2016. Projector: Agile reconfigurable data center interconnect. In *Proceedings of the 2016 ACM SIGCOMM Conference*. 216–229.
 - [21] Sangyoon Han, Tae Joon Seok, Kyoungsik Yu, Niels Quack, Richard S. Muller, and Ming C. Wu. 2018. Large-Scale Polarization-Insensitive Silicon Photonic MEMS Switches. *Journal of Lightwave Technology* 36, 10 (MAY 15 2018), 1824–1830.
 - [22] 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 the Conference of the ACM Special Interest Group on Data Communication (SIGCOMM '17)*. Association for Computing Machinery, New York, NY, USA, 29–42.
 - [23] Jürgen Jahns and Miles J. Murdocca. 1988. Crossover networks and their optical implementation. *Appl. Opt.* 27, 15 (Aug 1988), 3155–3160. <https://doi.org/10.1364/AO.27.003155>
 - [24] Masahiko Jinno, Hidehiko Takara, Bartłomiej Kozicki, Yukio Tsukishima, Yoshiaki Sone, and Shinji Matsuoka. 2009. Spectrum-efficient and scalable elastic optical path network: architecture, benefits, and enabling technologies. *IEEE communications magazine* 47, 11 (2009), 66–73.
 - [25] Norm Jouppi, George Kurian, Sheng Li, Peter Ma, Rahul Nagarajan, Lifeng Nai, Nishant Patil, Suvinay Subramanian, Andy Swing, Brian Towles, et al. 2023. Tpu v4: An optically reconfigurable supercomputer for machine learning with hardware support for embeddings. In *Proceedings of the 50th Annual International Symposium on Computer Architecture*. 1–14.
 - [26] S. A. Jyothi, A. Singla, P. B. Godfrey, and A. Kolla. 2016. Measuring and Understanding Throughput of Network Topologies. In *SC '16: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. 761–772.
 - [27] Simon Kassing, Asaf Valadarsky, Gal Shahaf, Michael Schapira, and Ankit Singla. 2017. Beyond fat-trees without antennae, mirrors, and disco-balls. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication (SIGCOMM '17)*. Association for Computing Machinery, 281–294.
 - [28] Jason Kelley, Alex Forencich, George C. Papen, and William M. Mellette. 2022. Characterization of Burst-Mode Links for Optical Circuit Switching. *Journal of Lightwave Technology* 40, 9 (2022), 2823–2829. <https://doi.org/10.1109/JLT.2022.3148278>
 - [29] He Liu, Feng Lu, Alex Forencich, Rishi Kapoor, Malveeka Tewari, Geoffrey M. Voelker, George Papen, Alex C. Snoeren, and George Porter. 2014. Circuit Switching Under the Radar with REACToR. In *11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 14)*. USENIX Association, Seattle, WA, 1–15. https://www.usenix.org/conference/nsdi14/technical-sessions/presentation/liu_he
 - [30] He Liu, Matthew K. Mukerjee, Conglong Li, Nicolas Feltman, George Papen, Stefan Savage, Srinivasan Seshan, Geoffrey M. Voelker, David G. Andersen, Michael Kaminsky, George Porter, and Alex C. Snoeren. 2015. Scheduling techniques for hybrid circuit/packet networks (CoNEXT '15). Association for Computing Machinery, New York, NY, USA, 13. <https://doi.org/10.1145/2716281.2836126>
 - [31] Hong Liu, Ryohei Urata, Kevin Yasumura, Xiang Zhou, Roy Bannon, Jill Berger, Pedram Dashti, Norm Jouppi, Cedric Lam, Sheng Li, Erji Mao, Daniel Nelson, George Papen, Mukarram Tariq, and Amin Vahdat. 2023. Lightwave Fabrics: At-Scale Optical Circuit Switching for Datacenter and Machine Learning Systems. In *Proceedings of the ACM SIGCOMM 2023 Conference (ACM SIGCOMM '23)*. Association for Computing Machinery, New York, NY, USA, 499–515. <https://doi.org/10.1145/3603269.3604836>
 - [32] Near Margalit, Chao Xiang, Steven M. Bowers, Alexis Bjorlin, Robert Blum, and John E. Bowers. 2021. Perspective on the future of silicon photonics and electronics. *Applied Physics Letters* 118, 22 (06 2021), 220501. <https://doi.org/10.1063/5.0050117>
 - [33] William M. Mellette, Ilya Agurok, Alex Forencich, Spencer Chang, George Papen, and Joseph E. Ford. 2024. A Scalable, High-Speed Optical Rotor Switch, In Optical Fiber Communication Conference (OFC) 2024. *Optical Fiber Communication Conference (OFC) 2024*, Th1A.5. <https://doi.org/10.1364/OFC.2024.Th1A.5>
 - [34] 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 *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*. 1–18.
 - [35] William Maxwell Mellette and Joseph E. Ford. 2015. Scaling Limits of MEMS Beam-Steering Switches for Data Center Networks. *Journal of Lightwave Technology* 33, 15 (2015), 3308–3318. <https://doi.org/10.1109/JLT.2015.2431231>
 - [36] 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 the Conference of the ACM Special Interest Group on Data Communication*. 267–280.
 - [37] William Maxwell Mellette, Glenn M Schuster, George Porter, George Papen, and Joseph E Ford. 2016. A scalable, partially configurable optical switch for data center networks. *Journal of Lightwave Technology* 35, 2 (2016), 136–144.
 - [38] William M. Mellette, Alex C. Snoeren, and George Porter. 2016. P-FatTree: A multi-channel datacenter network topology. In *Proceedings of the 15th ACM Workshop on Hot Topics in Networks (HotNets '16)*. Association for Computing Machinery, New York, NY, USA, 78–84. <https://doi.org/10.1145/3005745.3005746>

- [39] Matthew Nance Hall, Klaus-Tycho Foerster, Stefan Schmid, and Ramakrishnan Durairajan. 2021. A Survey of Reconfigurable Optical Networks. *Optical Switching and Networking* 41 (2021), 100621. <https://doi.org/10.1016/j.osn.2021.100621>
- [40] Nvidia 2023. NVidia DGX Platform. <https://www.nvidia.com/en-us/data-center/dgx-platform/>. (2023).
- [41] Siddharth Pal, Liangyu Zhao, Jason Fantl, Joud Khoury, Arvind Krishnamurthy, and Prithwish Basu. 2023. Efficient All-to-All Collective Communication Schedules for Direct-Connect Topologies. (2023). [arXiv:cs.DC/2309.13541](https://arxiv.org/abs/2309.13541)
- [42] Polatis 2024. Series 7000 384x384 port switch. <https://www.polatis.com/series-7000-384x384-port-software-controlled-optical-circuit-switch-sdn-enabled.asp>. (2024).
- [43] George Porter, Richard Strong, Nathan Farrington, Alex Forencich, Pang Chen-Sun, Tajana Rosing, Yeshaihu Fainman, George Papen, and Amin Vahdat. 2013. Integrating microsecond circuit switching into the data center. *SIGCOMM Comput. Commun. Rev.* 43, 4 (aug 2013), 447–458. <https://doi.org/10.1145/2534169.2486007>
- [44] Leon Poutievski, Omid Mashayekhi, Joon Ong, Arjun Singh, Mukarram Tariq, Rui Wang, Jianan Zhang, Virginia Beauregard, Patrick Conner, Steve Gribble, et al. 2022. Jupiter evolving: transforming Google's datacenter network via optical circuit switches and software-defined networking. In *Proceedings of the ACM SIGCOMM 2022 Conference*. 66–85.
- [45] Arslan Sajid Raja, Sophie Lange, Maxim Karpov, Kai Shi, Xin Fu, Raphael Behrendt, Daniel Cletheroe, Anton Lukashchuk, Istvan Haller, Fotini Karinou, et al. 2021. Ultrafast optical circuit switching for data centers using integrated soliton microcombs. *Nature communications* 12, 1 (2021), 5867.
- [46] A. Rylyakov, J. E. Proesel, S. Rylov, B. G. Lee, J. F. Bulzacchelli, A. Ardey, B. Parker, M. Beakes, C. W. Baks, C. L. Schow, and M. Meghelli. 2015. A 25 Gb/s Burst-Mode Receiver for Low Latency Photonic Switch Networks. *IEEE Journal of Solid-State Circuits* 50, 12 (Dec 2015), 3120–3132.
- [47] Vishal Shrivastav, Asaf Valadarsky, Hitesh Ballani, Paolo Costa, Ki Suh Lee, Han Wang, Rachit Agarwal, and Hakim Weatherspoon. 2019. Shoal: A network architecture for disaggregated racks. In *16th USENIX Symposium on Networked Systems Design and Implementation (NSDI 19)*. 255–270.
- [48] Silicom 2024. Silicom N5013/N5014 FPGA SmartNIC. <https://www.silicom-usa.com/pr/4g-5g-products/4g-5g-adapters/n5013-n5014-fpga-smartnic-intel-stratix-based/>. (2024).
- [49] C Su, LK Chen, and KW Cheung. 1997. Theory of burst-mode receiver and its applications in optical multiaccess networks. *IEEE Journal of Lightwave Technology* 15, 4 (APR 1997), 590–606. <https://doi.org/10.1109/50.566680>
- [50] Rajeev Thakur, Rolf Rabenseifner, and William Gropp. 2005. Optimization of Collective Communication Operations in MPICH. *Int. J. High Perform. Comput. Appl.* 19, 1 (feb 2005), 49–66. <https://doi.org/10.1177/1094342005051521>
- [51] Toshiba. 2015. Reliability of Enterprise Hard Disk Drive. <https://toshiba.semicon-storage.com/content/dam/toshiba-ss-v3/master/en/storage/technology-center/whitepaper-reliability-of-ehdd-e-201509.pdf>. (2015).
- [52] Ryohei Urata, Hong Liu, Kevin Yasumura, Erji Mao, Jill Berger, Xiang Zhou, Cedric Lam, Roy Bannon, Darren Hutchinson, Daniel Nelson, et al. 2022. Mission Apollo: Landing optical circuit switching at datacenter scale. *arXiv preprint arXiv:2208.10041* (2022).
- [53] L. G. Valiant. 1982. A Scheme for Fast Parallel Communication. *SIAM J. Comput.* 11, 2 (1982), 350–361. <https://doi.org/10.1137/0211027>
- [54] Bhanu Vattikonda, George Porter, Amin Vahdat, and Alex C. Snoeren. [n. d.]. Practical TDMA for Datacenter Ethernet (*EuroSys '12*).
- [55] Guohui Wang, David G. Andersen, Michael Kaminsky, Konstantina Papagiannaki, T.S. Eugene Ng, Michael Kozuch, and Michael Ryan. 2010. c-Through: part-time optics in data centers. In *Proceedings of the ACM SIGCOMM 2010 Conference (SIGCOMM '10)*. Association for Computing Machinery, New York, NY, USA, 327–338. <https://doi.org/10.1145/1851182.1851222>
- [56] Weiyang Wang, Moein Khazraee, Zhizhen Zhong, Manya Ghobadi, Zhihao Jia, Dheevatsa Mudigere, Ying Zhang, and Anthony Kewitsch. 2023. TopoOpt: Co-optimizing Network Topology and Parallelization Strategy for Distributed Training Jobs. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*. 739–767.
- [57] Tegan Wilson, Daniel Amir, Vishal Shrivastav, Hakim Weatherspoon, and Robert Kleinberg. 2023. Extending Optimal Oblivious Reconfigurable Networks to all N. In *2023 Symposium on Algorithmic Principles of Computer Systems (APOCS)*. SIAM, 1–16.
- [58] Liang Wu, William Maxwell Mellette, Glenn M Schuster, and Joseph E Ford. [n. d.]. Fast quasi-static beam steering via conformally-mapped gratings. In *Proc. of SPIE Vol.* 11136. 111360W–1.
- [59] Ming C. Wu, Olav Solgaard, and Joseph E. Ford. 2006. Optical MEMS for Lightwave Communication. *Journal of Lightwave Technology* 24, 12 (2006), 4433–4454. <https://doi.org/10.1109/JLT.2006.886405>
- [60] XDP 2024. XDP - eXpress Data Path. <https://prototype-kernel.readthedocs.io/en/latest/networking/XDP/>. (2024).
- [61] Xuwei Xue and Nicola Calabretta. 2022. Nanosecond optical switching and control system for data center networks. *Nature Communications* 13, 1 (APR 26 2022). <https://doi.org/10.1038/s41467-022-29913-1>
- [62] Shizhen Zhao, Rui Wang, Junlan Zhou, Joon Ong, Jeffrey C Mogul, and Amin Vahdat. 2019. Minimal rewiring: Efficient live expansion for clos data center networks. In *16th USENIX Symposium on Networked Systems Design and Implementation (NSDI 19)*. 221–234.

Appendices

Appendices are supporting material that has not been peer-reviewed.

A ROTOR SWITCH HARDWARE

This appendix provides a description of the fabrication and initial hardware-level characterization of our rotor switch⁴.

Figure A.1 shows a cross-sectional diagram of our current rotor switch architecture. A two-dimensional array of single-mode fibers (see Fig. A.3) serves as both the input and output with the optical I/O signals spatially interleaved. A microlens array collimates the optical signals and telecentric 4f relay optics (lens 1 and lens 2 in Fig. A.1) are used to image the fiber array with demagnification onto a rotating disk shown at the top of the figure. The demagnification of the input fiber array onto the rotating disk decreases the overall size of the signal array on the disk and increases the switching speed at a fixed disk rotation rate ω as is discussed below.

The disk is spatially patterned with conformally mapped blazed diffraction gratings so that as it rotates, the diffraction angle is stable within each annular grating sector [58]. Reconfiguration occurs when the boundary between diffractive sectors on the disk is scanned across the full spatial extent of the image of the signal array on the disk (see Fig. 4).

Our prototype uses a rectangular 8×32 signal array imaged to a $200 \times 800 \mu\text{m}$ area on the grating 20 mm from the center of the disk, which spins at 15,000 RPM. The use of a rectangular signal array with a smaller size Z in the azimuthal direction compared to the size in the radial dimension decreases the time T the disk across the fixed image of the signal array. This time defines the switching time $T = Z/v$ where $v = \omega r$ is the linear velocity of the disk across the image of the signal array at a radial distance r from the center of the disk. Scaling the switch to more ports can be readily supported at the same or faster speeds by:

- (1) Increasing the radial distance to the diffractive sectors by using a larger disk.
- (2) Increasing the angular rotation rate ω .
- (3) Reducing the image size (ie. increasing the demagnification) of the fiber array on the disk.
- (4) Changing the aspect ratio of the signal array to reduce the size in the azimuthal direction compared to the radial direction.

For the parameters used in our prototype, the worst-case reconfiguration delay is $7 \mu\text{s}$ (see Fig. A.7). This is approximately $1,000\times$ faster than the MEMS tilt mirrors used in commercial optical switches.

After the rectangular array of optical signals is diffracted from the rotary switching engine, they pass through a second set of 4f relay optics (lens 2 and lens 3 in Fig. A.1, which is spatially

⁴This material is partially adapted and expanded from [33].

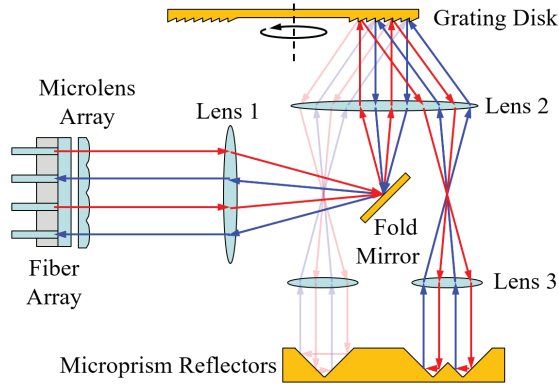


Figure A.1: Cross-sectional diagram of rotor switch architecture with red and blue arrows indicating forward and reverse signal paths.

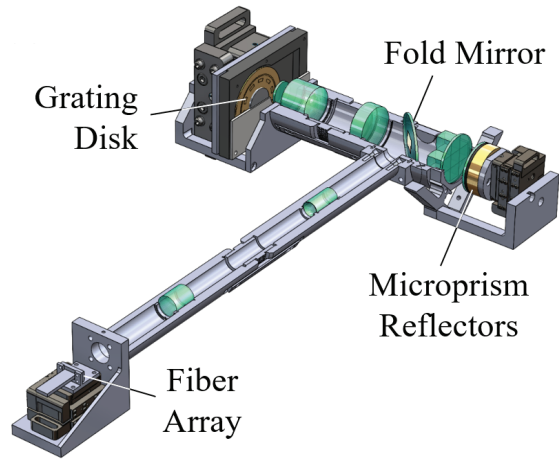


Figure A.2: Optomechanical CAD with lens barrel cutouts to show the optics.

segmented via aperture division multiplexing [37] to image the signal array onto one of four spatially distinct locations.

At each of these locations, a micro-optic structure spatially re-arranges the incoming elements of the signal array to create input/output mappings before reflecting signals to pass back through the switch and into the output fibers. As the disk spins, each of the four mappings is accessed periodically. For the current version of the rotor switch, we used reflective prismatic structures with 90° corners and various pitches to achieve connection patterns corresponding to the set of so-called “crossover” mappings [23]. By permuting the input and/or output fiber cabling between the fiber array and bulkhead fiber optic connectors, the effective connection patterns can be made completely orthogonal between switches (despite each switch having the same internal micro-optic structures), supporting the topology requirements of the overall RotorNet and Opera networks [34, 36].

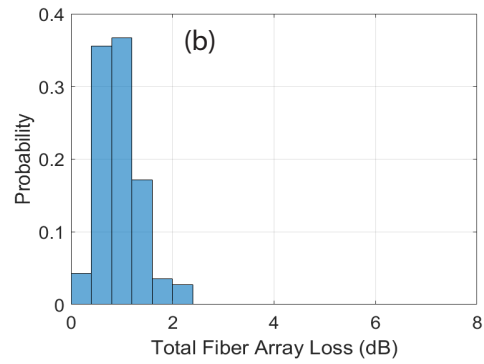
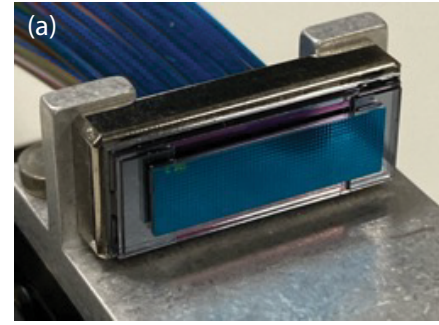


Figure A.3: (a) Photograph of 256-fiber collimator array. (b) Histogram of the measured insertion loss across the 256 fibers in the array.

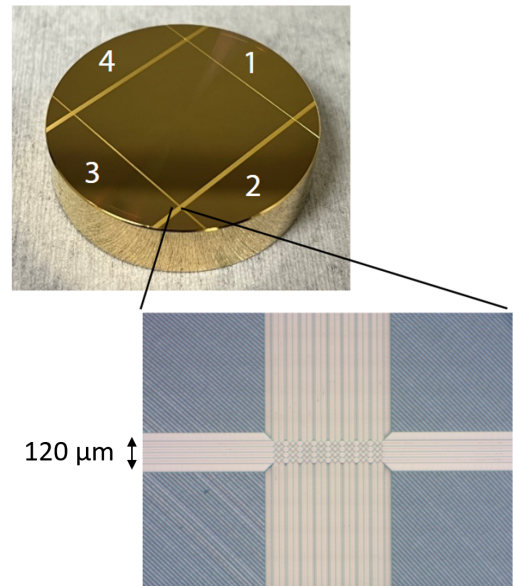


Figure A.4: The four sets of micro-optics used to define the mapping were diamond ruled on a single substrate.

Figure A.2 shows a CAD model of the 128×128 port rotor switch optomechanical assembly with custom lenses and optomechanics. The lens barrel assembly acts as a common mechanical element for integrating all switch subsystems, and we integrated off-the-shelf micro-positioning stages to align the fiber array, grating disk, and microprisms to the barrels.

The fiber array (Fig A.3(a)) was manufactured using an etched silicon plate to position 256 single-mode fibers on a $250 \mu\text{m}$ pitch grid in an 8×32 array. An antireflection (AR) coated glass block was bonded to the array face with index matched epoxy to minimize back reflections. An AR coated silicon microlens array (MLA) fabricated with greyscale lithography was positioned above the fiber array to collimate the signals, with the Gaussian beam waist located at the MLA surface. Including fiber positioning and tilt errors along with MLA radius of curvature (ROC) variability, the worst-case insertion loss of the 256-element collimated fiber array assembly was 2 dB, which was measured by return coupling. The histogram of the fiber array insertion loss is shown in Fig. A.3(b).

The diffractive switching element was fabricated by nano-imprint replication of a greyscale-lithographically produced master onto a 57 mm diameter substrate. The design has 4 diffraction angles corresponding to the four network interconnection patterns supported by the micro-prism arrays. The diffraction efficiency of the 1st order blazed grating was -0.8 dB with 0.3 dB polarization-dependent loss. The 25 mm diameter central hole allows the disk to be mounted to a commercially available 15,000 RPM brushless DC motor hard disk spindle that consumes ≈ 5 W. An additional diffractive track on the disk is used with an optical encoder to monitor the position of the disk for synchronization.

The four microprism reflector arrays shown in Fig. A.4 were fabricated by diamond ruling four orthogonal sets of $70 \mu\text{m}$ deep grooves into a common metal substrate, which was then coated with a layer of gold. The largest feature was a single $140 \mu\text{m}$ wide v-groove used to spatially invert the entire signal array (matching 3 in Fig. A.5). The smallest features were an array of 16 adjacent v-grooves with $32 \mu\text{m}$ pitch which exchanged neighboring signals pairwise within the signal array (matching 1 in Fig. A.5). Fig. A.5 shows top-down and side-profiles of the four matching regions.

Figure A.6 shows the measured fiber-to-fiber transmission and crosstalk spectra for a representative path through the switch.

The center-band insertion loss is 4 dB, and despite the diffractive switching element, the imaging layout of the switch enables a 1-dB bandwidth of 120 nm. Crosstalk between nearest-neighbor fibers in the array is -50 dB, and crosstalk due to the -1 order of the diffraction grating is -35 dB. The polarization-dependent loss (PDL) is 0.7 dB. The loss, -1 order crosstalk, and PDL can all be improved with optimizations to the diffraction grating design and fabrication process. PDL can also be eliminated entirely by introducing a quarter-wave ($\lambda/4$) waveplate before lens 3 to rotate polarization by 90° between the first and second bounce off the diffraction grating [16]. Loss can be further reduced by improving the fiber position, fiber tilt, and microlens array and radius of curvature tolerances in the fiber array assembly.

Figure A.7 shows the reconfiguration speed for two different switching transitions: $7 \mu\text{s}$ when the signals are switched between opposite edges of the signal array (worst-case), and $< 0.5 \mu\text{s}$ when the signals are in the same row in the array (best-case). The switch

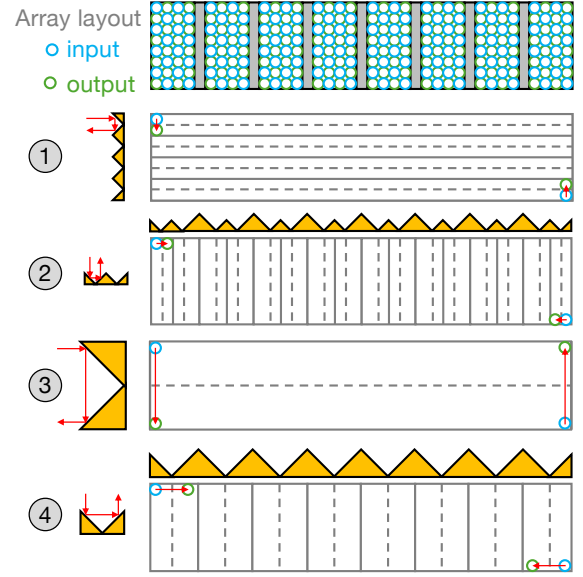


Figure A.5: Schematic of the four sets of micro-prism arrays used to generate the four mappings, showing both top-down and side-profile views.

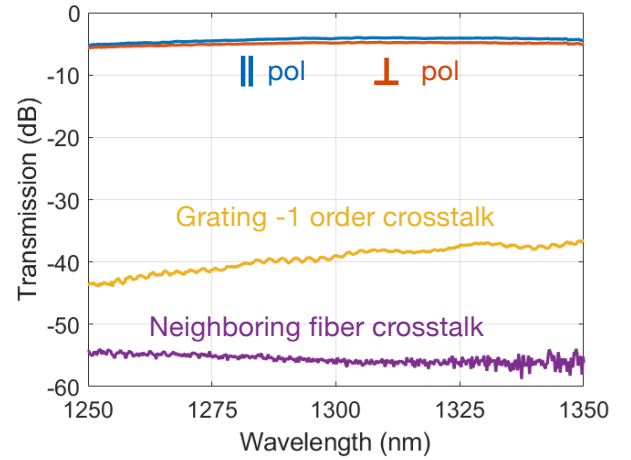


Figure A.6: Spectral transmission for best- and worst-case polarizations (upper set of curves) and crosstalk (lower set of curves) over 100 nm in the 1300 nm optical O-band.

can be phase-locked to an external or internal pulse-per-revolution (PPR) reference synchronization signal. This phase-locking functionality, which is needed for multiple rotor operation, is discussed in § 4.3.

B ADDITIONAL HARDWARE

This appendix describes two additional hardware components developed to meet the needs of RotorNet. The first hardware component is a high-performance, FPGA-based NIC prototyping platform

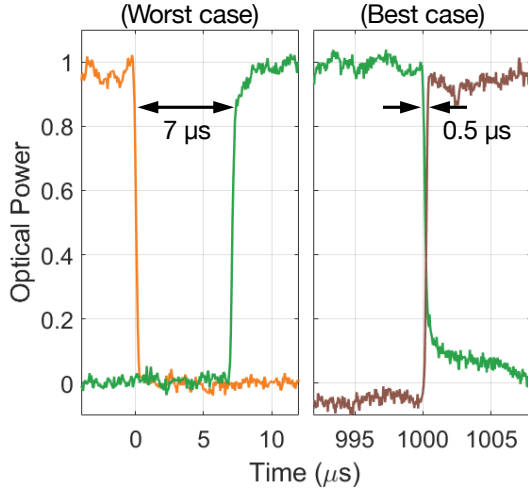


Figure A.7: Worst case and best case rotor switch time.

called Corundum [18]. This appendix describes how Corundum was used to support RotorNet. (Appendix C describes how Corundum was used at the physical layer to measure transceiver characteristics.) The second hardware component is a motor control board that matched the rotation rate and phase of the hard disk spindle to the rate and phase of a precision clock reference running on all of the hosts.

B.1 Corundum FPGA-based NIC

The Corundum FPGA-based NIC prototyping platform was developed to support novel networking architectures and networking protocols at high line rates⁵. The codebase for this platform has been ported to over 25 FPGA development boards and NICs. This section provides a high-level overview of Corundum and its use to support RotorNet.

The Corundum platform has several unique architectural features that were developed with RotorNet in mind. First, hardware queue states are stored efficiently in FPGA block RAM, enabling support for thousands of individually-controllable queues. These queues are associated with interfaces, and each interface can have multiple ports, each with its own independent transmit scheduler. This enables extremely fine-grained control over packet transmission. The scheduler module is designed to be modified to implement different transmit scheduling schemes. This feature was used to enable the high precision TDMA used in RotorNet. The platform also has an application block where custom packet-processing logic and other application-specific logic can be implemented.

A block diagram of the Corundum NIC is shown in Figure B.1. At a high level, the NIC consists of three main nested modules. The top-level module primarily contains support and interfacing components. These components include the PCI express hard IP core and DMA interface, the Precision Time Protocol (PTP) hardware clock, and Ethernet interface components including MACs, PHYs, and associated serializers. The top-level module also includes one or more interface module instances. Each interface module

corresponds to an operating-system-level network interface (e.g. eth0). Each interface module contains the queue management logic, descriptor and completion handling logic, as well as the core transmit and receive datapaths including DMA engines, schedulers, and ingress/egress pipelines. The queue management logic maintains the queue state for all of the NIC queues—transmit, receive, completion, and event queues. Each interface module also contains one or more port module instances, which contain per-port FIFOs and some layer 2 functionality.

For each port, the transmit scheduler decides which queues are designated for transmission. The transmit scheduler generates commands for the transmit engine, which coordinates operations on the transmit datapath. The scheduler module is a flexible functional block that can be modified or replaced to support arbitrary schedules, which may be event driven. The default implementation of the scheduler is simple round robin. All ports associated with the same interface module share the same set of transmit queues and appear as a single, unified interface to the operating system. This enables flows to be migrated between ports or load-balanced across multiple ports by changing only the transmit scheduler settings without affecting the rest of the network stack.

This dynamic, scheduler-defined mapping of queues to ports is a unique feature of Corundum that can enable research into new protocols and network architectures, including parallel networks such as P-FatTree [38] and optically-switched networks such as RotorNet and Opera. In the receive direction, incoming packets pass through a flow hash module to determine the target receive queue and generate commands for the receive engine, which coordinates operations on the receive datapath. Because all ports in the same interface module share the same set of receive queues, incoming flows on different ports are merged together into the same set of queues. It is also possible to add customized modules to the NIC to pre-process and filter incoming packets before they traverse the PCIe bus.

The components on the NIC are interconnected with several different interfaces including AXI lite, AXI stream, and a custom segmented memory interface for DMA operations, which will be discussed later. AXI lite is used for the control path from the driver to the NIC. It is used to initialize and configure the NIC components and to control the queue pointers during transmit and receive operations. AXI stream interfaces are used for transferring packetized data within the NIC, including both PCIe transaction layer packets (TLPs) and Ethernet frames. The segmented memory interface serves to connect the PCIe DMA interface to the NIC datapath and to the descriptor and completion handling logic.

The majority of the NIC logic runs in the PCIe user clock domain, which is nominally 250 MHz for all of the current design variants. Asynchronous FIFOs are used to interface with the MACs, which run in the serializer transmit and receive clock domains as appropriate—156.25 MHz for 10G, 390.625 MHz for 25G, and 322.266 MHz for 100G. A separate PTP clock domain, driven by the Ethernet reference clock, is used for the PTP subsystem to improve the time synchronization performance.

⁵Corundum codebase: <https://github.com/corundum/corundum>

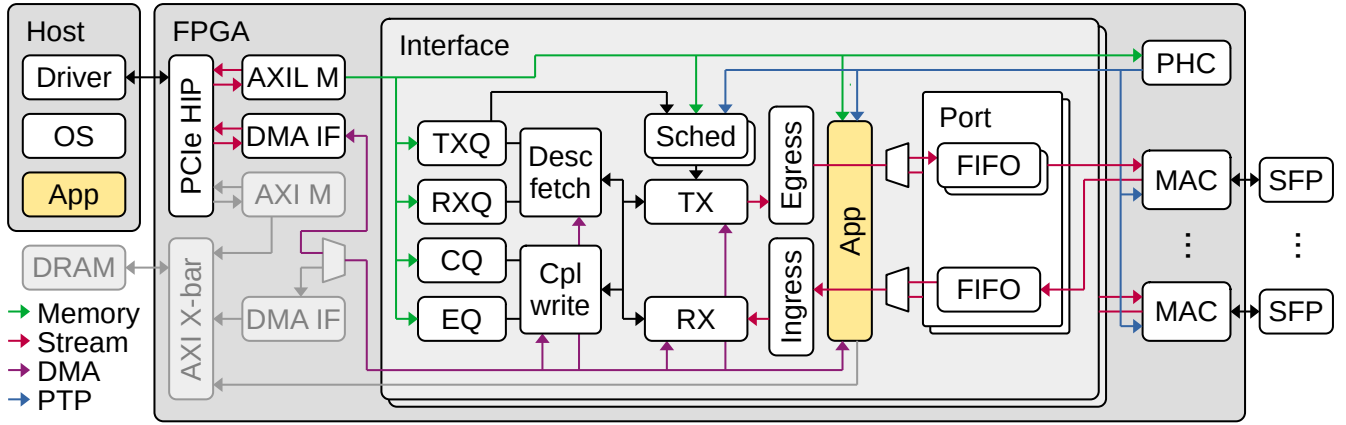


Figure B.1: Block diagram of the Corundum NIC. PCIe HIP: PCIe hard IP core; AXIL M: AXI lite master; DMA IF: DMA interface; PHC: PTP hardware clock; TXQ: transmit queue manager; RXQ: receive queue manager; CQ: completion queue manager; EQ: event queue manager; MAC: Ethernet media access controller (MAC) and physical interface layer (PHY).

Communication of packet data between the Corundum NIC and the driver is mediated via descriptor and completion queues. Descriptor queues form the host-to-NIC communications channel, carrying information about where individual packets are stored in system memory. Completion queues form the NIC-to-host communications channel, carrying information about completed operations and associated metadata. The descriptor and completion queues are implemented as ring buffers that reside in DMA-accessible system memory, while the NIC hardware maintains the necessary queue state information. This state information consists of a pointer to the DMA address of the ring buffer, the size of the ring buffer, the producer and consumer pointers, and a reference to the associated completion queue. The required descriptor state for each queue fits into 128 bits.

The queue management logic for the Corundum NIC must be able to efficiently store and manage the state for thousands of queues so that it can support fine-grained control over outgoing data. This means that the queue state must be completely stored in block RAM (BRAM) or ultra RAM (URAM) on the FPGA. Since a 128 bit RAM is required and URAM blocks are 72x4096, storing the state for 4096 queues requires only 2 URAM instances. Utilizing URAM instances enables scaling the queue management logic to handle at least 32,768 queues per interface.

In order to support high throughput, the NIC must be able to process multiple descriptors in parallel. Therefore, the queue management logic must track multiple in-progress operations, reporting updated queue pointers to the driver as the operations are completed. The state required to track in-progress operations is much smaller than the descriptor state, and as such it can be stored in flip-flops and distributed RAM.

The NIC design uses two queue manager modules: `queue_manager` is used to manage host-to-NIC descriptor queues, while `cpl_queue_manager` is used to manage NIC-to-host completion queues. The modules are similar except for a few minor differences in terms of pointer handling, fill handling, and doorbell/event generation. Because of the similarities, we will discuss only the operation of the `queue_manager` module.

The BRAM or URAM array used to store the queue state information requires several cycles of latency for each read operation, so the `queue_manager` is built with a pipelined architecture to facilitate multiple concurrent operations. The pipeline supports four different operations: register read, register write, dequeue/enqueue request, and dequeue/enqueue commit. Register-access operations over an AXI lite interface enable the driver to initialize the queue state and provide pointers to the allocated host memory as well as access the producer and consumer pointers during normal operation.

B.2 Corundum for RotorNet

For RotorNet, several pieces of functionality are required on the NICs, at multiple different layers in the network stack. Some of this functionality is part of Corundum itself, and some of it sits in Corundum's application block.

B.2.1 Time synchronization. Time synchronization is required to coordinate operations between the hosts and the optical switch. Corundum natively supports PTP time synchronization with a PTP hardware clock and timestamping logic.

In the current test setup, all of the Ethernet links are connected to the switch, so the PTP hardware clocks on the NICs are synchronized from the host via PCIe with `phc2sys`.

B.2.2 Admission control. Admission control is required to manage the flow of data into the network in an efficient manner in coordination with the host network stack. The Corundum schedulers are specifically designed to support TDMA applications. Corundum supports a large number of hardware transmit queues so that outgoing traffic can be classified on a fine-grained basis in software, then precisely controlled by the hardware transmit schedulers. The schedulers can be controlled based on PTP time, such that only the queues that are eligible to transmit will be enabled in a given timeslot.

B.2.3 Routing logic. Dedicated hardware is required to implement routing and store-and-forward for the Opera and RotorLB

protocols. Custom routing logic in the Corundum application section was implemented to perform the time-aware Opera forwarding. The deep buffering and store-and-forward implementation for RotorLB is left for future work, requiring FPGA boards with sufficient memory capabilities.

For Opera, this logic involves classifying incoming data on each port to extract previously-indirected Opera traffic. This traffic is aggregated and checked again to separate out traffic destined for the local host. Traffic that's destined for other hosts is then merged with new outgoing opera traffic and sent to the time-based routing logic. The time-based routing logic has a lookup table that maps both the destination host and current timeslot (which is derived from PTP time) to a destination port. For low latency, the Opera datapath is constructed in parallel with the main datapath, connecting as close to the MACs as possible to avoid the latency of the main TX and RX FIFOs.

For RotorLB, previously-indirected traffic destined for a different host has to be temporarily stored in relatively large DRAM buffers for transmission in a subsequent timeslot. This requires a significant amount of memory bandwidth, either multiple DDR4 channels on the PCB or on-package HBM. The boards used in the testbed do not have sufficient logic resources to implement the necessary memory controllers and other logic to manage the buffering.

B.2.4 TDMA guard enforcement. In an optically-switched network, the precise timing of packet transmissions through the optical switch is paramount, as packets sent at the wrong time will be lost. With the rotor switch, there are also some point defects within the normal slots that also need to be avoided. To provide this functionality, custom logic drives a pause signal to the transmit datapath right where the data is handed off to the per-port MACs for transmission. These signals are driven by a small RAM that is indexed by PTP time.

This hardware actually provides several different capabilities at the same time. First, with the appropriate guard delays, it prevents packets from being sent when a point defect is present on the rotor disk. This prevents the defect from causing any packet loss, with minimal impact on throughput. Second, it prevents packets from being sent while the switch is reconfiguring. There is a fair amount of delay and variance between the scheduler starting to issue transmit requests and those packets actually going out on the wire, necessitating rather large guard delays on both ends of the timeslot. But, with the guard enforcement enabled, the scheduler can safely start issuing transmit requests significantly ahead of the link being ready, relying on the guard enforcement to hold the data until the beginning of the timeslot. This results in better timeslot utilization by eliminating ramp-up issues at the start of the timeslot.

B.2.5 TDMA BER measurement. A small amount of dedicated PRBS test logic connected to the PHYs provides a great deal of visibility into the overall performance of all of the optical links with the switch in operation. On the transmit side, the PRBS test logic uses a linear-feedback shift register to generate PRBS-31 test data that is sent over the link. On the receive side, self-synchronizing LFSR error detectors check the received data for errors. The error counts can then be accumulated in a RAM, with the index computed based on PTP time. This setup can accumulate the errors in time bins, where each bin represents a physical region of the switch

disk. This setup can be used to check the optical performance of the switch and all of the links without having to physically reconfigure any optical connections. Additionally, this setup can verify that the switch and all of the hosts are properly time-synchronized and all of the transceivers are properly configured.

B.3 Control board

The control board used to manage the rotor switch and implement the spindle motor control loop is shown in Fig. B.2. This board is installed alongside the switch, as shown in Fig. 3. The main element of the control board is a Zynq UltraScale+ System on Module (SoM), a Trenz TE0803 with an XCZU4CG FPGA. This module runs Linux and has programmable hardware and additional I/O. The control board interfaces to the rest of RotorNet via ssh using a standard 1-Gb/s Ethernet interface (GbE) and connector (RJ-45). The SoM is synchronized to the rest of the RotorNet using PTP running on the Linux stack. A Trinamic TMC4671 motor controller on an eval board is used to interface with the brushless DC motor that drives the rotor switch disk.

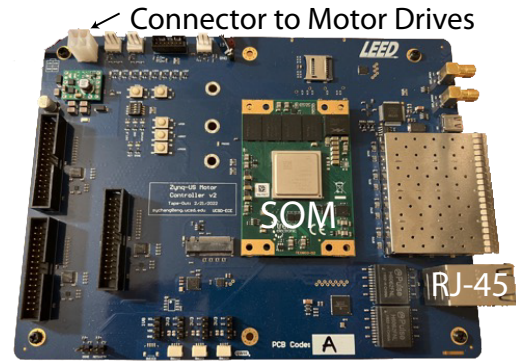


Figure B.2: The control board used to manage the rotor switch and synchronize the rotor switch to the hosts.

C THE PHYSICAL LAYER

This appendix provides background material on the physical layer, how we determined which commercial transceiver to use in RotorNet, and how different physical-layer signaling protocols affect the locking characteristics.

C.1 Background on the physical layer

In an optically-switched network, the system-level reconfiguration time is a vital metric that determines the overall system performance. The system-level reconfiguration time represents the amount of time it takes to reconfigure the network from the standpoint of the dataplane. This time includes every aspect of the physical-layer reconfiguration operation, including not only the time to reconfigure the switch itself, but also the time required for the receiver Physical Medium Attachment (PMA) and the Physical Coding Sublayer (PCS) to lock on to and decode the data and any additional guard delays as necessary to account for uncertainties in synchronization. In order to effectively utilize a fast optical switch, the total delay to implement all of these functions must be reduced

to the point where the overall system-level reconfiguration time is sufficiently low enough to achieve the required cycle time and duty cycle. A fast optical switch is wasted if it takes too long to bring up the physical-layer links every time the switch is reconfigured.

The first step in the sequence is to drain the link before the switch is reconfigured. The transmitter must stop sending data with a guard delay sufficiently large to ensure that the packet payload data is not affected by the switch reconfiguration. This must take into consideration uncertainty from the system time synchronization, uncertainty in terms of the internal reaction time between when the flow of data is paused and transmission of the last frame ends, and other PCS-level aspects such as the alignment of forward error correction (FEC) blocks relative to the end of the last packet.

After the link is drained and is only carrying control information, then switch can be safely reconfigured without losing any data. Depending on the design of the switch, there can be a variation in the actual switch reconfiguration time, with the start and duration potentially varying based on the network configuration. If this variation is predictable, then the guard delays can potentially be adjusted dynamically, otherwise the guard delays must assume the worst case variation in the optical switch reconfiguration time to prevent the loss of data.

Once the switch has been reconfigured, the receiver must recover the data. Depending on the specifics of the physical-layer protocol, which are discussed in § C.3, this can involve a number of different steps and can require a significant amount of time. Many of these steps can be accelerated, especially if the protocol is designed with fast locking in mind.

The analog portion or “front end” of the receiver is the first component to handle the data. In general, the optical power can vary, so an automatic gain control (AGC) loop must first track the power level to select the correct gain and threshold for data detection. Next, equalization is often used to compensate for intersymbol interference due to dispersion, high frequency rolloff, and other effects. The duration of the initialization of the equalizer must be included in the overall locking time budget.

Clock/data recovery (CDR) is also performed to extract the clock and sample the data at the appropriate time. The CDR phase-locked loop (PLL) also requires some time to lock onto the correct clock phase and frequency. Both the CDR and equalizer settings can potentially be cached based on the switch configuration to provide a better starting point for adaptation, speeding up the overall process. Additionally, fast locking algorithms can be used to identify the clock phase more quickly. Burst-mode CDRs tend to use fast-locking routines for both AGC and CDR, in combination with assistance from the transmitter in the form of a preamble that facilitates fast locking [17].

Once the raw data is recovered by the CDR, the last step is to decode the protocol in the physical coding sublayer (PCS). Depending on the specifics of the line code, this can include frame synchronization, scrambler synchronization, FEC block lock, alignment marker lock, and inter-lane deskew. Many of these steps involve various implementation trade-offs that can trade protocol design/overhead, logic complexity, power consumption, and locking time. Depending on the specific physical-layer protocol, these functionalities may reside completely inside the transceiver module or they may be implemented in another part of the physical layer.

For our implementation, we use commercial off-the-shelf (COTS) optical transceivers and standard data communication protocols. Moving to higher line rates is possible, but would require significant additional customization at both the transceiver and protocol level. The rates of 10 Gb/s and 25 Gb/s are somewhat of a sweet spot in terms of simple physical-layer protocols that lock quickly and provide reasonable levels of performance. Faster lane rates, multiple lanes, and PAM-4 add significant additional complexity. For example, at higher line rates, optical transceivers tend to contain retimer chips or even more complicated physical-layer chips than can correct for other physical-layer impairments [52]. Even for the transceivers that we used, the receiver CDRs were problematic and had to be disabled for 25 Gb/s operation through the rotor switch.

C.2 Transceiver characterization for RotorNet

In an optically circuit-switched network, the physical-layer connections must be quickly re-established after every optical switching event. The end-to-end lightwave connection is changed by breaking the optical connection before the switching event and re-establishing a new optical connection after the optical switch reconfiguration delay. This process leads to the receiver “seeing” a new data stream from a different source that can have a different clock phase, frequency, and power level compared to the data stream from the source used before the optical switching event.

This process is distinctly different compared to the unswitched point-to-point links used in static topology packet-switched networks. For that case, the receiver initially locks onto a data stream and maintains that link over the lifetime of the transceiver. Once a link is established, it is kept “alive” irrespective of any data transmission by sending idle control characters over the link so that a transceiver never sees a break in the data stream.

For transceivers used for electrical packet-switched networks, the speed of the overall CDR process, which includes all of the steps discussed in § C.1, is not typically a concern because the overall CDR process is done during the optical link initialization and, ideally, is not done again until the module is replaced. However, the speed of the overall CDR process is a key parameter for optically switched networks and in this context, the CDR acquisition or “locking” process is called burst-mode operation.

The current optical transceiver ecosystem was designed for static topology, electrically packet-switched networks. Therefore, burst-mode functionality is not specified in existing datacom standards used for data center optical interconnects.

Burst-mode operation is a common method for passive optical networks (PONs) and is specified in an IEEE standard [1] for both 25 Gb/s and 50 Gb/s transmission rates. There is also a substantial research literature on burst-mode optical receivers that could be adapted for use in data center transceivers [12, 13, 46, 49] as well as work on characterizing system-level performance of burst-mode operation [17]. Other proposed methods are based on caching the clock phase from a previous connection [11] so that the initial clock phase estimate for locking is more accurate.

In the absence of standards, we developed our own characterization methods [28] for commercial datacom transceivers to determine their viability in RotorNet. For a given module, we determined that the locking time depends on the nominal channel attenuation,

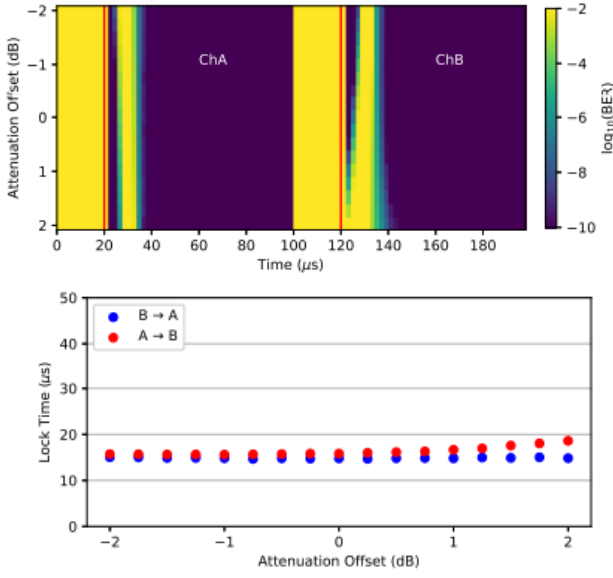


Figure C.1: Top: Sample heat map at 10 Gb/s for an emulated switch time of 20 μ s. The $B \rightarrow A$ transition starts at 20 μ s and the $A \rightarrow B$ transition starts at 120 μ s. (The blue band after the switching event is an artifact of the BER algorithm.) Bottom: Lock times as a function of the attenuation offset during a switching event.

the power offset ratio between the two channels before and after a switching event, the reconfiguration time of the optical switch, and the relative clock phase and clock frequency offset between the receiver and the incoming data stream.

The lock time characteristics were measured using a custom time-resolved bit error rate (BER) measurement developed on the same Corundum platform that was used to develop the NIC functionalities described in Appendix B. The characterization method is an extension of a BER method developed using gated error detectors for high-speed switched links [17].

Bit errors can be temporally resolved using a method analogous to a sampling oscilloscope. The errors are accumulated in temporal bins based on the relative time with respect to an accurate trigger event. In this method, errors in received bits are counted and grouped in temporal bins.

The transient locking characteristics were measured by tracking the BER of a single receiver using a temporal resolution of 1 μ s. The link is subjected to varying conditions before, during, and after an optical switching event.

The measurement time was configured such that each bin contains the errors seen over 3.9×10^9 transmitted bits. The processed data is displayed in the form of a “heat map”. Figure C.1 shows a sample heat map for the commercial optical module used in RotorNet using an emulated circuit switch time of 20 μ s and nominal channel attenuation of 9 dB. The heatmap at a data rate of 25 Gb/s is shown in Fig. C.2.

The y axis on the heat maps in Figs. C.1 and C.2 is the power-offset ratio $10 \log_{10}(P_A/P_B)$ between the two channels that are

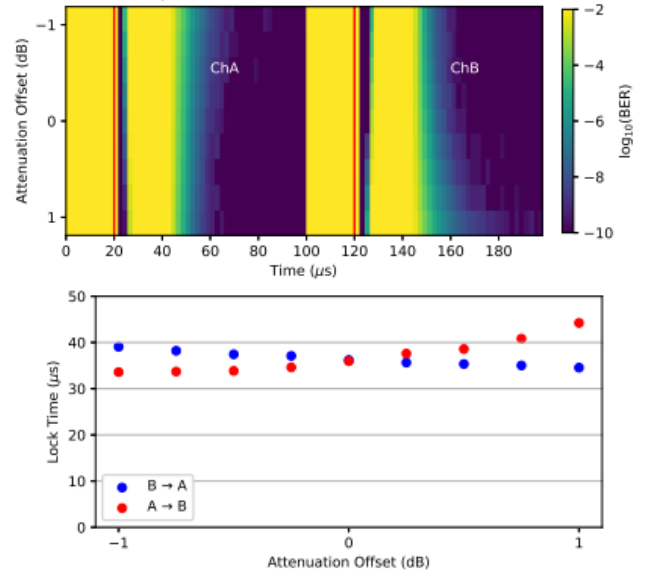


Figure C.2: Top: Sample heat map at 25 Gb/s for an emulated optical switch time of 20 μ s. The $B \rightarrow A$ transition starts at 20 μ s and the $A \rightarrow B$ transition starts at 120 μ s. (The blue band after the switching event is an artifact of the BER measurement algorithm.) Bottom: Lock times as a function of the attenuation offset during a switching event.

switched and the x axis is time in microseconds. The color represents the measured BER in each temporal bin. Bins that have a measured low bit error rate are displayed in blue and bins with a high bit error rate are displayed in yellow. For example, Fig. C.1 shows a relatively constant locking time of less than 20 μ s irrespective of the attenuation offset for both the $A \rightarrow B$ and the $B \rightarrow A$ transition. The locking time when the same module was operated at 25 Gb/s increased to 30–40 μ s depending on the conditions.

Using our custom characterization method, we identified the best-performing commercial module with respect to the locking time as a function of the nominal attenuation and the attenuation offset between channels. When used in RotorNet, these modules produced error free performance for all 512 switch connections when operated at 10 Gb/s. However, when operating at 25 Gb/s about 1/4 of the connections did not perform well because, for similar performance at this higher data rate, the received optical power needs to be approximately 2.5 $\times$ greater than the power required at 10 Gb/s.

This current limitation on the data rate is strictly a consequence of existing physical layer protocols used in data centers not accommodating the optical switch time and not supporting burst-mode operation. It is not a fundamental limitation of our architecture. Burst-mode standards exist in other contexts as has been previously discussed and custom transceivers have been developed to overcome optical switch loss [52].

C.3 Physical-layer protocols

This section describes how the design of the physical-layer protocol can significantly impact the time it takes to bring up an optical link. The design of the physical-layer protocol can include functions such as the specification of the line code, frame synchronization, scrambler synchronization, FEC block lock, alignment marker lock, and inter-lane deskew. These functions are implemented in the physical coding sublayer (PCS) in the PHY.

The purpose of the line code is to map the packet data into symbols in a way that provides a number of useful characteristics. The line code generally needs to provide certain electrical characteristics, including limiting the run length and providing DC balance. Additionally, the line code is usually responsible for transferring control information that is distinct from packet data, which can be used to delineate the start and end of packets as well as indicate errors. The line code must perform all of these functions without adding significant overhead to the link. Forward error correction (FEC) is also common, especially for higher data rates and with higher-order modulation formats like PAM, QPSK, and QAM where the uncorrected BER of the link can be relatively high.

The line code can also contain provisions for inter-lane deskew, where multiple parallel serial lanes are bonded together to form a single link. These links can be carried over separate wires, separate fibers, or separate wavelengths on the same fiber, undergoing slightly different delays which need to be compensated at the receiver.

For 10GBASE-R/25GBASE-R Ethernet without FEC, the PCS mainly needs to achieve frame sync and scrambler sync. The BASE-R flavors of Ethernet use the 64b/66b line code, which packs data into 66 bit blocks that contain 64 data bits and a 2 bit sync header. The serialization delay for one block is 6.4 ns for 10G and 2.56 ns for 25G. The data portion is scrambled with a self-synchronizing linear-feedback shift register (LFSR) to provide DC balance. Since the scrambler is self-synchronizing, it only requires one 66-bit block to synchronize. The time to achieve frame sync is random and implementation-dependent.

The simplest method is to bit slip the deserializer until the sync header is consistently valid. The sync header is 2 bits wide and only 01 and 10 are valid sync headers. If it's not synchronized, then the two bits will be random, so there is a 50% chance that it incorrectly looks like a valid sync header so the PCS has to look at the next block. Eventually an invalid sync header will be seen, causing the PCS to request a bit slip. It's common for the bit slip request to have some latency associated so there will usually be a delay of a few blocks after every bit slip. In the worst case, 65 bit slips will be required. Assuming a bit slip latency of 8 clock cycles and that an average of two blocks have to be observed at each position, $(8+1)*65 = 585$ blocks need to be observed, which requires $585*6.4 = 3744$ ns at 10G and $585*2.56 = 1498$ ns at 25G. This is already fast enough for effective operation with microsecond-scale optical switches like the rotor switch, but there are ways to improve frame synchronization. One potential alternative method is to check all pairs of adjacent bits in parallel and then barrel-shift the data in one shot instead of bit slipping. This requires some additional logic, but has the potential to achieve frame sync by observing only a handful of blocks.

For 100GBASE-R Ethernet without FEC, the PCS uses 20 virtual lanes (VLs) internally that are bit-interleaved onto the physical lanes. After de-interleaving, block lock is carried out independently on each VL at the receiver. Since each VL runs at 1/20th the overall data rate, the serialization time of a 64b/66b block is 12.8 ns. Alignment markers are inserted periodically on all of the VLs, and these markers serve to identify the VLs and provide inter-lane deskew. The alignment markers are 66 bits in size, consisting of a control sync header and 64 bits of data that identifies each VL, and are inserted into the data stream after every 16383 blocks on each VL. The alignment markers are inserted simultaneously on all 20 VLs. The deskew process requires receiving two alignment markers on each VL, which requires $16384*2*12.8 = 419$ μ s in the worst case. This is an order of magnitude longer than the reconfiguration time of the rotor switch.

Currently, there is no other way to deskew the lanes without somehow transmitting additional alignment information. One potential method is to send alignment markers more frequently, but this has the downside of reducing the throughput as the alignment markers interrupt the data stream, wasting bandwidth. This method is used in Energy-efficient Ethernet (EEE) when exiting from low-power idle (LPI); modified alignment markers are transmitted on each VL after every 7 blocks 54 times before reverting to the normal spacing. Another potential method could be to send ordered sets that contain VL numbers and sequence numbers instead of idle control characters, which could enable the link to be deskewed much more quickly when idle.

For 100GBASE-R Ethernet with FEC, the 64b/66b symbols are transcoded to 256b/257b and packed into FEC codewords consisting of 528 (KR4) or 544 (KP4) 10-bit symbols, which contain 514 data symbols and either 14 (KR4) or 30 (KP4) RS-FEC parity symbols. Alignment markers are inserted every 4096 codewords, or every 210 μ s. The 10-bit symbols are then striped across the physical lanes. In this case, bit slipping is used to achieve alignment marker lock, and two alignment markers must be seen before the link can be used. For an 80 bit serdes interface (how the UltraScale+ CMAC is configured), the worst case is 79 bit slips, which requires $(79+1)*210 \mu\text{s} = 17$ ms. This is three orders of magnitude longer than the reconfiguration time of the rotor switch. There are potentially a few ways to speed this up. One is to use additional logic to check for multiple alignment marker offsets simultaneously, and then barrel-shifting once a reliable match is found. Another method is to transmit additional information, either sending alignment markers at a higher rate as in LPI exit where they are sent at the start of every other FEC codeword, or by encoding lane and alignment information within ordered sets such that it can be extracted out of the re-packed symbols.

Higher line rates (200G, 400G, 800G, etc.) have a similar construction to 100GBASE-R with FEC, so they should have similar performance in terms of FEC alignment marker lock times (milliseconds). Also, higher speed modules tend to also include protocol conversion, for example 100G DR1 QSFP28 modules tend to convert DR1 to CAUI-4, re-packing the lanes (1x PAM-4 vs. 4x NRZ) and swapping out the FEC (KP4 vs. KR4, or no FEC). Since the DR1 is handled in a PHY chip inside the module itself, it's not possible to make any modifications to an existing module to speed up the synchronization process.

For our implementation, one significant advantage of using 10G Ethernet is that this protocol avoids the long time delays required for inter-lane deskew and FEC that can be an issue for higher-speed protocols. Resolving these issues for higher line rates may

potentially require custom MAC and PCS logic in order to shorten the time required to bring up the links.