Shoal: A Network Architecture for Disaggregated Racks

Vishal Shrivastav, *Cornell University;* Asaf Valadarsky, *Hebrew University of Jerusalem;* Hitesh Ballani and Paolo Costa, *Microsoft Research;* Ki Suh Lee, *Waltz Networks;* Han Wang, *Barefoot Networks;* Rachit Agarwal and Hakim Weatherspoon, *Cornell University*

https://www.usenix.org/conference/nsdi19/presentation/shrivastav

This paper is included in the Proceedings of the 16th USENIX Symposium on Networked Systems Design and Implementation (NSDI '19).

February 26–28, 2019 • Boston, MA, USA

ISBN 978-1-931971-49-2

Open access to the Proceedings of the 16th USENIX Symposium on Networked Systems Design and Implementation (NSDI '19) is sponsored by

Shoal: A Network Architecture for Disaggregated Racks

Vishal Shrivastav
Cornell University

Asaf Valadarsky
Hebrew University of Jerusalem

Hitesh Ballani
Microsoft Research

Paolo Costa
Microsoft Research

Ki Suh Lee
Waltz Networks

Han Wang
Barefoot Networks

Rachit Agarwal
Cornell University

Hakim Weatherspoon
Cornell University

Abstract

Disaggregated racks comprise dense pools of compute, memory and storage blades, all interconnected through an internal network. However, their density poses a unique challenge for the rack's network: it needs to connect an order of magnitude more resource nodes than today's racks without exceeding the rack's fixed power budget and without compromising on performance. We present Shoal, a power-efficient yet performant intra-rack network fabric built using fast circuit switches. Such switches consume less power as they have no buffers and no packet inspection mechanism, yet can be reconfigured in nanoseconds. Rack nodes transmit according to a static schedule such that there is no in-network contention without requiring a centralized controller. Shoal's congestion control leverages the physical fabric to achieve fairness and both bounded worst-case network throughput and queuing. We use an FPGA-based prototype, testbed experiments, and simulations to illustrate Shoal's mechanisms are practical, and can simultaneously achieve high density and high performance: 71% lower power and comparable or higher performance than today's network designs.

1 Introduction

Traditional datacenter use a server-centric architecture in which a number of racks, each comprising tens of servers connected via a top-of-the-rack (ToR) switch, are interconnected by the datacenter network. However, the end of Dennard's scaling [18] and the slowdown of Moore's Law [14] are challenging the long-term sustainability of this architecture [19]. Consequently, a new paradigm has emerged: *rack-scale* architecture, where a server is replaced by a rack as the unit of computation, with each rack hosting a number of System-on-Chip (SoC) [15, 35, 65] microservers, each comprising multi-core CPUs integrated with some local memory, combined with separate pools of non-volatile memory, storage and custom compute (e.g., Google TPUs [82], GPGPUs [74, 78] and FPGAs [43]) blades, all interconnected through an internal network. This enables *resource disaggregation* as compute units are decoupled from memory and storage units. The benefits disaggregation are well understood in the computer architecture community [5, 41]: it enables fine-grained resource pooling and provisioning, lower power consumption and higher density than traditional server-centric architectures, thus enabling each rack to host hundreds of resource "nodes" (compute/memory/storage blades). Several examples of rack-scale architecture have been proposed both in industry (Intel [72], Facebook [65, 74], Microsoft [43], SeaMicro [79], HPE [28], Google [82]) and academia [5, 6, 15, 27, 35, 41].

Increasing rack density, however, poses new challenges for the rack's network. Traditional ToR switches can support only around a hundred ports at high speed. Therefore, interconnecting several hundreds or even a thousand nodes requires either a high-port count chassis switch or a number of low-port count switches arranged in a hierarchical topology, e.g., a folded Clos [1]. Such a design, when coupled with state-of-the-art protocols [2, 4, 20, 25], can provide high throughput and low latency that could potentially meet the requirements of disaggregated workloads [19]. Unfortunately, such packet-switched designs are significantly less power and cost efficient as compared to today's intra-rack networks (§2). Power is a particular concern as the rack's total power has a hard limit due to cooling [35, 60], so network inefficiency ultimately limits the density of other resources.

The limitations of packet-switched networks have already prompted network designs that leverage circuit switches in datacenters [11, 23, 24, 38, 42, 53]. Such switches can be optical or electrical, and the fact that they operate at the physical layer with no buffers, no arbitration and no packet inspection mechanisms means they can be cheaper and more power efficient than an equivalent packet switch (§5). Adopting these designs for intra-rack connectivity would thus alleviate the power concern. However, achieving low latency would still be challenging as traditional circuit switches have reconfiguration delays of the order of few microseconds to even milliseconds. Such a solution, thus, would either compromise on performance or still have to rely on a separate packet-switched network to handle latency-sensitive traffic. In summary, adapting existing network solutions to high-density racks would either compromise on power (packet-switched) or on performance (purely circuit-switched).

In this paper, we show that it is possible to design a rack-

scale network that operates comfortably within the rack's power budget while achieving performance comparable to packet-switched networks. Our work is motivated by fast circuit switches that can be reconfigured in a few to tens of nanoseconds while still being power-efficient. These are available commercially [76] as well as research prototypes [10, 16, 17, 30, 35, 36, 48, 52]. Unfortunately, it is not sufficient to simply take existing circuit-switch-based architectures and upgrade their switches as these architectures were designed under the assumption of slow reconfiguration times. In particular, these solutions rely either on a centralized controller to reconfigure the switches [11, 23, 24, 35, 42, 53], which would be infeasible at a nanosecond scale, or on a scheduler-less design with a large congestion control loop [38], which prevents taking advantage of fast reconfiguration speeds.

We present Shoal, a power-efficient yet performant network fabric for disaggregated racks built using fast circuit switches. Shoal reconfigures the fabric using a static schedule that connects each pair of rack node at an equal rate. This avoids the need for a centralized scheduler that can operate at a sub-microsecond granularity. To accommodate dynamic traffic patterns atop a static schedule, traffic from each node is uniformly distributed across all rack nodes which then forward it to the destination; a form of detour routing. Such *coordination-free scheduling*, first proposed by Chang et al. [9] as an extension of Valiant's method [50], obviates the complexity and latency associated with centralized schedulers while guaranteeing the worst-case network throughput across *any* traffic pattern [9]. Such scheduling, however, requires that all nodes are connected through what looks like a single non-blocking switch. To achieve this, Shoal's fabric uses many low port-count circuit switches connected in a Clos topology. When reconfigured synchronously, the switches operate like a single circuit switch. Further, we decompose the static, equal-rate schedule for the fabric into static schedules for the constituent switches.

Overall, this paper makes the following contributions:

- We present a network architecture for disaggregated racks that couples fast circuit switches with the servers' network stack to achieve low and predictable latency at low cost and power.
- We designed a fabric that uses low port-count circuit switches to offer the abstraction of a rack-wide circuit switch. We also scaled the coordination-free scheduling technique to operate across the fabric.
- We devised an efficient congestion control mechanism to run atop Shoal's fabric. This is particularly challenging to achieve due to high multi-pathing—traffic between a pair of nodes is routed through all rack nodes. Shoal leverages the observation that the static schedule creates a periodic connection between any pair of rack nodes to implement an efficient backpressure-based congestion control, amenable to hardware implementation.
- We implemented Shoal's NIC and circuit switch on an FPGA; our prototype achieves small reconfiguration delay (6.4 ns) for the circuit switches and is a faithful implementation of our entire design including the scheduling and the congestion control mechanisms.
- We incorporated the NIC and the switch prototype into an end-to-end small-scale rack testbed that comprises six FPGA-based circuit switches in a leaf-spine topology connecting eight FPGA-based NICs at end hosts.

Experiments on this small-scale testbed shows that Shoal offers high bandwidth and low latency; yet our analysis indicates that its power can be 71% lower than an equivalent packet-switched network. Using a cross-validated simulator, we show that Shoal's properties hold at scale too. Across datacenter-like workloads, Shoal achieves comparable or higher performance than a packet-switched network using state-of-the-art protocols [2, 25, 54], with improved tail latency (up to 2× lower as compared to NDP [25]). Further, through simulations based on real traces [19], we also demonstrate that Shoal can cater to the demands of emerging disaggregated workloads.

2 Motivation

We first consider how conventional datacenter networks could be adapted for disaggregated racks and the shortcomings of such an approach.

Strawman 1. Chassis switches with hundreds of ports, often used at higher levels of a datacenter's network hierarchy, could connect all rack nodes but at significant cost, power, and space. For example, the Cisco Nexus 7700 switch can support 768 ports at 10 Gbps (only 192 at 100 Gbps). Yet, it consumes 4 KW power and occupies 26 RU [61], which is 26% and 54% of the rack's power and space budget respectively. A rack's total power has a hard limit of around 15 KW due to constraints on power supply density, rack cooling and heat dissipation [35, 60, 66]. We also considered a custom solution involving commodity switches arranged in a Clos topology, which would still consume around 8.72 KW to connect 512 nodes (§ 5). The key reason for this is that packet switching necessitates buffers and arbitration inside each switch and serialization-deserialization at each switch port, which are major contributors (up to 70%) to the switch's chip area and package complexity [34, 62], and in turn, its power.

Strawman 2. Motivated by the observation that enabling high-density racks requires a step change in the power-efficiency of the network, practitioners have attempted to integrate several very low-port (typically four or six ports) packet switches in the system-on-chip (SoC) of the microserver. Thus, instead of building a ToR-based network, the microservers can be connected to each other using direct-connect topologies prevalent in HPC and super-computing

systems, e.g., a 3D torus [41, 68, 79]. This design significantly reduces the overall network power consumption as the additional logic per SoC is small. However, a key drawback of direct-connect networks is that they have a static topology which cannot be reconfigured based on current traffic pattern. Hence their performance is workload dependent—for dynamically changing workloads such as datacenter workloads, it results in routing traffic across several rack nodes, which hurts network throughput and latency (§7.3) and complicates routing and congestion control [15].

Circuit switching. These strawmans lead to the question whether packet-switched networks are well-suited to support high-density racks. On the upside, packet-switched networks offer excellent performance and allow the network core to be loosely coupled with the servers' network stack. In datacenters and WANs, this has been a good trade-off—the increased power of switches is not a key concern yet loose coupling has allowed the core network technologies to evolve independent of the servers' network stack. This also allows the network to be asynchronous, which helps scaling. These benefits, however, do not hold up inside a rack. The physical size of a rack means that achieving rack-wide synchronization is feasible. Further, many density and cost benefits of disaggregated racks come from the co-design of servers and the network, so independent evolution is not critical.

Instead, we argue that a circuit-switched network offers a different set of trade-offs that are more suited to disaggregated racks. Compared to a packet switch, circuit switches can draw less power and be cheaper due to their simplicity, and these gains could grow with future optical switches (§5). Thus, they can better accommodate higher density. On the flip side, circuit switching does necessitate a tight coupling where all nodes are synchronized and traffic is explicitly scheduled. Further, past solutions with slow circuit switches have had to rely on a separate packet-switched network to support low latency workloads which increases complexity and hurts network manageability. Using fast circuit switches helps on the performance front yet makes the scheduling harder. We show that these challenges can be solved at the scale of a rack and it is feasible to build a rack network that satisfies its power constraints while achieving performance on par with a packet-switched network.

3 Design

Shoal is a network architecture for disaggregated racks. It comprises a network stack at the rack nodes which is tightly coupled with a circuit-switched physical fabric.

3.1 Design overview

Shoal's architecture is shown in Fig. 1. Each rack node is equipped with a network interface connecting it to the Shoal fabric. The fabric comprises a hierarchical collection of smaller circuit switches, electrical or optical, that are reconfigured synchronously. Hence, the fabric operates like a single, giant circuit switch (§3.2). The use of a circuit switched fabric means that we need to schedule it. One possible approach is to schedule it *on-demand*, i.e., connect nodes depending on the rack's traffic matrix. However, such on-demand scheduling requires complicated scheduling algorithms and demand estimation, and would make it hard to meet low-latency constraints.

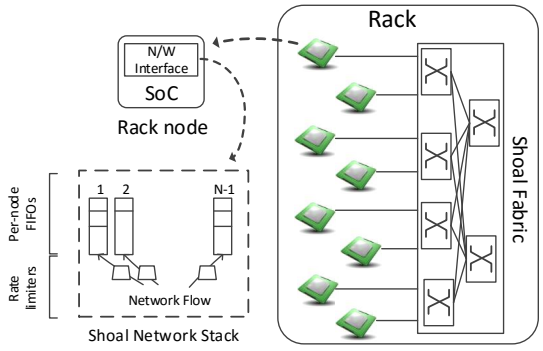


Figure 1: Shoal architecture.

Instead, Shoal uses *coordination-free* scheduling [9]. Specifically, each circuit switch forwards fixed-sized packets or "*cells*" between its ports based on a predefined "schedule". These per-switch schedules, when taken together, yield a schedule for the fabric which dictates when different node pairs are connected to each other. The schedule for individual switches is chosen such that the fabric's schedule provides equal rate connectivity between each pair of nodes. To accommodate any traffic pattern atop the equal rate connectivity offered by the fabric, each node spreads its traffic uniformly across all other rack nodes, which then forward it to the destination (§3.3.1).

The second mechanism implemented in Shoal's network stack is a congestion control technique that ensures that network flows converge to their max-min fair rates, while bounding the maximum queuing at all rack nodes. Our main insight here is that the periodic connection of rack nodes by the fabric enables backpressure-based congestion control amenable to hardware implementation. One of the main challenges in implementing backpressure-based mechanisms over multi-hop networks is instability for dynamic traffic [26]. In Shoal, we restrict the backpressure mechanism to a *single hop*, avoiding the instability issue altogether.

3.2 Shoal fabric

Shoal uses a predefined, static schedule to reconfigure the fabric such that the rack nodes are connected at an equal rate. Fig. 3 shows an example schedule with $N = 8$ nodes. Thus, in a rack with N nodes, each pair of nodes is directly connected by the fabric once every $N-1$ time slots, where a slot refers to the cell transmission time.

However, constructing a monolithic switch, electrical or optical, with hundreds of high-bandwidth ports and fast re-

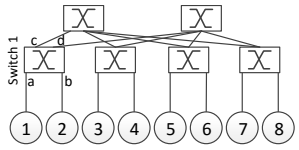


Figure 2: Circuit switches in a two-stage Clos topology.

Time slot

Node	1	2	3	4	5	6	7
1	2	3	4	5	6	7	8
2	3	4	5	6	7	8	1
.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.
8	1	2	3	4	5	6	7

Figure 3: Fabric schedule for a rack with 8 nodes.

Time slot

Port	1	2	3	4	5	6	7
a	d	c	d	c	d	c	d
b	c	d	c	d	c	d	c
c	b	a	b	a	b	a	b
d	a	b	a	b	a	b	a

Figure 4: Switch 1's schedule (see Fig. 2 for topology).

configuration is intractable due to fabrication constraints. Instead, Shoal's fabric comprises low port-count circuit switches connected in a non-blocking Clos topology. Arranging k-port circuit switches in a two-stage Clos topology allows the fabric to connect $\frac{k^2}{2}$ nodes. For e.g., using 64-port electrical circuit switches allows us to connect a rack with 2,048 nodes. Fig. 2 shows six 4-port circuit switches arranged in such a topology to implement an 8-port fabric. Packets between any two nodes are always routed through both stages of the topology, even if the nodes are connected to the same switch (like nodes 1 and 2 in the figure). Since the topology is non-blocking, this does not impact network throughput. It ensures, however, that the distance between any two nodes is the same which, in turn, aids rack-wide time synchronization (§3.4).

We decompose the schedule of the overall fabric into the schedule for each constituent circuit switch. Consider the example fabric shown in Fig. 2. Fig. 3 shows the schedule for this fabric while Fig. 4 shows the schedule for switch 1. Each switch's schedule is contention-free, i.e., at a given instant, any port is connected to only one port. This allows the switch to do away with any buffers and any mechanisms for packet inspection or packet arbitration.

3.3 Shoal network stack

Shoal's mechanisms operate at the data link layer (layer-2) of the network stack. At each node, Shoal spreads its layer-2 traffic uniformly across the rack to ensure guaranteed network throughput and implements a congestion control technique that ensures fair bandwidth sharing and low latency.

3.3.1 Forwarding plane

Rack nodes send and receive fixed-sized cells. Packets received from higher layers are thus fragmented into cells at the source node and reassembled at the destination. Each cell has a header (Fig. 5) that contains the corresponding packet's destination and other control information.

Cells sourced by a node, irrespective of their destination, are sent to the next node the source is connected to. This uniformly spreads traffic across all rack nodes. Each node has a set of FIFO queues, one for every node in the rack. Cells arriving at an intermediate node are put into the queue corresponding to their final destination. This ensures traffic is detoured through at most one intermediate node. These queues are served according to the node's transmission schedule.

We highlight two key aspects of this simple design. First, uniformly distributing traffic is perfectly suited to the equal rate connectivity provided by the Shoal fabric. This guarantees the worst-case throughput across *any* traffic pattern [9]—Shoal's network throughput can be at most $2\times$ worse than that achieved by a hypothetical, rack-wide ideal packet switch. To compensate for this throughput reduction due to detouring, we double the aggregate bisection bandwidth of the fabric for Shoal. This is a good trade-off as circuit switches are expected to be cheaper and hence, adding fabric bandwidth is inexpensive; in §5, the cost of the resulting network is still estimated to be lower than the cost of a traditional packet-switched network.

Second, when the fabric's schedule connects node i to node j, the former *always* transmits a cell; the cell at the head of the queue $i \rightarrow j$ is transmitted, otherwise an empty cell is sent. This ensures that each node periodically receives a cell from every other node, which enables implementing an efficient backpressure-based congestion control (§3.3.2) and simple failure detection (§3.5).

3.3.2 Congestion control

Each node sending traffic computes the appropriate rate for its traffic to avoid congesting the network. We begin with a discussion of the network topology resulting from periodic reconfiguration of the Shoal fabric and its implications for congestion control, followed by the details of our design.

High Multi-pathing. The periodic reconfiguration of Shoal's fabric means that the entire network can be seen as an all-to-all mesh with virtual links between each pair of nodes. For e.g., consider a rack with 8 nodes whose schedule is shown in Fig. 3. Since each node is connected to every node $1/7^{th}$ of the time, the network provides the illusion of a complete mesh with virtual links whose capacity is $1/7^{th}$ of each node's total network bandwidth.

Shoal's use of detouring means that each node's traffic is routed through all the rack nodes on their way to their destination, resulting in very high multi-pathing. In contrast, the TCP suite of protocols, including protocols tailored for datacenters [2, 51] and recent protocols for RDMA networks [39,54] only use a single path. Even multi-path extensions like MPTCP [44] target scenarios with tens of paths, which is an order of magnitude less than the number of paths used by traffic in our fabric.

Design insights. Shoal's congestion control design is based on three key insights. First, we leverage the fact that the fab-

ric in an N-node rack directly connects each pair of nodes once every $N-1$ time slots. We refer to this interval as an *epoch*. This means that, when the queues at an intermediate node grow, it can send a timely backpressure signal to the sender. As we detail below, the periodic nature of this signal coupled with careful design of how a sender reacts to it allows us to bound the queue size across rack nodes.

Second, achieving per-flow fairness with backpressure mechanisms is challenging [54], especially in multi-path scenarios. In Shoal, a *flow* refers to all layer-2 packets being exchanged between a pair of nodes. For network traffic, this includes all transport connections between the nodes. For storage traffic, this includes all IO between them. Each flow comprises $N-1$ *subflows*, one corresponding to each intermediate node. Shoal achieves max-min fairness across flows by leveraging the fact that each flow comprises an equal number of subflows that are routed uniformly across a symmetric network topology, so we can achieve per-flow fairness by ensuring per-subflow fairness. We thus treat each subflow independently and aim to determine their fair sending rates. The mechanism can also be extended to accomodate other flow-level sharing policies.

Finally, each subflow traverses two virtual links, either of which can be the bottleneck. For e.g., a subflow $i \rightarrow j \rightarrow k$ can either be bottlenecked at the virtual link between nodes i and j, or between nodes j and k. Shoal maintains a queue, Q_{ij}, at node i to store cells destined to node j. We use the length of the queue Q_{ij} as an indication of the load on the virtual link between nodes i and j. Note that the node sourcing the traffic, node i, can observe the size of the local queue Q_{ij}. It, however, also needs to obtain information about the size of the remote queue Q_{jk} that resides at node j.

Congestion control mechanism. We use a subflow from source i to destination k through intermediate node j, $i \rightarrow j \rightarrow k$, as a running example to explain Shoal's congestion control. When node i sends a cell to node j, it records the subflow that the cell belongs to. Similarly, when node j receives the cell, it records the index k of the queue that the cell is added to. The next time node j is connected to node i, it embeds the current length of queue Q_{jk} into the cell header:

$$\textit{rate limit feedback}_{ji} = \textit{len}(Q_{jk}) \tag{1}$$

Each pair of nodes in the rack exchange a cell every epoch, even if there is no actual traffic to be sent. Thus, when node i sends a cell to node j, it gets feedback regarding the relevant queue at j within the next epoch. Let us assume that node i receives this feedback at time T. At time t ($\geq T$), it knows the instantaneous length of its local queue to node j, $Q_{ij}(t)$, and a sample of the length of the remote queue between nodes j and k, $Q_{jk}(T)$. The max-min fair sending rate for a subflow is governed by the most bottlenecked link on its path, i.e., the link with the maximum queuing. As a result, the next cell for this subflow should only be sent after both the queues have had time to drain, i.e., at least, $max(len(Q_{ij}(T)), len(Q_{jk}(T)))$ epochs have passed since the feedback was received. To achieve this, node i releases a cell for this subflow into its local queue for j only when the current length of the queue, after accounting for the time since the last feedback, exceeds the size of the remote queue Q_{jk}, i.e., a cell is released into Q_{ij} at time t when,

$$\textit{len}(Q_{ij}(t)) + (t-T) \geq \textit{len}(Q_{jk}(T)) \tag{2}$$

Thus, when a new cell is released into the queue at its source, the previous cell in that queue is guaranteed to have been sent to the remote queue while the previous cell in the remote queue is guaranteed to have been sent to the destination. This ensures the *invariant* that at any given time a subflow has at most one cell each in both the queue at its source and the queue at its intermediate node. As a consequence, at any given time, the size of each queue Q_{ij} is bounded by:

$$\textit{len}(Q_{ij}) \leq \textit{outcast degree}(i) + \textit{incast degree}(j) \tag{3}$$

Thus, this mechanism ensures that, for each virtual link, Shoal performs *fair queuing* at cell granularity across all the subflows sharing that link. This, in turn, results in a tighter distribution of flow completion times.

Note that while Shoal's basic design assumes a single traffic class for the flows, it can be easily extended to support multiple traffic classes as explained in Appendix C.

3.3.3 Improving network latency

While Eq. 3 bounds the queue size, it also highlights one of the challenges of detouring: network latency experienced by a cell, while bounded, is impacted by cross-traffic — traffic from remote nodes at the cell's source node and traffic from local node at the cell's intermediate node. To reduce this impact of detouring, we introduce following optimizations:

Reducing cell latency at the intermediate node. In addition to queue Q_{jk}, node j also maintains a ready queue R_{jk}. Instead of adding cells to Q_{jk} from local flows that satisfy Eq. 2, Shoal adds the corresponding flow ids into the ready queue R_{jk}. Thus,

$$\textit{len}(R_{jk}) \leq \textit{outcast degree}(j) \qquad \leq N-1 \tag{4}$$

Shoal then scans the local flow ids in R_{jk}, and adds the corresponding cells into the queue Q_{jk} such that at any given time there is at most one local cell in Q_{jk}. Thus Eq. 3 changes to:

$$\textit{len}(Q_{jk}) \leq 1 + \textit{incast degree}(k) \qquad \leq N \tag{5}$$

However, to ensure that the rate limit feedback accounts for the local subflows, Eq. 1 needs to be updated accordingly:

$$\textit{rate limit feedback}_{ji} = \textit{len}(Q_{jk}) + \textit{len}(R_{jk}) - 1 \tag{6}$$

The rack network is thus still shared in a max-min fashion, while simultaneously reducing the impact of local traffic on the latency of remote cells — the latency experienced by a remote cell at any intermediate node is determined only by the incast degree of cell's destination.

Reducing cell latency at the source node. While Eq. 5 reduces the impact of detouring at the intermediate node, at

the source node i, the latency for a local cell in Q_{ij} is governed by incast degree of intermediate node j. To reduce the impact of cross traffic (i.e., non-local traffic), Shoal selectively adds cells from a new flow to queue Q_{ij} only if $len(Q_{ij}) \leq 2^{\text{age}}$, where age is measured in epochs since the flow started. Thus, for the first few epochs, cells will be released to queues over virtual links with low contention, and afterwards will quickly converge to uniform load-balancing using all virtual links after a max of $log(N)$ epochs. This achieves uniform load-balancing for long flows, and hence preserves Shoal's throughput bounds, while reducing completion time for short flows.

The impact of these optimizations is evaluated in Fig. 14.

3.3.4 Bounded queuing

Eq. 5 guarantees that at any given time, the size of each queue Q_{ij} at node i is bounded by the instantaneous number of flows destined to destination j plus one, with at most one cell per flow. This queue bound can be used to determine the maximum buffering needed at each node's network interface to accomodate even the worst-case traffic pattern of all-to-one incast. In a rack with 512 nodes and 64 B cells, this requires a total buffering per node of 17 MB. Importantly, since Shoal accesses a queue only once every epoch for transmission, and assuming the access latency of off-chip memory is less than an epoch, Shoal only needs to buffer one cell from each queue Q_{ij} on the on-chip memory, resulting in $N-1$ total cells. Using the example above, this leads to on-chip cell buffer size of just 32 KB per node.

3.4 Shoal slots and guard band

Shoal operates in a time-slotted fashion. Slots are separated by a "guard band" during which the switches are reconfigured. The guard band also accounts for any errors in rack synchronization.

Circuit switch reconfiguration. Shoal uses fast reconfigurable circuit switches. For example, our prototype implements an FPGA-based circuit switch that can be reconfigured in 6.4 ns (§4.1). Electrical circuit switches with fast reconfiguration are also commercially available [76] while fast optical circuit switches with nanosecond-reconfiguration time have also been demonstrated [12, 16, 17, 30, 36, 48, 52].

Time synchronization. Shoal's slotted operation requires that all rack nodes and switches are time synchronized, i.e., they agree on when a slot begins and ends. Synchronizing large networks is hard, primarily because of high propagation delay and the variability in it. In contrast, fine-grained rack-wide synchronization is tractable due to their size—a typical rack is only a few meters high which means that, even when using optical transmission with a propagation delay of 5 ns/m, the maximum propagation latency across a rack is about 10-15 ns. Furthermore, the rack can be constructed with tight tolerances to aid synchronization. For example, if all links are the same length with a tolerance of $\pm$ 2 cm, the propagation delay would vary by a maximum of 0.2 ns. Small physical distance also mitigates the impact of temperature variations that could lead to variable propagation delay.

Shoal leverages the WhiteRabbit synchronization technique [32, 37, 40, 45] to achieve synchronization with bit-level precision. WhiteRabbit has been shown to achieve sub-50 picoseconds of synchronization precision [45]. The main idea is to couple frequency synchronization with a time synchronization protocol (§6.1).

Frequency synchronization is achieved by distributing a global clock to all the nodes and switches in the rack. This global clock is generally derived from one of the rack nodes, designated as the clock master. The clock can be distributed explicitly, or implicitly through Synchronized Ethernet (SyncE) [75] whereby nodes derive a clock from the data they receive and use this clock for their transmissions.

In Shoal, time synchronization protocol like PTP [70] or DTP [33] need to run only between the end nodes (and not the switches). At bootstrap, each switch's circuits are configured according to their respective schedule's configuration at time slot 1 (e.g. Fig. 4) and they do not change. End nodes then start running the time synchronization protocol. Once all the nodes are synchronized to a desired level of precision, they send a bootstrap signal to the switches, followed by actual data according to the fabric schedule (Fig. 3). Switches on receiving the bootstrap signal start reconfiguring their circuits according to their respective schedules (Fig. 4).

Slot size configuration. Overall, the guard band size is the sum of the reconfiguration delay, variability in propagation and the precision of synchronization. Given the guard band size, the slot size can be configured to balance the trade-off between latency and throughput: a smaller slot reduces epoch size resulting in smaller latency, yet it imposes higher guard band overhead resulting in smaller duty cycle and hence lower throughput.

Epoch size and multiple channels. In Shoal, two nodes exchange cells at the interval of an epoch. Therefore, each queue drains at the rate of one cell per epoch, meaning a smaller epoch size results in smaller queuing delay. We can reduce the epoch size by taking advantage of the fact that network links comprise multiple channels. For e.g., 100 Gbps links actually comprise four 25 Gbps channels, which can be switched independently. Thus, in Shoal, each channel is used to send cells to a quarter of the rack nodes in parallel. Given a fixed slot size (as determined based on guard band size), this shrinks the epoch size by a quarter ($epoch = \frac{N-1 \ slots}{num \ of \ channels}$). Finally, the actual cell size is determined by the slot size and channel speed, for e.g. a slot size of 20.5 ns (without guard band) will correspond to 64 B cells over a 25 Gbps channel.

3.5 Practical concerns

We now discuss a few practical concerns of the design.

Clock and data recovery (CDR). A key challenge for any network relying on fast circuit switches is that each node

source id	10 bits	destination id	10 bits
sequence number	22 bits	rate limit feedback	11 bits
start-of-packet	1 bit	end-of-packet	1 bit
last-cell-dropped	1 bit	CRC checksum	8 bits

Figure 5: Header fields in the 8 B cell header carried by each cell. Header field sizes assume a max of 1024 rack nodes.

needs to be able to receive traffic from different senders at each time slot. This requires that, at each time slot, the incoming bits are sampled appropriately so as to achieve error-free reception. The sampling is done by the Clock and Data Recovery (CDR) circuitry at the receiver and typically takes a few hundred microseconds [46]. However, we note that this is only a problem when using layer 0 circuit switches that operate at the raw physical layer, e.g., when using an optical circuit switch. Such a switch imposes no latency overhead but requires very fast CDR at the receiver in order to achieve a reasonable guard band. Recent work has shown that sub-nanosecond CDR is achievable in datacenter settings [13].

Electrical circuit switch can also operate at layer 1 [76]. When a circuit is established between ports $i \rightarrow j$, the switch retimes data received on port i before sending it to port j. With such switches, each link in the network is a point-to-point link and thus, fast CDR is not needed. Each switch, however, does need to be equipped with a small buffer to account for any differences in the clocks associated with ports i and j. For Shoal, only a few bits worth of buffering is required since the entire rack is frequency synchronized and the buffer is only needed to absorb any clock jitter.

Cell reordering and reassembly. The `sequence number` field in each cell's header is used to assemble cells in-order at the destination. Note that Shoal's congestion control is robust to reordering, as it operates at the granularity of individual subflows with a congestion window of size 1. Once the cells have been reordered, the `start-of-packet` and `end-of-packet` fields in the cell header are used to figure out the packet boundary, and the cells within each packet boundary are then assembled together to re-construct the original packet.

Failures. To detect failures, Shoal relies on the fact that a node sends a cell to every other node in the rack, even if there is no traffic to send, once every epoch. A path refers to the set of links and switches through which a node j sends a cell to some other node i once every epoch.

When a node i does not receive a cell from a node j in it's corresponding slot, either due to path failures or node failure, it conservatively infers that node j has failed, and i) stops sending any further cells to j, ii) notifies other nodes that it can no longer communicate with j, so other nodes stop forwarding cells destined to j via i, iii) forwards the last cell (if it happens to be i's local cell) it sent to j via some other node, and iv) discards all the outstanding cells it was supposed to forward to j. Shoal relies on a higher layer end-to-end transport protocol to recover from the loss of those outstanding cells. Finally, in case of an actual node failure, Shoal again relies on the transport protocol to recover all the cells that were queued to be forwarded at the failed node. If the failed node was the primary clock reference for synchronization, another node needs to take over and remaining nodes seamlessly switch to it as the new reference. ITU standard for SyncE [75] already supports this.

1 node	1 node-leaf link	1 leaf switch	1 leaf-spine link	1 spine switch
$\approx 1/N$	$\approx 1/N$	$\approx 1/\sqrt{2N}$	$\approx 2/N$	$\approx 2\sqrt{2}/\sqrt{N}$

Table 1: Fraction of failed slots against different failed components, for a two-stage clos topology. N = no of rack nodes.

Note that the failure detection mechanism is symmetric — when node i infers that node j has failed, it immediately stops sending cells to j, causing j to infer that i has failed, and hence immediately stop sending cells to i. This ensures the consistency of Shoal's closed-loop congestion control mechanism (§3.3.2), even in the face of failures.

One of the consequences of Shoal's design is if a node can no longer "directly" communicate with some other node, either because the other node or the path to it has failed, it hurts node's throughput as the corresponding slot is marked as failed and hence goes unused (Table 1). We evaluate network performance against fraction of failed nodes in §7.3.

Scalability. Shoal's scalability is mainly limited by two factors: i) On-chip resource consumption on the NIC, in particular on-chip memory, and ii) epoch size, which contributes to network latency. The on-chip memory consumption for Shoal scales as $\Theta(N^2 log(N))$ bits (§4). Even for a very dense rack comprising $\sim$1000 nodes, this results in a memory consumption only of the order of a megabyte. On the other hand, epoch size increases linearly with the number of nodes (§3.4). The impact of increasing epoch size on network latency is evaluated in §7.3.

4 Implementation

In this section, we discuss our FPGA-based implementation of Shoal's switch and NIC. We used Bluespec System Verilog [57] ($\sim$1,000 LOCs). Our design runs at a clock speed of 156.25 MHz, thus each clock cycle is 6.4 ns.

4.1 Switch design

Our circuit switch operates at layer 1, i.e., data traversing the switch is routed through the PHY block at the ingress and egress ports (Fig. 6). The mapping between the ingress and egress ports varies at every time slot according to the static schedule. This mapping is implemented using p different p:1 multiplexers, where p is the number of ports in the switch. The control signals to these multiplexers are driven by p registers, one per multiplexer. In each time slot, all the p registers are configured in parallel according to the schedule.

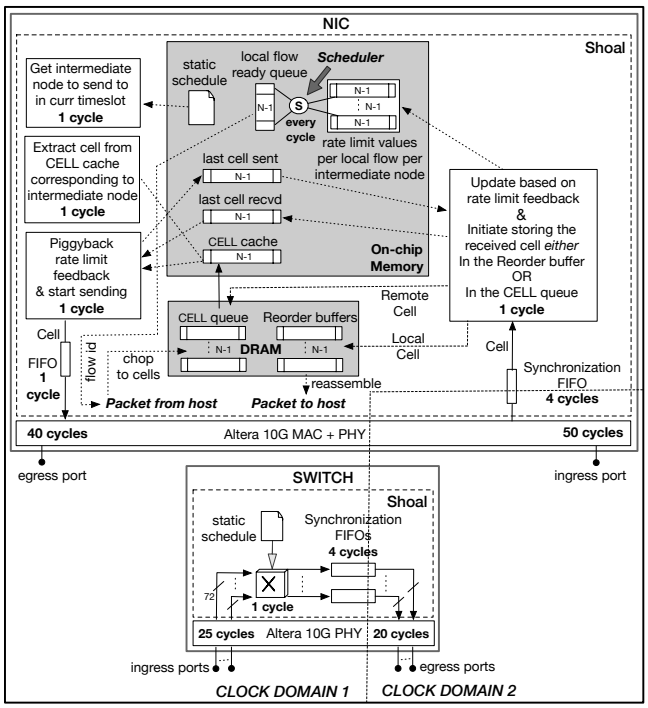


Figure 6: Switch and NIC implementation with the latency of each block. Clock cycle is 6.4 ns. N = num of rack nodes.

Hence, the switch reconfiguration delay is simply the time it takes to update the registers, which can be done in one clock cycle. Our switch is driven by the clock that drives the interface to PHY. The interface to 10G Ethernet PHY (XGMII) runs at 156.25 MHz, resulting in reconfiguration delay of 6.4 ns for our switch. However, for higher link speeds the clock frequency can be higher, for e.g., at 50 Gbps, the interface to PHY (LGMII) runs at 390.625 MHz [55], yielding a reconfiguration delay of 2.5 ns.

The transmit and receive paths of the switch are located in two separate clock domains: the transmit path is driven by the clock distributed throughout the rack, while the receive path is driven by the clock recovered from the incoming bits. To move data safely across clock domains, we use synchronization FIFOs. The total port-to-port latency of our switch is 50 cycles (320 ns): PHY block (45 cycles) + switching (1 cycle) + synchronization FIFO (4 cycles).

4.2 NIC design

Fig. 6. shows the routing and congestion control pipelines implemented in Shoal's NIC. Each NIC maintains a cell cache on the on-chip memory, of size $N-1$ cells, which stores the next cell to forward per intermediate node. Remaining cells sit in the DRAM. The backpressure-based mechanism underpinning Shoal's congestion control (§3.3.2) is implemented using two vectors of size $N-1$ each, that record the last cell sent (received) to (from) each intermediate node, and a $(N-1)\times(N-1)$ matrix that stores the rate limit feedback received from each intermediate node for each active local flow. The scheduler uses these data structures to schedule local cells into a ready queue in accordance with the logic described in § 3.3.2 and § 3.3.3.

NIC latency is dominated by the PHY and MAC IP blocks, with the routing and congestion control logic only adding 4 and 5 cycles on the transmit and receive paths, resp. Thus, Shoal's additional mechanisms impose low overhead.

Appendix B details the resource consumption for Shoal's FPGA-based implementation.

5 Power and cost implications

We now compare the power and cost of a Shoal network to that of a packet-switched network (PSN). Along with the performance evaluation in §7, we demonstrate that for 71% lower power and an estimated cost reduction of up to 40%, Shoal's circuit-switched fabric can reduce tail latency by up to 2× as compared to state-of-the-art congestion control protocols such as NDP [25] atop a PSN.

We analyze a 512-node rack. For a PSN, we consider today's packet switches [58, 81], which support 64 ports at 50 Gbps and consume a maximum of 350 W [47, 49]. Nodes have 50 Gbps NICs with copper cables (i.e., no opto-electronic transceivers) and connecting them using a non-blocking Folded Clos topology requires 24 such switches. For Shoal, extrapolating from today's circuit switches [76], we estimate that a 64×50 Gbps circuit switch would consume 38.5 W. To compensate for the throughput overhead of detouring packets, each node is equipped with 100 Gbps links. So the Shoal network has 48 circuit switches. The small physical size of the actual circuit switch ASIC means that the space required for the extra switches is manageable. Based on current SoC trends [59, 73, 79], we expect the NIC to be integrated with the CPU on a single SoC and to benefit from the same 10× reduction in power consumption. Given a typical power consumption of 12.4 W for today's 100 Gbps NICs [77], this would lead to an estimated power consumption of 1.37 W for the Shoal's NIC (including 11% overhead as computed in Appendix B) and of 0.62 W for PSN's. Thus, the total power of the Shoal network is 2.55 KW, 71% lower than PSN (resp. 8.72 KW). Lower power density is critical because a rack's total power has a hard limit around 15 KW [35, 60, 66].

Quantifying the cost of the Shoal network is harder as it requires determining the at-volume cost of circuit switches. Circuit switches can be electrical or optical; today, electrical circuit switches are commercially used in scenarios like HDTV [76] and are capable of fast switching while fast optical switches only exist as research prototypes [16, 17, 30, 36, 48, 52]. Thus, instead of focussing on absolute costs, we ask: *how cheap would circuit switches need to be, relative to equivalent packet switches, for Shoal to offer cost benefits over PSN?* We assume Shoal NICs cost between 2 and 3× PSN NICs to account for the 2× bandwidth and extra func-

tionality they provide. Fig. 11 shows how the relative cost of the Shoal network varies as a function of the relative cost of circuit switches to packet switches. A Shoal network would cost the same as a PSN as long as circuit switches are 33.3–41.6% the cost of packet switches while providing power and performance gains. If the cost circuit switches was 9.6–18.3% of the cost of packet switches, then Shoal would offer a 40% cost reduction. While absolute costs are hard to compare as they also depend on several non-technical factors, the analysis below and our estimations based on hardware costs indicate that at-volume circuit switches could cost as low as 15% of equivalent packet switches.

The lack of buffering, arbitration and packet inspection in circuit switches means that they are fundamentally simpler than packet switches which should mean lower cost. For electrical switches, designers often use the switch package area as a first-order approximation of switch cost—the actual chip area dictates yield during fabrication and therefore fabrication cost while the total package area dictates the assembly and packaging cost. In today's packet switches 50% of the total area is attributed to memory, 20% to packet processing logic while 30% is due to serial I/O (SerDes) [62]. Electrical circuit switches, by contrast, have no memory and packet processing, so the first two components have negligible contribution. While the amount of I/O bandwidth in a circuit switch remains the same, the actual SerDes is much smaller because they are only retiming the signals, instead of serializing and deserializing data from a high-rate serial channel to lower-rate parallel channels. Even assuming the SerDes are only halved in size, the total packaged area for circuit switching could be as low as 15% that of a packet switch. Overall, this analysis indicates that, with volume manufacturing, the relative cost of electrical circuit switches can be low enough for Shoal to simultaneously reduce both power and cost as compared to PSN.

Looking ahead, optical circuit switches hold even more promise: as bandwidth increases beyond 100Gbps per channel, copper transmission becomes noise limited even at intra-rack distances and optical transmission becomes necessary [64]. Optical circuit switches further reduce cost and power because they obviate the need for expensive transceivers for opto-electronic conversions. However, a few technical challenges need to be solved for optical switches to be used in Shoal [7]. For example, while several technologies being studied in the optics community can achieve nanosecond switching, practical demonstrations have been limited to 64-128 ports [12]. Another longstanding challenge is to achieve fast CDR at layer 0 although recent work has shown the feasibility of such CDR within 625ps [13].

6 Prototype

In this section, we evaluate our FPGA-based implementation of Shoal through a 8-node prototype, shown in Fig. 7.

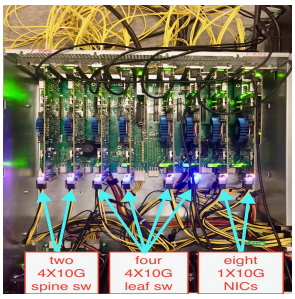


Figure 7: Shoal's FPGA-based prototype.

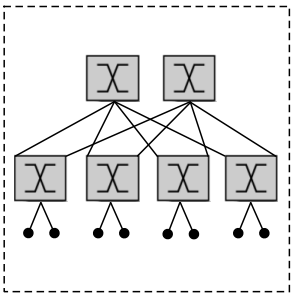


Figure 8: Shoal prototype's physical topology.

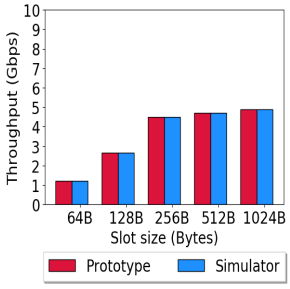


Figure 9: [Prototype] Avg destination throughput for full permutation matrix.

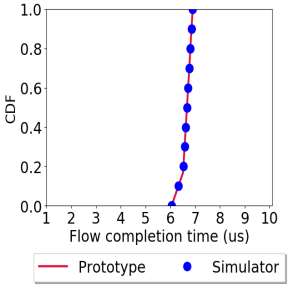


Figure 10: [Prototype] Flow completion time for 7:1 synchronized incast.

6.1 Prototype setup

Our prototype comprises eight Terasaic DE5-Net boards [63], each with an Altera Stratix V FPGA [80] and four 10 Gbps SFP+ transceiver modules. Two FPGAs are used to implement eight NICs, one per port. The remaining six FPGAs implement six 4-port circuit switches. The switches are connected in a leaf-spine topology and the NICs are connected to the leaf switches as shown in Fig. 8. We connect all eight FPGAs to a Dell T720 server. We use the PCIe clock as the global clock and distribute it to the Phase-locked loop (PLL) circuit running on each FPGA. Thus all the local clocks derived from the respective PLL circuits on each FPGA are frequency synchronized. For time synchronization we use DTP [33].

Guard band. Our prototype achieves synchronization precision of less than a clock cycle. Further, the switch reconfiguration delay is one clock cycle (§4.1), and all wires are of same length. Hence a guard band of one clock cycle (6.4 ns) is sufficient.

Slot size. To keep the guard band overhead to around 10%, we select a slot size of 12 clock cycles (76.8 ns). This includes 1 cycle of guard band overhead and 24 B (3 cycles) of Altera MAC overhead. Thus the usable slot size equals 8 cycles (51.2 ns), which translates to 64 B cells at 10 Gbps link speed. The epoch size equals 0.53 us.

6.2 Prototype experiments

We used the prototype to verify that our implementation achieves throughput and latency in accordance with the design. We also use it to cross-validate our simulator which, in turn, is used for a large-scale evaluation regarding the viability and benefits of a real world deployment of Shoal.

Throughput. We consider a permutation matrix with $N = 8$ flows: each node starts a single long-running flow to another random node such that each node has exactly one incoming and outgoing flow. For throughput, this is the worst-case traffic matrix. In Fig. 9, we show performance in terms of *destination throughput*, measured as the amount of "useful" cells (i.e., excluding the cells to forward and the empty ones) received by each destination. For full permutation matrix, the throughput for Shoal is expected to converge to $\sim$50% of the ideal throughput. Interestingly, however, the throughput is significantly lower for smaller slot sizes, and it converges towards 50% only for larger slot sizes. This is an artifact of the small scale of our prototype, which causes the node-to-node cell propagation latency (1.57 us: 40 ns of wire propagation latency + 3$\times$320 ns of switching latency (dominated mostly by Altera 10G PHY latency) for three switches along the path + 576 ns of Altera 10G MAC and PHY latency at the two end nodes (Fig. 6)) to be higher than the epoch size (0.53 us for 64 B cells). The problem is that Shoal's congestion control mechanism prevents a node from sending its next cell to an intermediate node until it has received feedback from it. Therefore, if the cell propagation latency spans multiple epochs, the overall throughput suffers as senders cannot fully utilize their outgoing bandwidth. As the slot size increases, the ratio between the cell propagation latency and the epoch size decreases, and this explains why in our prototype the throughput improves with larger slots. In practice, however, even for modest-sized racks, this issue will not occur as the cell propagation latency will be much smaller than the epoch size, and can be easily accomodated in the schedule as explained in Appendix A.

Latency. We consider all-to-one incast, where seven nodes each send 448 B of data (seven 64 B cells) to the same destination at the same time. Fig. 10 shows the distribution of flow completion time (FCT) of all seven flows. The queue at each node corresponding to the destination node grows upto a maximum of 7 (Eq. 5). This results in maximum FCT of 6.9 us : 3.76 us of queuing delay plus 2$\times$1.57 us of propagation latency. Also note that the difference between the fastest and slowest flow is fairly small (6.05 us vs. 6.9 us), highlighting Shoal's fair queuing.

Overall, across all experiments, the prototype and simulation results were in agreement.

7 Simulation

We complement the prototype experiments in §6 with simulations to investigate the scalability of Shoal.

7.1 Simulation setup

We use the packet-level simulator that was cross-validated against our prototype (§6). We simulate a 512-node rack, where each node is equipped with an interface bandwidth of 100 Gbps, connected using a full bisection bandwidth Clos topology comprising circuit switches.

Guard band. We assume a guard band of 2.75 ns, based on a 2.5 ns switch reconfiguration delay (§4.1) and 0.25 ns to account for any variability in propagation and synchronization imprecision (§3.4).

Slot size. To keep the guard band overhead to around 10% (resulting in max throughput of 90 Gbps), we select the slot size of 23.25 ns. This results in 20.5 ns of usable slot size. As explained in §3.4, we use the fact that existing 100 Gbps links comprise 4$\times$25 Gbps channels, resulting in 4 parallel uplinks and an epoch size of 2.9 us. Finally, usable slot size of 20.5 ns translates to 64 B cells at 25 Gbps channel speed.

7.2 Microbenchmarks

We start with a set of microbenchmarks to verify that the behavior observed in our testbed holds at large scale too.

Throughput. In Fig. 12, we plot the average destination throughput, as defined in §6.2, for the permutation traffic matrix: each communicating node sends and receives one flow. We vary the number of communicating pairs from 1 to 512. As there is no contention at any of the source and destination nodes, the ideal destination throughput equals the maximum interface bandwidth. However, for Shoal, as the number of communicating pairs increases, so does the amount of detouring traffic, resulting in the expected throughput trend: it starts from the peak value for a single flow and then monotonically decreases until it halves when all pairs are communicating (full permutation traffic matrix).

Fairness. To verify Shoal's fairness, we ran several workloads comprising a variable number of flows from 50 to 1,024 with randomly selected sources and destinations. We compared the throughput achieved by each flow against its ideal throughput computed using the max-min water-filling algorithm [8]. Across all workloads, 99% of the flows achieve a throughput within 10% of the ideal one. This shows that, despite the simplicity of its mechanisms, Shoal closely approximates max-min fairness.

Latency under Incast. We evaluate Shoal under incast, the most challenging traffic pattern for low latency. A set of nodes send a small flow of size 130 KB each, to the same destination at the same time. In Fig. 13, we plot the flow completion time (FCT) of the slowest flow as well as the mean completion time, against increasing number of sending nodes. As expected, at each intermediate node, the queue corresponding to the destination node grows linearly with increasing number of sending nodes, but bounded by the incast degree of the destination (Eq. 5). Hence the FCT for the slowest flow increases linearly and is also the optimal

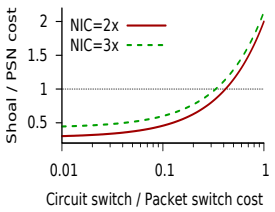


Figure 11: Relative cost of Shoal network vs. packet switch network (PSN).

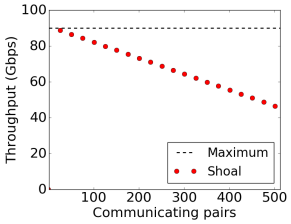


Figure 12: Average destination throughput vs. size of permutation matrix.

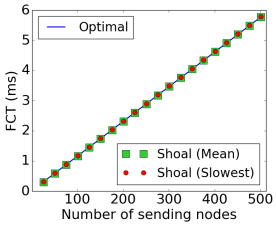


Figure 13: Flow completion times against synchronized short flow incast.

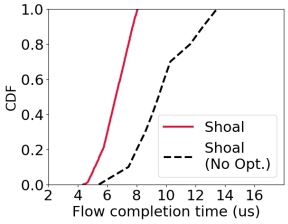


Figure 14: Reduced impact of detouring on latency via optimizations in §3.3.3.

maximum FCT under such incast. The mean completion time coincides with the slowest flow's FCT, thus highlighting Shoal's fair queuing.

Reducing the impact of detouring on network latency. To show that the optimizations described in §3.3.3 indeed improve the network latency, we choose two nodes, Node-511 (source) and Node-0 (destination), to exchange short flows (20 KB) at regular intervals, and generate random background traffic amongst the remianing nodes. We plot the distribution of flow completion time (FCT) of short flows exchanged between nodes 511 and 0 in Fig. 14. The optimizations in §3.3.3 enable Shoal to achieve much smaller and predictable FCT for target short flows—at the source, Shoal selectively adds cells to local queues where there is low contention, and at the intermediate node the queue length is bounded to two regardless of the cross-traffic, as the incast degree of Node-0 is one (Eq. 5). However, without the optimizations, the cross-traffic due to detouring significantly increases the FCT of target short flows.

7.3 Datacenter workloads

We now investigate the performance of Shoal in dynamic settings, using more realistic workloads.

Workload. We generate a synthetic workload, modeled after published datacenter traces [2, 22]. Flow sizes are heavy tailed, drawn from a Pareto distribution with shape parameter 1.05 and mean 100 KB [3, 4]. Flows arrive according to a Poisson process and each simulation ends when one million flows have completed. Flow sources and destinations are chosen with uniform probability across all nodes (we will study the impact of skewed workloads in §7.4).

Network load. We define network load $L = \frac{F}{R \cdot N \cdot \tau}$ where F is the mean flow size, R is the per-node bandwidth, N is the number of nodes, and τ is the mean inter-arrival flow time, e.g., $L = 1$ means that, on average, there are N active flows.

Evaluation metric. We evaluate Shoal based on the flow completion time (FCT) for short flows ($\leq$100 KB) and average goodput (i.e., throughput after accounting for the 8 B cell header overhead (§3.5)) for long flows ($\geq$1 MB).

Baseline 1: Direct-connect network. We start with comparing Shoal against a rack-scale network using a direct-connect topology. We arranged the 512 nodes into a 3D torus, which is the topology used in the AMD SeaMicro 15000-OP [79]. As with the Shoal network, we assume an aggregate node bandwidth of 100 Gbps. We use R2C2 [15] for congestion control. For all values of load, Shoal consistently outperforms the rack-scale setup up to a factor of 14.9 for tail FCT for short flows (resp. a factor of 4.8 for avg goodput for long flows). This is due to the multi-hop nature of direct-connect topologies; it significantly increases the end-to-end latency as queuing can occur at any hop. Further, node bandwidth is also used to forward traffic originating several hops away, which reduces the overall throughput. This does not occur in Shoal as packets only traverse one hop and the congestion control guarantees bounded queues.

Baseline 2: Packet-switched network. Now we compare Shoal against a 512-node packet-switched network (PSN), that connects the nodes using Clos topology with full-bisection bandwidth. The interface bandwidth of each node is 50 Gbps. Thus Shoal is provisioned with 2$\times$ bandwidth, to compensate for the throughput overhead of detouring packets. Note that despite the extra bandwidth, Shoal's power is still estimated to be lower than that of PSN with a comparable or lower cost (§5). As baselines, we use DCTCP [2], NDP [25] and DCQCN [54] as the three state-of-the-art congestion control algorithms atop a packet-switched network. The baselines are based on the simulator used in [25]. We assume 1500 B packets for all of them. DCTCP and DCQCN use standard ECMP routing, with the congestion window size of 35 packets and queue size of 100 packets. NDP uses packet spraying for routing with initial window size of 35 packets and queue size of 12 packets.

As shown in Fig. 15a, at low to moderate load, Shoal exhibits an average FCT comparable to DCQCN and DCTCP and slightly higher than NDP. This increase is a consequence of the use of detouring due to the static schedule (3.3.1). However, Shoal outperforms DCTCP and DCQCN in terms of tail FCT for short flows by a factor of 3$\times$ at low load and 2$\times$ at high load (resp. outperforms NDP by a factor of 1.5$\times$ and 2$\times$). The reason for this is three-fold: i) 2$\times$ bandwidth per node in Shoal reduces the serialization delay, ii) selectively adding cells from new flows to local queues with low contention reduces queuing delay at the source, and iii)

Shoal's congestion control ensures small and bounded queuing at intermediate nodes, thus reducing the queuing delay at intermediate nodes. Shoal also outperforms all three baselines in terms of long flow goodput (Fig. 15b) by a factor of 1.7×, even at high load. This is primarily due to the fact that each Shoal node is equipped with more bandwidth.

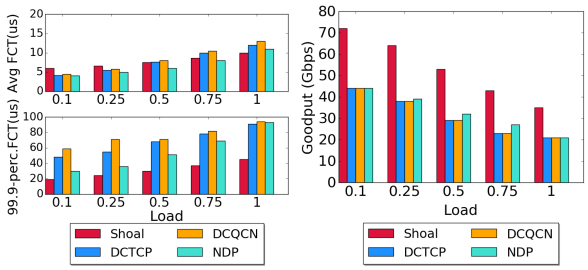
Queuing and reordering. To validate our claim that Shoal operates with very small queues, we plot the maximum queuing observed across all nodes in Fig. 16. Even at high load, the maximum queue size is 11 cells (704 B) and the maximum aggregate queue per node is 336 cells (21 KB). Maximum cell reordering within a flow across all nodes and across all values of load is 200 KB (Fig. 16).

Node failures. We now focus our attention to the impact of node failures. We ran the same workload as in the previous experiments ($L = 1$) but at the beginning of each experiment we fail an increasing fraction of nodes (up to 50%). As expected, the goodput decreases linearly (2× worse for 50% failure rate, Fig. 17) because the slots corresponding to the failed nodes are wasted. We can alleviate this with a more sophisticated mechanism that, on detecting long-term failures, updates the schedule of both rack nodes and switches to discount the failed nodes. FCT also increases with increasing failed nodes, as the number of paths along which cells from a flow can be sent is reduced, resulting in higher subflow collision and increased queuing. However, Fig. 17 shows that even for high failure rate the increase in completion time is rather marginal, e.g., 1.5× for a 40% failure rate (resp. 1.2× for 20% failure rate), thus making Shoal amenable even for sealed rack-scale deployments in which replacing failed nodes is not possible.

Impact of epoch size on network latency. Next, we study the impact of epoch size on the FCT. Larger epoch size results in higher latency (§3.4). In the first experiment, we reduced the number of channels from 4×25 Gbps to 2×50 Gbps, thus doubling the epoch size. This increased tail FCT by 1.26× at low load (resp. 1.15× at high load). In the second experiment, we kept the number of channels constant at 4, and increased the number of nodes to 1,024, again doubling the epoch size. In this case, tail FCT at low load grew by 1.28× (resp. 1.2× at high load). This, in turn, shows that Shoal's performance scales reasonably well with number of nodes, making it suitable even for very dense racks.

7.4 Disaggregated workloads

Finally, we evaluate the performance of Shoal on disaggregated workloads, based on recently published traces [19]. These traces comprise a variety of real-world applications, including batch processing, graph processing, interactive queries, and relational queries. To generate the workloads, we mapped each rack node to one of the server resources (CPU, memory, and storage) and created flows between them following the distributions observed in these traces. This yielded a much more skewed workload than the one in §7.3



(a) Flow completion time. (b) Average flow goodput.

Figure 15: Flow completion time (short flows ≤100 KB) and avg flow goodput (long flows ≥1 MB) vs. traffic load.

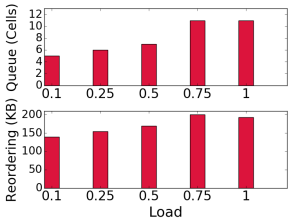


Figure 16: Max queue size and max cell reordering vs. traffic load.

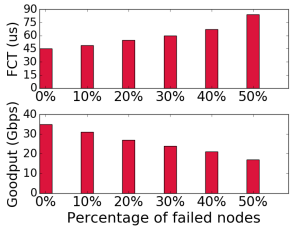
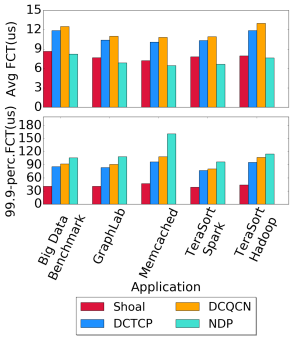
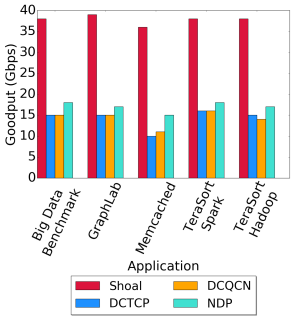


Figure 17: Short flow 99.9p FCT and long flow avg goodput vs. failure.



(a) Flow completion time.



(b) Average flow goodput.

Figure 18: Flow completion time (short flows ≤100 KB) and avg flow goodput (long flows ≥1 MB) for different applications with disaggregated workload.

with more than 84% of the flows being generated among a third of the nodes.

Fig. 18 shows the results for all the six applications, assuming a mean inter-arrival time of 12.65 ns. Shoal significantly outperforms the baselines in terms of both the tail FCT for short flows (factor of 2× or more) and avg goodput for long flows (factor of 2.5×). As explained in §7.3, this is due to higher bandwidth provisioning in Shoal in combination with its highly effective scheduling and congestion control mechanisms (resulting in maximum queue size of just 10 cells across all applications).

These results show the versatility of Shoal and its ability to carry different types of traffic, including disaggregated workloads, with high throughput and low latency.

8 Discussion

The focus of this paper is on the design of a network for disaggregated racks. Here we discuss a few open questions.

Integrating a Shoal rack with rest of the datacenter. A key question is how to seamlessly integrate a rack-scale network, such as Shoal, with the rest of the datacenter, which might consist of both disaggregated and traditional racks. Existing rack-scale designs [35] typically use a few rack nodes as gateways that are then directly connected to the datacenter network. Such a design could be adapted for Shoal as well—the gateway nodes would act as a bridge between Shoal's network stack and datacenter-wide network stack, such as TCP/IP over Ethernet. There are, however, several key challenges that still remain to be addressed such as the interplay between different congestion controls and how to design the interface between IP packets and Shoal cells (e.g., packet reassembly/fragmentation and encapsulation).

Running applications on top of Shoal. Shoal is a link layer architecture with support for congestion control. Running applications on top of Shoal, however, requires a transport layer providing application multiplexing, reliability, and flow control. One option would be to re-use an existing transport layer such as TCP, although the impact of the interaction of its congestion-control mechanism with Shoal's remains an open question. Another approach would be to design a Shoal-specific transport layer. This would be significantly simpler as congestion control is already handled by Shoal. We leave the exploration of these options for future work.

9 Related work

Disaggregated architectures promise significant cost, power, and performance gains [56, 67, 71]. However, unlocking these benefits requires solving numerous challenges.

Topology and technology. Several topologies have been investigated for disaggregated racks. Direct-connect topologies whereby each node is connected to a small subset of other rack nodes through point-to-point links are common in super computers and have been adopted in some commercial disaggregated racks. For example, both AMD SeaMicro [79] and HP Moonshot [69] use the 3D Torus topology and custom routing. Motivated by the fact that the best direct-connect topology is workload-dependent, reconfigurable networks have emerged as an attractive alternative. At rack scale, XFabric [35] combines circuit switches with SoC-based packet switches to reconfigure the rack's topology on the fly, while for datacenter networks, electrical [10], optical [11, 21, 24, 42, 53] and wireless technologies [23] that operate like a circuit switch have been proposed. These solutions typically rely on a centralized controller to schedule traffic. This imposes significant communication and computation overhead and requires accurate demand estimation.

Congestion control through tight network-host coupling. Shoal's congestion control tightly couples the network fabric with host network stack. For packet-switched networks, there is already a trend towards tighter coupling between the network and servers for low latency congestion control in datacenters; for example, using ECN as a feedback signal in DCTCP [2]. Congestion control mechanisms specialized for RDMA over converged Ethernet, such as DCQCN [54] and TIMELY [39], also rely on a closer coupling with the network. Shoal is an extreme design point in this direction as the coupling of its congestion mechanism to its fabric achieves bounded queuing and fairness despite very high multi-pathing. For direct-connect topologies, R2C2 [15] is a recently proposed congestion control that relies on broadcasting of flow events across the rack. It achieves computation tractability at the expense of network utilization.

Load-balanced switch. In 2002, load-balanced switches [9, 29] were proposed as a way to obviate arbitration in monolithic switches. Shoal's fabric operates like a load-balanced switch. However, instead of using an explicit intermediate stage (i.e., special nodes for detouring) as in the original proposal, Shoal detours cells through other rack nodes. Furthermore, while the original technique focused on monolithic switches, we scale it to a hierarchy of switches.

The load-balancing approach is also at the core of RotorNet [38], an optical network for datacenters that does not require a centralized controller. By leveraging multiple rack uplinks, RotorNet reduces epoch duration and proposes a novel indirection technique that lowers the throughput impact of load balancing. However, it uses optical circuit switches with a relatively high reconfiguration delay (20 μs) and hence, requires a separate packet-switched network for low-latency traffic. It also does not ensure bounded queuing. In contrast, Shoal works atop circuit switches with nanosecond-reconfiguration, and proposes novel congestion control and scheduling mechanisms that achieve high throughput, low latency, fairness and bounded queuing for all flows atop a purely circuit-switched fabric.

10 Summary

We presented Shoal, a network architecture for disaggregated racks that couples a circuit-switched fabric with the nodes' network stack. The fabric operates like a rack-wide switch with a static schedule. Rack nodes achieve coordination-free scheduling by detouring their traffic uniformly, and implement backpressure-based congestion control which achieves fairness and bounded queuing. Our FPGA-based prototype achieves good performance and illustrates that Shoal's mechanisms are amenable to hardware implementation. Our results show that Shoal can achieve high throughput and low latency across diverse workloads while operating comfortably within the rack's power budget. This demonstrates that disaggregated architectures can be deployed using today's technologies and not need be gated on the viability of future advancements in low-power technologies for packet-switches.

Acknowledgments

We thank the anonymous reviewers, George Porter, Michael Schapira, and our shepherd, Alex Snoeren, for their useful comments and suggestions. This research was partially supported by DARPA CSSG (D11AP00266), NSF (1053757, 1440744, 1422544, 1413972, and 1704742), European Unions Horizon 2020 research and innovation programme under the SSICLOPS project (agreement No. 644866), Google Faculty Research Award, Microsoft Research PhD scholarship, and gifts from Cisco, Altera and Bluespec.

References

[1] M. Al-Fares, A. Loukissas, and A. Vahdat. A Scalable, Commodity Data Center Network Architecture. In *SIGCOMM*, 2008.

[2] M. Alizadeh, A. Greenberg, D. A. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta, and M. Sridharan. Data Center TCP (DCTCP). In *SIGCOMM*, 2010.

[3] M. Alizadeh, A. Kabbani, T. Edsall, B. Prabhakar, A. Vahdat, and M. Yasuda. Less Is More: Trading a Little Bandwidth for Ultra-Low Latency in the Data Center. In *NSDI*, 2012.

[4] M. Alizadeh, S. Yang, M. Sharif, S. Katti, N. McKeown, B. Prabhakar, and S. Shenker. pFabric: Minimal Near-optimal Datacenter Transport. In *SIGCOMM*, 2013.

[5] K. Asanovic. FireBox: A Hardware Building Block for 2020 Warehouse-Scale Computers. In *FAST*, 2014. Keynote.

[6] S. Balakrishnan, R. Black, A. Donnelly, P. England, A. Glass, D. Harper, S. Legtchenko, A. Ogus, E. Peterson, and A. Rowstron. Pelican: A Building Block for Exascale Cold Data Storage. In *OSDI*, 2014.

[7] H. Ballani, P. Costa, I. Haller, K. Jozwik, K. Shi, B. Thomsen, and H. Williams. Bridging the Last Mile for Optical Switching in Data Centers. In *Optical Fiber Communication Conference (OFC)*, 2018.

[8] D. Bertsekas and R. Gallager. *Data Networks*. Prentice Hall, 1987.

[9] C.-S. Chang, D.-S. Lee, and Y.-S. Jou. Load Balanced Birkhoff-von Neumann Switches, Part I: One-stage Buffering. *Comput. Commun.*, 25(6), Apr. 2002.

[10] A. Chatzieleftheriou, S. Legtchenko, H. Williams, and A. Rowstron. Larry: Practical Network Reconfigurability in the Data Center. In *NSDI*, 2018.

[11] K. Chen, A. Singla, A. Singh, K. Ramachandran, L. Xu, Y. Zhang, X. Wen, and Y. Chen. OSA: An Optical Switching Architecture for Data Center Networks with Unprecedented Flexibility. In *NSDI*, 2012.

[12] Q. Cheng, A. Wonfor, J. L. Wei, R. V. Penty, and I. H. White. Demonstration of the feasibility of large-port-count optical switching using a hybrid Mach-Zehnder interferometer-semiconductor optical amplifier switch module in a recirculating loop. *Optics Letters*, 39(18), 2014.

[13] K. Clark, H. Ballani, P. Bayvel, D. Cletheroe, T. Gerard, I. Haller, K. Jozwik, K. Shi, B. Thomsen, P. Watts, H. Williams, G. Zervas, P. Costa, and Z. Liu. Sub-Nanosecond Clock and Data Recovery in an Optically-Switched Data Centre Network. In *European Conference on Optical Communication (ECOC), Post-deadline paper*, 2018.

[14] R. Colwell. The chip design game at the end of Moore's law. In *HotChips*, 2013.

[15] P. Costa, H. Ballani, K. Razavi, and I. Kash. R2C2: A Network Stack for Rack-scale Computers. In *SIGCOMM*, 2015.

[16] M. Ding, A. Wonfor, Q. Cheng, R. V. Penty, and I. H. White. Scalable, Low-Power-Penalty Nanosecond Reconfigurable Hybrid Optical Switches for Data Centre Networks. In *Proceedings of the Conference on Lasers and Electro-Optics (CLEO)*, 2017.

[17] V. Eramo and M. Listanti. Power Consumption in Bufferless Optical Packet Switches in SOA Technology. *IEEE/OSA Journal of Optical Communications and Networking*, 1(3), 2009.

[18] H. Esmaeilzadeh, E. Blem, R. St. Amant, K. Sankaralingam, and D. Burger. Dark Silicon and the End of Multicore Scaling. In *ISCA*, 2011.

[19] P. X. Gao, A. Narayan, S. Karandikar, J. Carreira, S. Han, R. Agarwal, S. Ratnasamy, and S. Shenker. Network requirements for resource disaggregation. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, Nov. 2016.

[20] P. X. Gao, A. Narayan, G. Kumar, R. Agarwal, S. Ratnasamy, and S. Shenker. phost: Distributed near-optimal datacenter transport over commodity network fabric. In *ACM Conference on Emerging Networking Experiments and Technologies (CoNEXT)*, page 1. ACM, 2015.

[21] M. Ghobadi, R. Mahajan, A. Phanishayee, N. Devanur, J. Kulkarni, G. Ranade, P.-A. Blanche, H. Rastegarfar, M. Glick, and D. Kilper. ProjecToR: Agile Reconfigurable Data Center Interconnect. In *SIGCOMM*, 2016.

[22] A. Greenberg, J. R. Hamilton, N. Jain, S. Kandula, C. Kim, P. Lahiri, D. A. Maltz, P. Patel, and S. Sengupta. VL2: A Scalable and Flexible Data Center Network. In *SIGCOMM*, 2009.

[23] D. Halperin, S. Kandula, J. Padhye, P. Bahl, and D. Wetherall. Augmenting Data Center Networks with Multi-gigabit Wireless Links. In *SIGCOMM*, 2011.

[24] N. Hamedazimi, Z. Qazi, H. Gupta, V. Sekar, S. R. Das, J. P. Longtin, H. Shah, and A. Tanwer. Firefly: A reconfigurable wireless data center fabric using free-space optics. In *SIGCOMM*, 2014.

[25] M. Handley, C. Raiciu, A. Agache, A. Voinescu, A. Moore, G. Antichi, and M. Wojcik. Re-architecting datacenter networks and stacks for low latency and high performance. In *SIGCOMM*, 2017.

[26] B. Ji, C. Joo, and N. B. Shroff. Exploring the inefficiency and instability of Back-Pressure algorithms. In *INFOCOM*, 2013.

[27] K. Katrinis, D. Syrivelis, D. Pnevmatikatos, G. Zervas, D. Theodoropoulos, I. Koutsopoulos, K. Hasharoni, D. Raho, C. Pinto, F. Espina, S. Lopez-Buedo, Q. Chen, M. Nemirovsky, D. Roca, H. Klos, and T. Berends. Rack-scale disaggregated cloud data centers: The dReDBox project vision. In *DATE*, 2016.

[28] K. Keeton. The Machine: An Architecture for Memory-centric Computing. `http://www.mcs.anl.gov/events/workshops/ross/2015/slides/ross2015-keeton.pdf`.

[29] I. Keslassy, S.-T. Chuang, K. Yu, D. Miller, M. Horowitz, O. Solgaard, and N. McKeown. Scaling Internet Routers Using Optics. In *SIGCOMM*, 2003.

[30] J. H. Kim, W. S. Kwon, H. Lee, K.-S. Kim, and S. Kim. A novel method to acquire ring-down interferograms using a double-looped mach-zehnder interferometer. In *Lasers and Electro-Optics (CLEO)*, 2014.

[31] I. Kuon and J. Rose. Measuring the Gap Between FPGAs and ASICs. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 26(2), Feb. 2007.

[32] M. Lapinski, T. Wlostowki, J. Serrano, and P. Alvarez. White Rabbit: a PTP Application for Robust Sub-nanosecond Synchronization. In *Proceedings of the International IEEE Sym-*

posium on Precision Clock Synchronization for Measurement Control and Communication, 2011.

[33] K. S. Lee, H. Wang, V. Shrivastav, and H. Weatherspoon. Globally Synchronized Time via Datacenter Networks. In *SIGCOMM*, 2016.

[34] S.-J. Lee, K. Lee, and H.-J. Yoo. Analysis and Implementation of Practical, Cost-Effective Networks on Chips. *IEEE Des. Test*, 22(5), Sept. 2005.

[35] S. Legtchenko, N. Chen, D. Cletheroe, A. Rowstron, H. Williams, and X. Zhao. XFabric: A Reconfigurable In-Rack Network for Rack-Scale Computers. In *NSDI*, 2016.

[36] O. Liboiron-Ladouceur, A. Shacham, B. Small, B. Lee, H. Wang, C. Lai, A. Biberman, and K. Bergman. The Data Vortex Optical Packet Switched Interconnection Network. *Journal of Lightwave Technology*, 26(13), 2008.

[37] M. Lipinski, T. Wlostowski, J. Serrano, P. Alvarez, J. D. G. Cobas, A. Rubini, and P. Moreira. Performance results of the first White Rabbit installation for CNGS time transfer. In *Proceedings of the International IEEE Symposium on Precision Clock Synchronization for Measurement Control and Communication*, 2012.

[38] W. M. Mellette, R. McGuinness, A. Roy, A. Forencich, G. Papen, A. C. Snoeren, and G. Porter. RotorNet: A Scalable, Low-complexity, Optical Datacenter Network. In *SIGCOMM*, 2017.

[39] R. Mittal, T. Lam, N. Dukkipati, E. Blem, H. Wassel, M. Ghobadi, A. Vahdat, Y. Wang, D. Wetherall, and D. Zats. TIMELY: RTT-based Congestion Control for the Datacenter. In *Sigcomm*, 2015.

[40] P. Moreira, J. Serrano, T. Wlostowski, P. Loschmidt, and G. Gaderer. White Rabbit: Sub-Nanosecond Timing Distribution over Ethernet. In *Proceedings of the International IEEE Symposium on Precision Clock Synchronization for Measurement Control and Communication*, 2009.

[41] S. Novakovic, A. Daglis, E. Bugnion, B. Falsafi, and B. Grot. Scale-out NUMA. In *ASPLOS*, 2014.

[42] G. Porter, R. Strong, N. Farrington, A. Forencich, P. Chen-Sun, T. Rosing, Y. Fainman, G. Papen, and A. Vahdat. Integrating Microsecond Circuit Switching into the Data Center. In *SIGCOMM*, 2013.

[43] A. Putnam, A. Caulfield, E. Chung, D. Chiou, K. Constantinides, J. Demme, H. Esmaeilzadeh, J. Fowers, G. P. Gopal, J. Gray, M. Haselman, S. Hauck, S. Heil, A. Hormati, J.-Y. Kim, S. Lanka, J. Larus, E. Peterson, S. Pope, A. Smith, J. Thong, P. Y. Xiao, and D. Burger. A Reconfigurable Fabric for Accelerating Large-Scale Datacenter Services. In *ISCA*, 2014.

[44] C. Raiciu, S. Barre, C. Pluntke, A. Greenhalgh, D. Wischik, and M. Handley. Improving Datacenter Performance and Robustness with Multipath TCP. In *SIGCOMM*, 2011.

[45] M. Rizzi, M. Lipinski, T. Wlostowski, J. Serrano, G. Daniluk, P. Ferrari, and S. Rinaldi. White Rabbit Clock Characteristics. In *Proceedings of the International IEEE Symposium on Precision Clock Synchronization for Measurement Control and Communication*, 2016.

[46] A. Rylyakov, J. E. Proesel, S. Rylov, B. G. Lee, J. F. Bulzacchelli, A. Ardey, B. Parker, M. Beakes, C. L. S. Christian W. Baks, and M. Meghelli. A 25 Gb/s Burst-Mode Receiver for Low Latency Photonic Switch Networks. *IEEE JOURNAL OF SOLID-STATE CIRCUITS*, 50(12), Dec. 2015.

[47] Spectrum. 32-port non-blocking 100gbe open ethernet switch system. `http://format.com.pl/site/wp-content/uploads/2015/09/pb_sn2700.pdf`.

[48] F. Testa and L. Pavesi, editors. *Optical Switching in Next Generation Data Centers*. Springer, 2017.

[49] Tomahawk. As7712-32x/as7716-32x series switch. `http://www.edge-core.com/_upload/images/AS7712_AS7716-32X_DS_R09_20170607.pdf`.

[50] L. G. Valiant and G. J. Brebner. Universal Schemes for Parallel Communication. In *ACM Symposium on Theory of Computing*, 1981.

[51] H. Wu, Z. Feng, C. Guo, and Y. Zhang. ICTCP: Incast Congestion Control for TCP in Data Center Networks. In *CoNEXT*, 2010.

[52] X. Ye, Y. Yin, S. Yoo, P. Mejia, R. Proietti, and V. Akella. DOS - A scalable optical switch for datacenters. In *ANCS*, 2010.

[53] X. Zhou, Z. Zhang, Y. Zhu, Y. Li, S. Kumar, A. Vahdat, B. Y. Zhao, and H. Zheng. Mirror Mirror on the Ceiling: Flexible Wireless Links for Data Centers. In *SIGCOMM*, 2012.

[54] Y. Zhu, H. Eran, D. Firestone, C. Guo, M. Lipshteyn, Y. Liron, J. Padhye, S. Raindel, M. H. Yahia, and M. Zhang. Congestion Control for Large-Scale RDMA Deployments . In *SIGCOMM*, 2015.

[55] 40g/50g high speed ethernet subsystem v1.0. `https://www.xilinx.com/support/documentation/ip_documentation/l_ethernet/v1_0/pg211-50g-ethernet.pdf`.

[56] Amazon joins other web giants trying to design its own chips. `http://bit.ly/1J5t0fE`.

[57] Bluespec. `www.bluespec.com`.

[58] Broadcom BCM56960 series (Tomahawk). `http://bit.ly/2wPnJHI`.

[59] Calxeda EnergyCore ECX-1000. `http://bit.ly/1nCgdHO`.

[60] Choosing the Optimal Data Center Power Density. `http://www.apc.com/salestools/VAVR-8B3VJQ/VAVR-8B3VJQ_R0_EN.pdf`.

[61] Cisco Nexus 7700 Switches Data Sheet. `http://www.cisco.com/c/en/us/products/collateral/switches/nexus-7000-series-switches/data_sheet_c78-728187.html`.

[62] Data Plane Programming at Terabit speeds. `https://p4.org/assets/p4_d2_2017_programmable_data_plane_at_terabit_speeds.pdf`.

[63] DE5-Net FPGA development kit. `http://de5-net.terasic.com.tw`.

[64] Electronic Design Blog. `http://bit.ly/2xsIhaU`.

[65] Facebook Engineering Blog. Introducing "Yosemite": the first open source modular chassis for high-powered microservers. `http://bit.ly/1MpHjuW`.

[66] How is a Mega Data Center Different from a Massive One? `http://www.datacenterknowledge.com/archives/2014/10/15/how-is-a-mega-data-center-different-from-a-massive-one`.

[67] How Microsoft Designs its Cloud-Scale Servers. `http://bit.ly/1HKCy27`.

[68] HP Labs. The Machine. `https://www.labs.hpe.com/the-machine`.

[69] HP Moonshot System. `http://bit.ly/1mZD4yJ`.

[70] IEEE Standard 1588-2008. `http://ieeexplore.ieee.org/xpl/mostRecentIssue.jsp?punumber=4579757`.

[71] Intel, Facebook Collaborate on Future Data Center Rack Technologies. `http://intel.ly/MRpOM0`.

[72] Intel Rack Scale Architecture. `http://ubm.io/1iejjx5`.

[73] Intel Xeon Processor D-1500. `http://intel.ly/2hrn1hR`.

[74] Introducing Big Basin: Our next-generation AI hardware.

http://bit.ly/2pQPu2S.

[75] ITU-T Rec. G.8262. http://www.itu.int/rec/T-REC-G.8262.

[76] Macom M21605 Crosspoint Switch. http://www.macom.com/products/product-detail/M21605/.

[77] Mellanox ConnectX-4 Ethernet Single and Dual QSFP28 Port Adapter Card User Manual. http://bit.ly/2flbOuZ.

[78] NVIDIA DGX-1 System Architecture. http://bit.ly/2xjV54K.

[79] SeaMicro SM15000 Fabric Compute Systems. http://bit.ly/1hQepIh.

[80] Stratix V FPGA. http://www.altera.com/devices/fpga/stratix-fpgas/stratix-v/stxv-index.jsp.

[81] The Next Platform Blog. Mellanox Comes Out Swinging With 100G Spectrum Ethernet. http://bit.ly/2cZXrwo.

[82] Under The Hood Of Googles TPU2 Machine Learning Clusters. http://bit.ly/2qfaUDd.

Appendix

A Accounting for propagation delay

Even at the scale of a rack, the propagation delay is not negligible as compared to the transmission time of a cell. This means that a cell sent at time slot t will not be received within the same slot at the receiver. More generally, say that the cell is received at slot $t+k$. For the feedback mechanism described in § 3.3.2 to work optimally in the face of such propagation delay, there should be at least k slots from the time node i transmits to node j and the time node j transmits to i, as j needs to know the destination of the last cell that i sent to j to send back the right rate limit feedback. We can easily re-arrange any cyclic permutation schedule such as in Fig. 3 to ensure this property, as long as k is less than half the number of slots in an epoch—when number of rack nodes, N, is odd, this constraint can be easily accommodated by inverting the order of the last $\frac{N-1}{2}$ columns. This would ensure that there are exactly $\frac{N-1}{2}$ slots between the slot in which i communicates with j and the one in which j communicates with i. This is shown in Fig. 19(a). For even N, this is slightly more complicated as it requires to introduce an additional empty slot per node to satisfy this requirement as demonstrated in Fig. 19(b), consequently resulting in a throughput reduction by a factor of $\frac{1}{N}$ for each node.

B Resource consumption for Shoal's FPGA-based implementation

To understand the resource utilization at scale, we synthesize our design onto a Stratix-V FPGA [80], which comprises 234,720 adaptive logic modules (ALMs) and 52 Mbits of BRAM. Assuming 500 nodes and 64-port switches, our NIC logic consumes about 84% of ALMs and 13% of BRAM (resp. 2% and 0.2% for the switch). Finally, we did a power analysis to quantify the overhead of Shoal's additional functionalities. We leverage the study done in [31] to translate the power consumption of our FPGA-based design into an equivalent ASIC design. Assuming a 500-node rack, this resulted in Shoal NIC's on-chip extra functionality – routing and congestion control – consuming up to 11% of the power consumed by commercial ASIC NICs, and Shoal switch's extra functionality – reconfiguration using a static schedule – consuming just 0.1% of the power consumed by commerical ASIC circuit switches.

Time slot

Node \ Time slot	1	2	3	4	5	6
1	2	3	4	7	6	5
2	3	4	5	1	7	6
3	4	5	6	2	1	7
4	5	6	7	3	2	1
5	6	7	1	4	3	2
6	7	1	2	5	4	3
7	1	2	3	6	5	4

(a) Fabric schedule for odd N

Time slot

Node \ Time slot	1	2	3	4	5	6	7	8
1	2	3	4	5	8	7	6	
2	3	4	5	6	1	8	7	
3	4	5	6	7	2	1	8	
4	5	6	7	8	3	2	1	
5	6	7	8		4	3	2	1
6	7	8	1		5	4	3	2
7	8	1	2		6	5	4	3
8	1	2	3		7	6	5	4

(b) Fabric schedule for even N

Figure 19: Fabric schedule accounting for propagation delay.

C Quality-of-Service

By default, Shoal assumes that each cell belongs to the same traffic class, and schedules them using a single per destination FIFO queue. However, Shoal can be easily extended to support multiple traffic classes by maintaining multiple per destination FIFO queues, one per traffic class, and scheduling cells from the FIFOs in the order of their priority. Note that in this case the queue bound (§3.3.4) will only hold for the highest priority traffic, while the lower priority traffic can see tail drops. To notify the sender of the tail drop, the node sets the `last-cell-dropped` field to 1 in the header of the next cell it sends to the sender. And finally, the rate limit feedback for a single traffic class in Eq 6 is updated for multiple traffic classes as follows—for a cell in traffic class c,

$$\textit{rate limit feedback}^{c}_{ji} = \sum_{p} [\textit{len}(Q^{p}_{jk}) + \textit{len}(R^{p}_{jk}) - 1]$$

$$\forall\ \textit{traffic class}\ p\ \ s.t.\ \textit{priority}(p) \succeq \textit{priority}(c)$$

D Recovering from cell corruption

Shoal uses the `CRC` field in the cell header to check for cell corruption. If node j receives a corrupted cell from node i, it first extracts the header fields corresponding to congestion control, namely the `rate limit feedback` and `last-cell-dropped`, and then discards the cell. It also signals the sender i that the cell was dropped by setting the `last-cell-dropped` field to 1 in the header of the next cell sent to node i. Node i then re-transmits the last cell it sent to node j. Since the `last-cell-dropped` field in the corrupted cell might also have been corrupted, node j takes the conservative approach of assuming the field was set to 1 and re-transmits the last cell it sent to node i. Further, if the `rate limit feedback` field also happens to be corrupted, the queue bound as described in §3.3.4 might get violated and there could be tail drops in the worst case. Shoal again uses the `last-cell-dropped` field in the cell header to notify the sender of any tail drop.