Shoal: A Lossless Network for High-density and Disaggregated Racks

Vishal Shrivastav*, Asaf Valadarsky†, Hitesh Ballani‡, Paolo Costa‡, Ki Suh Lee*,
Han Wang*, Rachit Agarwal*, Hakim Weatherspoon*

* Cornell University † The Hebrew University of Jerusalem ‡ Microsoft Research Cambridge

ABSTRACT

Rack-scale computers comprise hundreds of micro-servers connected to internal storage and memory through an internal network. However, their density and disaggregated nature pose a problem for existing packet-switched networks: they are too costly, draw too much power, and the network latency is too high for converged traffic (comprising IP, storage, and memory traffic). We propose Shoal, a rack-scale network that tightly integrates a circuit-switched physical fabric with the nodes' network stack to efficiently support converged traffic. Shoal's fabric comprises circuit switches with no buffers, no arbitration, and no packet inspection mechanism. Micro-servers transmit according to a static schedule such that there is no in-network contention. Shoal's congestion control leverages the physical fabric to achieve fairness, losslessness, and both bounded worst-case throughput and queuing. We use an FPGA-based prototype and simulations to illustrate Shoal's mechanisms are practical and achieve low latency within the rack at low cost and power.

1. INTRODUCTION

Rack-scale computers have the potential to replace servers as the basic unit of deployment in large-scale. They comprise hundreds of micro-servers, connected to storage and memory through an internal network, within a single enclosure. The micro-servers (or rack nodes) are built as single-die SoCs (systems-on-chip), which yields power and performance gains and allows many nodes to be packed into the same rack [11, 24, 30]. Several rack-scale computer designs have been proposed, including commercial products [22, 52, 53, 61, 69] and research platforms [6, 7, 12, 34, 38, 65]. Their density and disaggregated nature, however, pose the following challenges for the network.

First, a typical rack is expected to house 10× the nodes in today's racks. These nodes can be connected through off-the-shelf chassis switches with hundreds of ports [54, 63]. However, such a switch would occupy more than half a rack and consume more than a quarter of the rack's total power budget, not to mention its very high cost (§2). This has prompted custom designs for rack-scale networks, typically involving direct-connect topologies whereby each node is connected to a few other rack nodes. However, this requires many NICs per node, and routing traffic across multiple rack nodes hurts network throughput and latency (§7) and complicates routing and congestion control [11].

Second, unlike traditional servers where CPUs, memory, and storage are all packed onto a motherboard, resources inside a rack may be disaggregated and, hence, accessed across the network [16, 17]. This necessitates a "converged network" that, apart from IP traffic, also carries traffic to remote storage and non-volatile memory. The network thus needs to offer very low latency; around 100 μs for storage traffic and 10 μs for non-volatile memory [43]. Furthermore, storage and memory protocols rely on lossless delivery, and their performance falls sharply in the face of packet losses [48].

Today's network designs are unable to cope with these challenges due to their packet switched nature. Packet switching necessitates buffers and arbitration at each switch, which are major contributors to the switch's chip area [29]. In datacenters and WANs, this is a small price to pay for having an asynchronous, loosely coupled network whereby the network core can evolve independent of the servers' network stack. These properties are, however, not as critical for rack-scale networks as they can be synchronized and are co-designed with the rack nodes, while the drawbacks of packet switching are very relevant. Increased chip area and hence, high switch power and cost makes it hard to accommodate high density. Loose coupling makes it hard to achieve low latency. Indeed, recent proposals for low latency in datacenter networks [32, 48] rely on a tighter coupling between the network and servers through mechanisms like priority-based flow control (PFC).

In this paper, we argue that circuit switches are better suited to high-density, disaggregated racks because they have no buffers, no arbitration and no packet inspection mechanisms. As a result, they are an order of magnitude cheaper and power efficient than an equivalent packet switch (§6) and once configured, impose very little latency. We thus propose Shoal, a rack-scale network that tightly couples a circuit-switched physical fabric with the network stack at rack nodes to provide low latency connectivity at low cost. Using circuit switches, however, poses two challenges.

First, it requires that we schedule the fabric, i.e., configure the appropriate circuits. Unlike past solutions that use

a central entity to reconfigure circuits based on traffic patterns [10, 21, 37], Shoal reconfigures the fabric using a static schedule that connects each pair of nodes at an equal rate. To accommodate dynamic traffic patterns atop such 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 [44], obviates the complexity and latency associated with centralized schedulers while guaranteeing the worst-case throughput across *any* traffic pattern [9].

Such scheduling, however, requires that all nodes are connected through a single circuit switch. To achieve this, Shoal's fabric comprises many low port-count electrical circuit switches connected in a Clos topology. When reconfigured synchronously, the switches operate like a single rack-wide circuit switch. Furthermore, we decompose the static, equal-rate schedule for the fabric into static schedules for the constituent switches. The use of detouring, however, means that such scheduling imposes a throughput overhead. We compensate for this simply by using more (cheap) switches in our fabric and equipping nodes with higher bandwidth NICs while maintaining lower cost.

Second, achieving low latency across a circuit-switched fabric requires both fast switching and efficient congestion control. The fabric switching latency is dictated by the reconfiguration time of the circuit switches and by the granularity of time synchronization. We leverage recent advancements in electrical circuit switches that, unlike commercial optical circuit switches, offer very fast reconfigurability. For example, commercial circuit switch ASICs can be reconfigured in 2 ns [64] while our prototype FPGA-based circuit switch can be reconfigured in $<$6.4 ns. We also show that nanosecond-granularity time synchronization can be achieved at rack-scale using existing techniques (§3.4).

Congestion control atop Shoal's fabric is hard due to high multi-pathing—traffic between a pair of nodes is routed through all rack nodes. Shoal leverages the fact that the fabric directly connects each pairs of nodes at regular intervals to achieve an efficient realization of backpressure-based congestion control. Specifically, each source node receives explicit feedback about the queue size at intermediate nodes and computes the fair sending rates for its flows. The tight coupling of congestion control to the fabric allows us to achieve a bound on the queue size at each node. Rack nodes can thus be provisioned to ensure that, in the absence of failures, the network is *lossless*. Furthermore, our backpressure-based mechanism is amenable to hardware implementation.

Overall, this paper makes the following contributions:

- We present a rack-scale network architecture that couples circuit switches with the servers' network stack to provide low-latency connectivity at low cost and power.
- We design a fabric that uses low port-count circuit switches to offer the abstraction of a rack-wide circuit switch. We also scale the coordination-free scheduling technique to operate across the fabric.
- We devise a simple congestion control mechanism that leverages the equal rate connectivity provided by Shoal's fabric to achieve fair allocation, bounded queuing, and a lossless network.
- We demonstrate that our FPGA-based prototype can achieve low-latency switching. We also use it to show that our scheduling and congestion control mechanisms can be efficiently implemented in hardware.

Through a four-node FPGA-based testbed, we show that our implementation achieves both high throughput and bounded queuing. Using a cross-validated simulator, we show that these properties hold at scale too. Across datacenter-like workloads, Shoal improves the throughput and latency up to a factor of 2 and 7.5 respectively, as compared to a packet-switched network using TCP. To compensate for the throughput impact of detouring packets, Shoal equips nodes with 2$\times$ bandwidth, yet its resulting cost can be more than 50% lower than a packet-switched network.

2. MOTIVATION

The density of rack-scale computers, coupled with the disaggregation of processing, memory, and storage inside the rack, poses new challenges for their internal network. We begin by considering how conventional datacenter networks could be adapted for rack-scale computers and the shortcomings of such an approach.

Strawman designs. 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 [54], 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 [49].

We also considered a custom solution involving commodity switches. The current generation of top of rack switches support 64 ports at 50 Gbps [71]. Connecting 512 nodes in a rack-scale computer through a non-blocking network with 50 Gbps per node would require 24 such switches connected in a two-stage Clos topology. The power consumption of the network would be around 3 KW and the total network would cost $1.17 M (see §6).

Packet vs. circuit switching. The high power and cost of these strawman designs is rooted in the packet switched nature of the network. Packet switches need to inspect incoming packets, buffer them, and arbitrate traffic across ports. While numbers for commercial switches are hard to obtain, in network-on-chip (NoC) designs, the logic for these mechanisms can consume over 90% of the chip area and 33% of the total energy [29, 35].

On the positive front, packet switching allows the network core to be loosely coupled with the servers' network stack. Traditionally, this has been a good trade-off—the increased cost and power of switches is justified given that loose coupling has allowed the core network technologies to evolve independent of the servers' network stack. This also makes it easier for servers to work around in-network failures and allows the network to be asynchronous. These positives, however, do not hold up inside a rack. The physical size of a rack means that achieving rack-wide synchronization is feasible. Furthermore, a lot of the density and cost benefits of rack-scale computers come from the co-design of servers and the network, so independent evolution is not critical. Finally, since the rack is deployed and upgraded as a unit, the failure modes are different from traditional networks.

Instead, we argue that a circuit-switched network offers a different set of trade-offs that are more suited to rack-scale computers. Compared to an equivalent packet switch, today's circuit switches are 25× cheaper and draw 4× less power due to the lack of buffers and arbitration (§6). Thus, they can better accommodate higher density. Circuit switching does necessitate a tightly coupled network whereby all rack nodes need to be synchronized and all traffic needs to be explicitly scheduled. While these are hard challenges, we show that they are manageable in a rack-scale setting and the resulting benefits are significant. For example, instead of sharing the network through TCP-like mechanisms that rely on implicit signals like packet loss, it allows for explicit congestion control mechanisms that are more amenable to meeting stringent performance requirements.

Motivated by these observations, in this paper, we investigate the feasibility of a circuit-switched network for high-density and disaggregated racks that can be built using commercially available components.

3. DESIGN

Shoal is a rack-scale network architecture. 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 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 may make it hard to meet low-latency constraints.

Instead, Shoal uses *coordination-free* scheduling [9]. Specifically, each circuit switch forwards fixed-sized packets or "*cells*" between its ports based on a pre-defined "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 uniform 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).

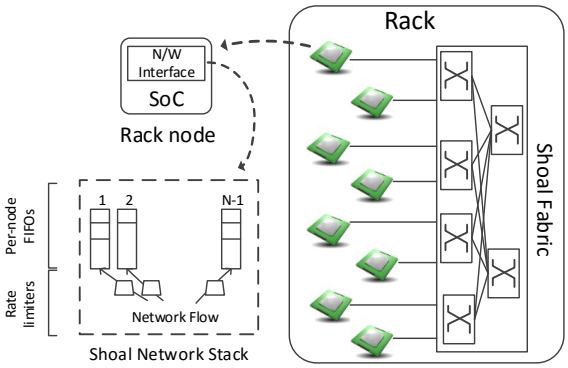


Figure 1: Shoal architecture: it comprises a network stack at rack nodes that are connected by the Shoal fabric. The fabric is constructed using low port-count circuit switches arranged in a Clos topology.

Node \ Time slot	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 2: Fabric schedule for a rack with 8 nodes.

The second mechanism implemented in Shoal's network stack is a congestion control technique that enables a lossless fabric through bounded-size queues (§3.3.2). Our main insight here is that the periodic connection of nodes in a rack by the fabric enables an efficient realization of backpressure-based congestion control. Specifically, one of the main challenges in implementing backpressure style mechanisms over multi-hop networks is instability for dynamic traffic patterns [23]. In Shoal, we restrict the backpressure mechanism to be a *single hop* only, avoiding the instability issue altogether while benefiting from the fact that such mechanisms allow for efficient hardware implementation.

In this paper, we focus on intra-rack connectivity. A key open question is the external connectivity of a Shoal rack. Existing rack designs typically use a few rack nodes as gateways which could also be adapted for Shoal. A more promising yet challenging option is to amend our design to explicitly accommodate multiple inter-connected Shoal racks. We leave an exploration of this avenue for future work.

3.2 Shoal fabric

Shoal uses a pre-defined, static schedule to reconfigure

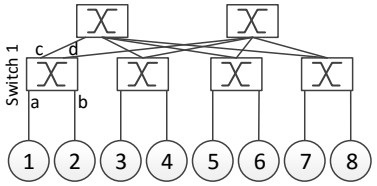


Figure 3: Circuit switches in a two-stage Clos topology.

Port \ Time slot	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. 3 for topology).

the fabric such that the rack nodes are connected at an equal rate. Fig. 2 shows an example schedule for a rack 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 with hundreds of high-bandwidth ports is intractable due to chip manufacturing 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 today's electrical circuit switches [64], with 64 ports at 50 Gbps, in such a topology allows us to connect a rack with 2048 nodes. Fig. 3 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 rack 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. 3. Fig. 2 shows the schedule for this fabric while Fig. 4 shows the schedule for one of the simple switches (switch 1). Notice that unlike the fabric's schedule, each simple switch's schedule does not provide equal rate connectivity between its ports. For e.g., on switch 1, ports a and b are never connected to each other, only to ports c and d; i.e., packets are never routed directly between ports a and b. Each switch's schedule is still 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 network stack spreads a node's traffic uniformly across the rack to ensure guaranteed network throughput, and implements a congestion control technique that ensures bounded network queuing. We explain these below.

3.3.1 *Data plane*

Rack nodes send and receive fixed-sized cells. Packets received from applications are thus fragmented into cells at the source node and reassembled at the destination. Each cell has a header 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 other 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 act of putting cells into the appropriate queue is what actually achieves the "switching" function. It also ensures traffic is forwarded 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. The low cost of our circuit switches makes this a good trade-off: as we show in §6, the cost of the resulting network is still estimated to be much lower than the cost of a traditional packet-switched network (with half the bandwidth for each node's uplink).

Second, when the fabric's schedule connects node i to node j, the former *always* transmits a cell; if the queue destined to node j is non-empty, the cell at the head of the queue is transmitted, else an empty cell is transmitted. This ensures that each node periodically receives a cell from every other rack node, which, in turn, enables implementing both an efficient backpressure-based congestion control (§3.3.2) and also a simple mechanism for detecting failures (§3.5).

3.3.2 *Congestion control*

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

High Multi-pathing. The periodic reconfiguration of Shoal's fabric means that the entire network can be seen as a complete mesh with virtual links between each pair of nodes. For example, consider a rack with 8 nodes whose schedule is shown in Fig. 2. 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 load balancing means that each nodes'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 [3, 46], as well as recently proposed protocols for RDMA networks [32, 48] only use a single path. Even multi-path extensions like MPTCP [40] target scenarios with few 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 fabric 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 [48], especially in multi-path scenarios. In Shoal, a *flow* refers to all 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 $N-1$ *flowlets*, 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 flowlets that are routed uniformly across a symmetric network topology, so we can achieve per-flow fairness by ensuring per-flowlet fairness. We thus treat each flowlet independently and aim to determine their fair sending rates.

Finally, each flowlet traverses two virtual links, either of which can be the bottleneck. For e.g., a flowlet $i \rightarrow j \rightarrow k$ can either be bottlenecked at the virtual link between nodes i and j, or between nodes j and k. We use the length of the queue $i \rightarrow j$, i.e., the queue at node i for cells destined to node j, 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 $i \rightarrow j$. It, however, also needs to obtain information about the size of the remote queue $j \rightarrow k$ that resides at node j.

Congestion control mechanism. We use a flowlet 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 flowlet 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 k into the cell header.

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, it knows the instantaneous length of its local queue to node j, $L(t)$, and a sample of the length of the remote queue between nodes j and k, $R(T)$. The max-min fair sending rate for a flowlet is governed by the most bottlenecked link on its path, i.e., the link with the maximum queuing. As a result, the next packet for this flowlet should only be sent after both the queues have had time to drain, i.e., at least, $max(L(T), R(T))$ epochs have passed since the feedback was received.

To achieve this, node i releases a cell for this flowlet into its local queue j only when the current length of the local queue, after accounting for the time since the last feedback, exceeds the size of the remote queue. Specifically, a cell is released at time t when the following condition holds:

$$L(t) + (t - T) \geq R(T) \quad (1)$$

Shoal's congestion control technique maintains the *invariant* that any flowlet has at most one cell each in both the local queue at its source and the remote queue at its intermediate hop. To explain how this is achieved, we consider the local queue first. A flowlet is allowed to release a new cell into the local queue only after the previous cell has been sent to the intermediate node and feedback about the remote queue has been received. This ensures that the local queue can have at most one cell for the flowlet. As for the remote queue, let us assume that the remote queue already had a cell belonging to the flowlet when the source received feedback about the queue size. Thus, the source will only transmit the next packet for this flowlet after the remote queue has been drained. While the remote queue may receive other cells in the meanwhile, the fact that it is serviced in a FIFO order ensures that the flowlet's previous cell is guaranteed to be sent to the destination before the source transmits the next cell towards the intermediate node.

When a new flow starts, there is no information about the remote queues for any of its flowlets. The flow is allowed to send one cell for each flowlet, thus a total of $N-1$ cells, before any rate limiting. However, as soon as all flowlets have sent one cell each, the sender gets the feedback about the queues at all intermediate nodes and hence, all flowlets are rate limited according to the mechanism described above.

Queue length guarantee. The invariant that each flowlet can have at most one cell queued at both the source and at the intermediate node ensures that Shoal's congestion control mechanism provides a guarantee for the maximum queuing at rack nodes. Consider the queue at node i towards node j. This queue can contain at most a cell for all flowlets and hence, all flows that originate at node i, and at most a cell for all flowlets and hence, all flows that have node j as their final destination. Node i can source at most $N-1$ flows while node j can be the destination for at most $N-2$ other flows (the flow from i to j is counted in the first set), so the maximum size of each queue is $(N-1)+(N-2) = 2N-3$.

This queue bound is very useful for several reasons: (i) It bounds the maximum buffer needed at each node's network interface in order to ensure that cells are never dropped and hence, the network is lossless. In a rack with 512 nodes and 256 B cells, each queue can grow to at most 260 KB. Since

each node has 511 queues, the maximum buffering needed at each node is 133.6 MB. Moreover, buffers can be very efficiently distributed between fast and slower memory, with only a small fraction of buffering needed in the fast memory. This is a consequence of our design where we access a buffer only once every epoch. In particular if the access latency of slower memory is less than an epoch, we only need to buffer $N-1$ cells in the fast memory, one cell per destination. Using the example above, this leads to a fast memory buffer size of just 128 KB. (ii) Next, it bounds the amount of bits required to encode the size of the queue in each cell's header; we only need $log(2N-3)$ bits. For a 512-node rack, this leads to 10 bits for the queue size feedback. (iii) Finally, it bounds the maximum size of the reassembly buffer needed when reassembling cells into packets.

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. We start by explaining these factors.

Circuit switch reconfiguration. Shoal uses electrical circuit switches that allow for rapid reconfiguration. Commercial circuit switches [64] can be reconfigured in 2 ns while our prototype implements an FPGA-based circuit switch that can be reconfigured in much less then 6.4 ns (§4.1).

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, with a signal 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 $\pm$ 5 cm, it ensures that propagation delay varies by a maximum of 0.5 ns. Small physical distance also mitigates the impact of temperature variations that could result in variable signal propagation delay.

Shoal leverages the WhiteRabbit synchronization technique [27, 31, 33] to achieve synchronization with bit-level precision. The main idea is to couple frequency synchronization with a time synchronization protocol like PTP [59] or DTP [28]. Frequency synchronization is achieved by distributing the clock from a rack node, designated as the clock master, to all other nodes and switches. The clock can be distributed explicitly, or implicitly through Synchronized Ethernet (SyncE) [62] whereby nodes derive a clock from the data they receive and use this clock for their transmissions.

Overall, the size of the guard band is the sum of the reconfiguration delay and the precision of the synchronization. Given the guard-band size, the operator can configure the length of each slot to balance the trade-off between network latency and throughput overhead: a smaller slot is better for network latency yet it imposes higher throughput overhead.

3.5 Practical concerns

We now discuss a few practical concerns resulting from Shoal's design.

Clock and data recovery (CDR). A key challenge for any network relying on fast circuit switches is that each node needs to be able to receive traffic from different senders at each timeslot. This requires that, at each timeslot, 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 [41]. However, we note that this is only a problem when using layer 0 circuit switches that operate at the raw physical layer, i.e., when a circuit is established between two ports, the ports are physically connected. Such a switch imposes no latency overhead but requires very fast CDR circuitry at the receiver [41] in order to achieve a reasonable guard band.

Today's circuit switch ASICs can also operate at layer 1 [64]. Shoal uses such layer 1 circuit switches—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.

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 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. While we omit the details for brevity, we can modify the fabric's schedule to satisfy this condition for all pair of rack nodes, as long as k is less than half the number of slots in an epoch.

Failures. Shoal needs to cope with failures of nodes, switches and the links connecting them. Tight integration of the nodes' network stack with the underlying fabric aids fault detection and diagnosis. Specifically, we use 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. When a node does not receive a cell from another node, it sends an alarm message to the other rack nodes. Given a set of alarms, a node can determine which node, switch or link has failed, although it may not always be possible to determine the ex-

	Logic utilization		Memory utilization
	ALMs	Pins	Block RAM (50MBits)
Shoal NIC	10%	10%	2%
Shoal Switch	7%	10%	1%

Table 1: Resource consumption for Stratix V FPGAs [70].

act root cause, especially if there are concurrent failures.

Once the failed component has been identified, the transmission schedule of the remaining nodes is updated. For example, when node i fails, all other nodes are told not to transmit at their slot to i and not to expect a cell at their slot from i. This allows the schedule of the circuit switches to be completely static but means that a failed node impacts the throughput of the remaining nodes. We study the impact of node failures on rack throughput in §7.3. This also makes it easier to handle the case in which the failed node is eventually replaced as it can just start re-using its original schedule.

If the failed node was the primary clock reference for synchronization, another node needs to take over as the reference and the remaining nodes need to switch to the new reference. This is already supported by the ITU standard for SyncE [62], which includes the ability of holding the current frequency and seamlessly locking to another available reference when failure is detected. If a link or a switch fails, some node(s) may be disconnected from the rest of the rack. However, this is also the case for today's racks.

4. IMPLEMENTATION

In this section, we discuss our FPGA-based implementation of the Shoal's switch and NIC. We used Bluespec System Verilog [51] (∼1,000 LOCs) and the total resource consumption, across Altera Stratix V FPGAs [70], is shown in Table 1. Note that most of the resource consumption comes from the PHY and the MAC IP blocks.

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. It forwards cells based on a static schedule, without performing any packet inspection or packet buffering. As shown in the bottom part of Fig. 5, the serial data is received through one of the ingress ports and it goes through the PHY block, which outputs a 72-bit vector. This vector is then forwarded to one of the egress ports. The mapping between the ingress and egress varies at every timeslot according to the static schedule. This is implemented using multiplexers whose control signal is driven by registers storing the clock counter and timeslot. Therefore, the switch reconfiguration latency corresponds to the time required to reconfigure the multiplexers, which is just a few logic gate delays and well below one FPGA clock cycle.

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.

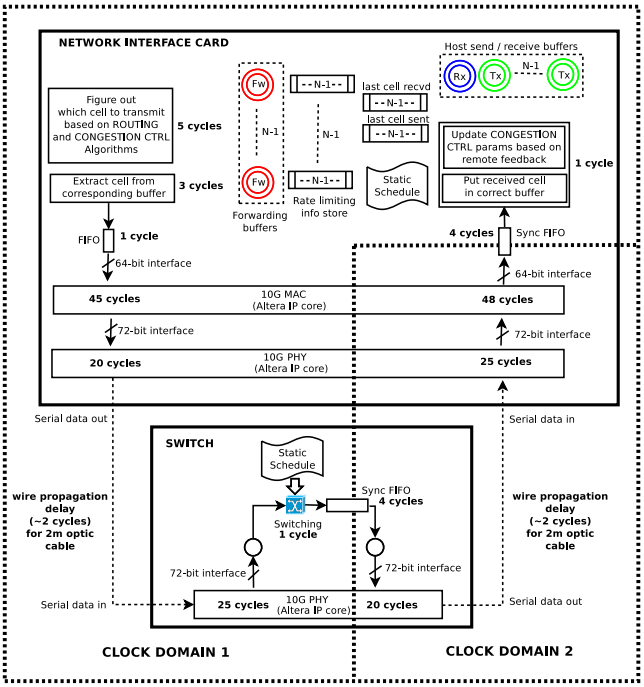


Figure 5: Shoal switch and NIC design with the latency of each block. A clock cycle is 6.4 ns.

This requires using a synchronization element, Sync FIFO, to move data safely across the two clock domains. In our implementation, Sync FIFO has a latency of four cycles.

Finally, in our implementation, we use the Altera IP 10G PHY blocks, which introduces an additional latency of 45 cycles (20 cycles on the transmit path and 25 cycles on the receive path respectively). In total, the end-to-end latency of our switch is 50 clock cycles. Since our FPGAs are clocked at 156.25 MHz, each clock cycle has a length of 6.4 ns, thus leading to an end-to-end latency of 320 ns. This, however, has no impact on throughput as the design is fully pipelined and, hence, it can operate at line rate as we will show in §5.

4.2 NIC design

Our implementation of the Shoal's NIC is shown in the top part of Fig. 5. We implemented the forwarding and the congestion-control logic according to the design described in § 3.3. Conceptually, Shoal maintains a single FIFO queue for each destination while a local queue, one per destination, (shown as *host send buffers* in Fig. 5) contains all locally generated cells. In practice, however, this conceptual design is expensive to implement as it requires copying a local cell from the local to FIFO queue. Instead, our implementation leaves all local cells in the local queue and stores a pointer to a scheduled local cell in a separate FIFO queue called *pointer queue* (not shown in Fig. 5). We have $N-1$ such queues, one per destination. By maintaining only pointers rather than copying the actual cells, we eliminate the overhead of a redundant memory-to-memory copy that can cause write contention. To maintain the design abstraction of *all*

cells being forwarded via a single FIFO queue per destination, along with the cell pointer, each entry of the *pointer queue* also includes the time at which the corresponding cell should be sent. At each time slot, if the entry at the top of the *pointer queue* for the destination indicated by the schedule has a time equal or lower than the current time, the corresponding cell is selected to be forwarded. Otherwise, the top cell from the forwarding queue is selected. If the latter contains no cells, then an empty cell is transmitted.

To implement the backpressure-based mechanism underpinning Shoal's congestion control (§3.3.2), each NIC also maintains two additional data structures. First, it uses an $(N-1)\times(N-1)$ matrix of registers (shown as *rate limiting info store* in Fig. 5) to keep track of the feedback received by each intermediate node j for destination k. The register (j,k) contains the number of epochs that NIC has to wait before it can send a cell for destination k to node j. Further, the NIC stores two vectors of size $N-1$ to record the last cell sent (received) to (from) each node.

When a cell is received, the NIC first checks its destination address. If this matches the local address, the cell is added to the *host receive buffer*. Otherwise, it is added to the forwarding FIFO queue corresponding to the final destination. If, instead, it is an empty cell, the payload is discarded. Next, the NIC extracts the rate limiting feedback from the header and updates the rate limiting value corresponding to the last cell sent to the remote node.

The NIC latency is dominated by the PHY and MAC IP blocks (in total 65 cycles on the transmit path and 73 cycles on receive) with the forwarding and congestion control logic only adding nine cycles on the transmit path and five cycles on the receive path. This illustrates that the additional NIC mechanisms introduced by Shoal impose low overhead.

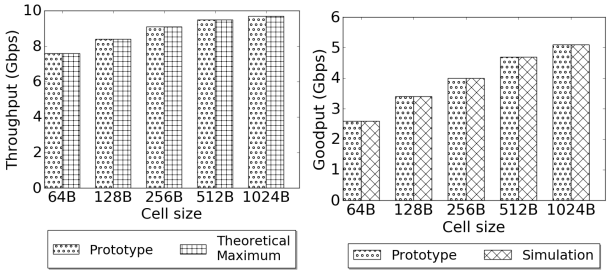
4.3 Slot size and guard band

Our implementation achieves frequency synchronization by distributing a global clock signal to all NICs and switches. It uses DTP [28] to synchronize clock counters. Together, this provides sub-nanosecond synchronization precision. As the switch reconfiguration latency is much smaller than the length of a clock cycle, and all wires are of same length, a guard band of one cycle (6.4 ns) is sufficient.

We select a slot size of 256 B, including 8 B for the cell header— source id (10 bits), destination id (10 bits), flow id (8 bits), sequence number (24 bits), rate limiting feedback (11 bits), and end-of-frame (1bit). Our FPGA transceivers operate at 10 Gbps, so the total slot length is 35 cycles (224 ns); 32 cycles (204.8 ns) for the 256 B cell plus three cycles (19.2 ns) for Ethernet overhead. The one-cycle guard band thus imposes a 3% overhead.

In our implementation, the total end-to-end latency, including both the NIC and the switch, is 206 cycles, which corresponds to approximately 6 time slots.

5. TESTBED EXPERIMENTS



(a) Average node throughput. (b) Average destination goodput.

Figure 6: Testbed throughput and goodput across a permutation workload ($N=4$) with varying cell sizes.

In this section, we evaluate our FPGA-based implementation through a small-scale prototype under a range of synthetic workloads. Each experiment ran for ten minutes and we measured the throughput observed at the destination and the maximum queue size per node. We repeated each experiment ten times and report the average value (error bars are omitted as the difference across runs is negligible).

Setup. Our testbed comprises three Terasaic DE5-Net boards [55], each equipped with an Altera Stratix V FPGA [70] and four 10 Gbps SFP+ transceiver modules. We use one FPGA board to implement a four-port circuit switch while the two remaining FPGAs are used to implement four NICs (two per FPGA) connected to the four-port switch with two-meter cables. This setup emulates a four-node rack. We connect all three FPGAs to a Dell T720 server and use the PCIe clock signal as the reference clock 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 were frequency synchronized.

Results. First, we assess the throughput that can be sustained by our implementation for varying cell sizes. We consider a permutation workload with $N=4$ flows, i.e., each node is the source and destination for exactly one flow, and measure the average throughput per node. The results in Fig. 6a show that our system always matches the maximum theoretical throughput. The throughput increases from 7.6 Gbps to 9.7 Gbps as the relative overhead due to Ethernet preamble and guard band decreases with larger cells.

Next, in Fig. 6b, we show performance in terms of destination *goodput*, measured as the amount of "useful" data (i.e., excluding the cells to forward and the empty ones) received by each node. Along with the testbed results, we also show the results obtained with a packet-level simulator that we developed. We use the simulator in §7 to evaluate the performance of Shoal at scale. Across all sizes, our testbed perfectly matches the performance predicted by the simulator. For the full permutation workload with four flows, the goodput is expected to converge to the theoretical value of $\sim$50% of the interface bandwidth. Interestingly, however, the goodput is significantly lower for small cell sizes, and it converges towards 50% goodput, 5 Gbps, only for large cell sizes (1,204 bytes). This is primarily an artifact of

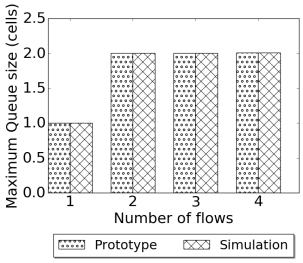


Figure 7: Maximum queue size for a permutation workload for different number of flows (testbed).

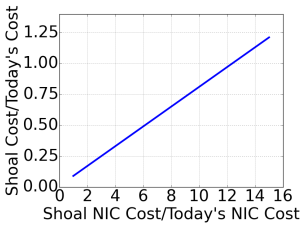


Figure 8: Shoal network would be 50% cheaper than PSN if the Shoal NIC costs six times today's NICs.

the small scale of our testbed, which causes the end-to-end transmission latency of a cell (six time slots in our prototype) to be higher than the epoch length (three time slots in our prototype). 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, like in our testbed, the end-to-end latency spans multiple epochs, the overall goodput suffers as senders cannot fully utilize their outgoing bandwidth. As the cell size increases, the ratio between the end-to-end latency and the epoch length decreases, and this explains why in our testbed the goodput improves with larger cells. In practice, however, even for modest-sized racks, this issue will not occur as the end-to-end latency will be much smaller than the epoch length ($N-1$ slots), and can be easily accommodated by modifying the schedule as discussed in §3.5.

Finally, in Fig. 7, we plot the queue size for a permutation workload with different number of flows. With only one flow in the system, the maximum queue occupancy is one cell. When more flows are added to the system, the queues grow up to a maximum of two cells. This is consistent with the queue length analysis in § 3.3.2 because each node is sourcing at most one flow and it is receiving at most one.

We also considered an incast workload (omitted for space), in which we randomly selected a different destination per run and varied the number of sources generating a flow to it. In all runs, the destination throughput was always at the peak. The maximum queue size, instead, grew linearly with the number of sources. Again, this matches our expectations as the maximum queue size is equal to the incast degree of the destination.

Overall, across all experiments, the testbed and simulations results are in agreement. This gives us confidence in our large-scale simulations (§7).

6. COST COMPARISON

We now compare the cost of a Shoal network to that of a packet-switched network (PSN) for a 512-node rack.

For a PSN, we consider the Mellanox Spectrum SN2700 switch, which supports 64 ports at 50 Gbps at a cost of $765 per switch port [71], 2 W per port, and $200 for the NIC [66]. Nodes have 50 Gbps links and connecting them using a non-oversubscribed Folded Clos topology requires 24 such switches. For Shoal, we consider a 64x50 Gbps crosspoint switch costing $30 per port and 0.5 W per port [30]. Since the switch will need additional circuitry to implement Shoal's schedule and synchronization primitives, we add a 50% overhead to the cost (i.e., $45 per switch port). Furthermore, to compensate for the throughput overhead of detouring packets, each node is equipped with 100 Gbps links. So the Shoal network has 48 switches. The total power of the Shoal network is 1.5 KW as compared to 3 KW for PSN.

Estimating the cost of the Shoal NIC is more challenging as it requires additional logic to implement forwarding and congestion control. Therefore, rather than setting on an arbitrary price, in Fig. 8 we show the relative cost of Shoal compared to a PSN when varying the ratio between the Shoal NIC and today's commodity NICs. It indicates that the Shoal network would be more than 50% cheaper as long as the Shoal NIC costs less than six times today's NICs and would provide cost savings as long as the Shoal NIC costs less than 12 times. As NICs with integrated programmable logic such as FPGAs (e.g., Mellanox Innova Flex 4 Lx EN [68]) and many-core architectures (e.g., Mellanox ConnectX-5 [67]) have recently appeared on the market at competitive prices, we believe that this is a realistic target.

7. SIMULATION EXPERIMENTS

We complement the testbed experiments presented in the previous section with simulations to investigate the scalability of Shoal. We first focus on static workloads (§7.2) and then use more realistic datacenter-like workloads (§7.3).

7.1 Simulation setup

We use the packet-level simulator that was cross-validated against our testbed prototype (§5). We simulate a 512-node rack, using the setup described in the previous section. Unless otherwise stated, we assume a cell size of 256 B, which includes the eight-byte cell header. For Shoal, we assume a guard band of 2.5 ns, based on a 2 ns reconfiguration latency for ASIC circuit switches in layer 1 mode [64] and 0.5 ns synchronization precision (§ 3.4). As explained in §3.5, with layer 1 switching, CDR is not required and, hence, it is not accounted for in the guard band. At 100 Gbps, the slot size for 256 B cells is 20.5 ns, so the guard band overhead is 11%.

7.2 Static workloads

We performed two sets of experiments with static workloads. First, we replicated the experiments with the permutation and incast workloads presented in §5 to verify that the behavior observed at small scale holds at large scale too. Second, we evaluated whether Shoal ensures max-min fairness among flows. We repeated each experiment ten times.

Permutation and incast workload. Fig. 9 shows the average destination goodput for the permutation workload. We do not plot error bars because the variance was negligible.

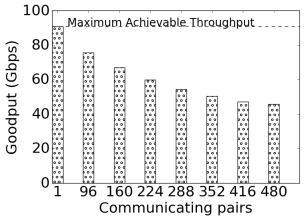


Figure 9: Average destination goodput (permutation workload).

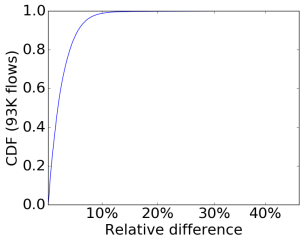


Figure 10: Relative difference between Shoal's flow goodput and ideal one.

The goodput values follow the expected trend. The goodput starts from the peak value and then it monotonically decreases until it halves. The maximum queue size across the entire range is two cells, which also matches expectations. The incast results are also as expected. The destination bandwidth is always fully saturated and the queue grows linearly with the incast degree up to a maximum of 511 cells.

These experiments confirm that the key properties exhibited at small scale (high goodput and bounded queues) are also maintained at large scale.

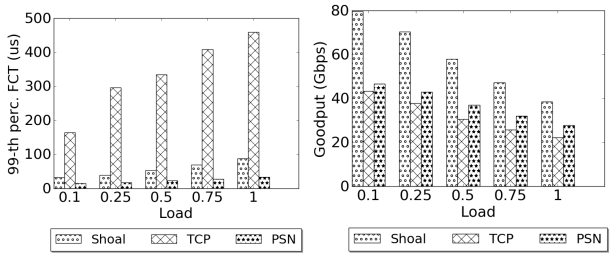
Fairness. To verify Shoal's fairness, we simulated several workloads comprising a variable number of flows from 50 to 1,024 with randomly selected sources and destinations. We compared the goodput achieved by each flow against its ideal goodput computed using the max-min water-filling algorithm [8].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

7.3 Datacenter workloads

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

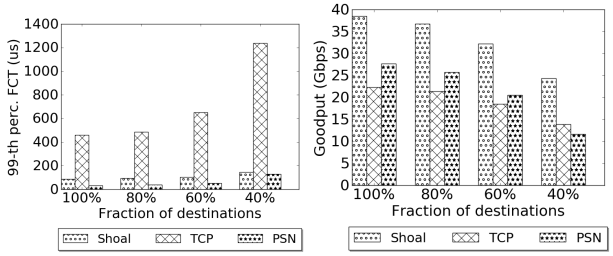
Setup. We generate a synthetic workload, modeled after published datacenter traces [3, 19]. Flow sizes are heavy tailed, drawn from a Pareto distribution with shape parameter 1.05 and mean 100 KB [4, 5]. Flows arrive according to a Poisson process and each simulation ends when one million flows have completed.

Baselines. We compare Shoal against the packet-switched network described in §6 that uses a Clos topology with full-bisection bandwidth. We use standard ECMP routing to forward packets at each hop and we use TCP for congestion control. We refer to this baseline as *TCP*. We also consider an idealized packet-switched network (*PSN*) that assumes perfect congestion control, i.e., it does not use TCP and ECMP but, instead, assumes per-flow hop-by-hop flow control and per-flow queuing at all switches and it uses packet spraying [14] to forward packets on all available paths. While this solution would be extremely costly to implement in practice, it is useful as it provides an upper bound on the performance achievable by *any* congestion control protocol in a packet-switched network.



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

Figure 11: Flow completion time (short flows) and average flow goodput (long flows) against traffic load.



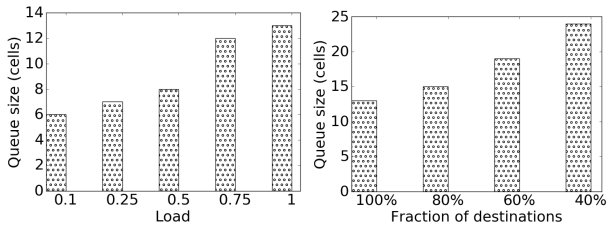
(a) Flow completion time. (b) Average flow goodput.

Figure 12: Flow completion time (short flows) and average flow goodput (long flows) against fraction of destinations.

Network load. In Fig. 11 we study the performance of Shoal and of the two baselines for different values of network load L. We define $L = \frac{F}{R \cdot N \cdot \tau}$ 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.

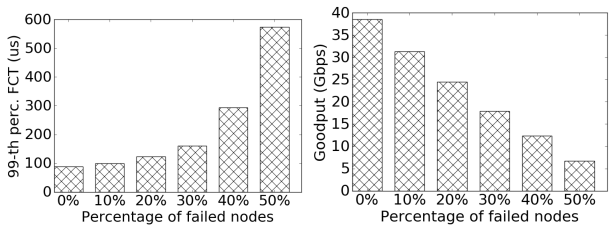
Fig. 11a shows the flow completion time (FCT) for short flows (size < 100 KB) while Fig. 11b the average flow goodput for long flows (size > 5 MB). Shoal significantly outperforms *TCP* for both the FCT (up to a factor of 7.5 at low load and 5.2 at high load) and goodput (up to a factor of 2 at low load and 1.7 at high load). The reason is twofold. First, TCP tends to fill the queues in the network, thus increasing the FCT for short flows. Second, ECMP can incur path collisions that degrade overall network throughput [1, 2]. To eliminate these two factors, we also plot the results of *PSN*. Shoal's FCT is within $2\times$ that of *PSN* yet it achieves a much higher throughput. The higher completion time is the price we pay for using statically-scheduled circuit switches instead of dynamically reconfigured, ideal packet switches. As we show later, Shoal's FCT improves significantly (and outperforms *PSN*) for smaller cell sizes. We also note that *PSN* represents an upper bound on the performance that can be achieved atop a packet-switched network. Shoal's increased goodput is because, to compensate for the detouring overhead, it uses 100 Gbps links (*PSN* uses 50 Gbps links). As explained in §6, however, despite the fact that Shoal has doubled the link bandwidth, it is still significantly cheaper than a packet-switched network.

To validate our claim that Shoal operates with very small



(a) Load. (b) Fraction of destinations.

Figure 13: Maximum queue size against load and fraction of destinations.



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

Figure 14: Flow completion time (short flows) and average flow goodput (long flows) against node failures.

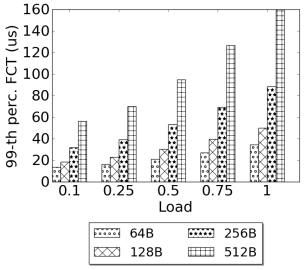


Figure 15: Flow completion time for different slot sizes.

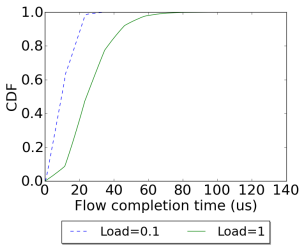


Figure 16: FCT for short flows (mean size is 32 KB).

queues, we plot the maximum queue size in Fig. 13a. Even at high load, Shoal yields a maximum queue size of 13 cells (3.25 KB) and a maximum aggregate queue per node of 571 cells (142.75 KB).

Skewness. Next, we want to study the impact of skewed workloads, which occur when some nodes are more popular than others. We model skewness by re-using the same workload of the previous experiments (we chose $L = 1$) but restricting the set of possible destinations from 100% to 40% of all nodes. For a constant load, a lower number of destinations implies a higher incast degree. This is particularly detrimental for *TCP* as it creates significant congestion at the switches. This explains why in Fig. 12a *TCP*'s FCT increases significantly when reducing the number of destinations. In contrast, Shoal maintains low FCT even in presence of very skewed workloads, while still maintaining high goodput (Fig. 12b). Again, this is due to its ability of keeping queues small as we show in Fig. 13b.

Node failures. We now focus our attention to the impact of failures in the Shoal network. 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 (Fig. 14b) 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. More interesting is the FCT behavior. Fig. 14a shows that for a moderate failure rate the increase in completion time is rather marginal, e.g., 1.38x for a 20% failure rate (resp. 1.81x for 30% rate). This is key because it indicates that, even with a simple approach, Shoal can maintain reasonably good performance in presence of failures, thus making it amenable even for sealed rack-scale deployments in which replacing failed nodes is not possible.

Slot size. In all experiments thus far, we have always used a slot size of 256 B. This size was selected in order to keep the guard band overhead low (11% in our setup). In the future, however, more mature synchronization technologies and circuit switches with lower reconfiguration latencies could make it possible to use a shorter guard band, which, in turn, would allow for smaller slot sizes. To understand its impact, we repeated our experiments, varying the slot sizes from 64 B to 512 B while always keeping the guard band relative overhead constant. The results in Fig. 15 show that the FCT is directly correlated with the slot size and smaller slots can even further reduce Shoal's FCT, e.g., up to 13.92 μs for $L = 0.1$ (resp. 34.38 μs for $L = 1$).

Direct-connect topologies. Now we compare Shoal against a rack-scale network architecture 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 [69]. As with the Shoal network, we assume an aggregate node bandwidth of 100 Gbps. We use R2C2 [11] for congestion control. We use the same workload as in the load experiments. For all values of load, Shoal consistently outperforms the rack-scale setup up to a factor of 14.9 for FCT (resp. a factor of 4.8 for goodput). 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 goodput. This does not occur in Shoal as packets only traverse one hop and the congestion control guarantees bounded queues.

Disaggregated workload. We conclude our experimental analysis by assessing Shoal's ability to support disaggregated workloads where flows of 10-25 KB are not uncommon [26] and some can be as small as 4 KB [17]. We conducted an experiment in which we reduced the mean flow size of our workload from 100 KB to 32 KB (cell size is 256 B). Since the flow size's distribution is heavy-tailed, this creates a workload in which 62% of flows have a size of 4 KB or lower. Fig. 16 shows the CDF for the FCT for short flows (size < 32 KB) for $L = 0.1$ and $L = 1$. At low load, the median value is 9.77 μs and the 99^{th}-perc. is 25.9 μs

(resp. 24.26 μs and 66.45 μs at high load), which indicates that Shoal is able to achieve a low FCT even for very short flows. Further, the moderate FCT increase between $L = 0.1$ and $L = 1$ demonstrates that the performance of short flows is not significantly affected by the increasing network load. This demonstrates Shoal's ability to carry different types of traffic with high performance.

8. RELATED WORK

Rack-scale computing promises significant cost, power and performance gains [50, 56, 60]. However, unlocking these gains requires numerous challenges to be solved.

Topology and technology. Several topologies and technologies are being investigated for rack-scale computers. 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 rack-scale computers. For example, both AMD SeaMicro [69] and HP Moonshot [57] use the 3D Torus topology and custom routing. Theia [45] proposes a new rack-scale topology. Motivated by the fact that the best direct-connect topology is workload-dependent, XFabric [30] combines circuit switches with SoC-based packet switches to allow the rack's topology to be reconfigured on the fly. New interconnect technologies like PCIe (eg., Huawei's DC3.0) [58] and Silicon Photonics (eg., Intel RSA [61], Firebox [6]) are also being explored.

While direct-connect topologies can provide high bisection bandwidth, they require that each SoC be equipped with many network interfaces and be able to sustain their cumulative bandwidth, not to mention the complexity of routing and congestion control [11] and topology reconfiguration [30]. As shown in §7.3, routing across multiple nodes also hurts throughout and latency. Overall, this paper shows that a purely circuit-switched rack is not only feasible, it can be simpler and more performant than a direct-connect fabric.

Circuit-switched networks. Circuit switching was used in ATM [13], telephony [15] and WAN networks [42] for long lasting circuits which amortizes the circuit set-up overhead. Recently, there has been extensive work on reconfigurable datacenter networks, using optical [10, 18, 21, 37, 47] and wireless technologies [20] that operate like a circuit switch. These solutions build on slower switching technologies (tens of μs or higher) and use a centralized controller for scheduling. To compensate for the additional latency due to these factors, they typically rely on a parallel packet switched network, which increases cost and complexity. Instead, Shoal caters to the rack-scale environment and uses low latency switching and coordination-free scheduling, thus removing the need of a centralized controller and a separate network to handle latency-sensitive traffic.

Low-latency congestion control. We considered recent proposals like Fastpass [36] and R2C2 [11] for congestion control atop our fabric. Fastpass, however, imposes significant communication and computation overhead as it uses a centralized entity to compute rate allocations. Computing max-min rates in high multi-path settings is challenging in itself, especially at low latency [39]. R2C2 also suffers from similar drawbacks, mostly because it targets general direct-connect topologies. It relies on broadcasting of flow events across the rack. Furthermore, it achieves computation tractability at the expense of network utilization. Finally, both Fastpass and R2C2 require accurate demand estimation. Instead, Shoal's congestion control leverages its fabric to achieve fair network sharing with bounded queues and low overhead and without requiring demand estimation.

There is already a trend towards tighter coupling between the network and servers for low latency congestion control in datacenters; for example, by using ECN as a feedback signal [3, 5]. Section 7.3 quantified the benefits of Shoal against an ideal transport protocol whose performance subsumes these techniques. Furthermore, they cannot achieve losslessness which is critical for storage workloads. The need for high throughput, low latency and losslessness has also prompted the use of RDMA atop converged Ethernet. Congestion control mechanisms [32, 48] specialized for these settings also rely on a closer coupling with the network. Shoal represents an extreme design point in this direction as the coupling of its congestion mechanism to its fabric achieves losslessness and fairness despite very high multi-pathing.

Load balanced switch. Load balanced switches [9, 25]. were proposed in the early 2000s as a way to obviate arbitration in monolithic, single-stage switches by uniformly detouring packets through an intermediate set of ports between the switch's inputs and outputs. Shoal's fabric operates like a load-balanced switch. However, instead of using an explicit intermediate stage, Shoal detours cells through other rack nodes. Furthermore, while the original technique focussed on monolithic switches, we scale it to a hierarchy of switches. Finally, we integrate the fabric with a novel congestion control mechanism.

9. SUMMARY

We presented Shoal, a network architecture for high density, 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 to achieve fairness and losslessness. 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, low latency and low queuing across diverse workloads.

10. ACKNOWLEDGEMENTS

This research is partially supported by DARPA CSSG (D11AP00266), NSF (1053757, 1440744, and 1422544), European Union's Horizon 2020 research and innovation programme under the SSICLOPS project (agreement No.

644866), and with gifts from Cisco, Altera and Bluespec.

11. REFERENCES

[1] M. Al-Fares, S. Radhakrishnan, B. Raghavan, N. Huang, and A. Vahdat. Hedera: Dynamic Flow Scheduling for Data Center Networks. In *NSDI*, 2010.

[2] M. Alizadeh, T. Edsall, S. Dharmapurikar, R. Vaidyanathan, K. Chu, A. Fingerhut, V. T. Lam, F. Matus, R. Pan, N. Yadav, and G. Varghese. CONGA: Distributed Congestion-aware Load Balancing for Datacenters. In *SIGCOMM*, 2014.

[3] 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.

[4] 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.

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

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

[7] 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.

[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] 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.

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

[12] A. Daglis, S. Novaković, E. Bugnion, B. Falsafi, and B. Grot. Manycore Network Interfaces for In-memory Rack-scale Computing. In *ISCA*, 2015.

[13] M. de Prycker. *Asynchronous transfer mode: solution for broadband ISDN*. Ellis Horwood, 1991.

[14] A. A. Dixit, P. Prakash, Y. C. Hu, and R. R. Kompella. On the Impact of Packet Spraying in Data Center Networks. In *INFOCOM*, 2013.

[15] L. Dryburgh and J. Hewett. *Signaling System No. 7 (SS7/C7): protocol, architecture, and services*. Cisco Press, 2005.

[16] P. Faraboschi, K. Keeton, T. Marsland, and D. Milojicic. Beyond Processor-centric Operating Systems. In *HotOS*, 2015.

[17] 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.

[18] 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.

[19] 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.

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

[21] 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.

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

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

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

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

[26] A. Klimovic, C. Kozyrakis, E. Thereksa, B. John, and S. Kumar. Flash storage disaggregation. In *Proceedings of the Eleventh European Conference on Computer Systems (EuroSys)*, page 29. ACM, Apr. 2016.

[27] 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 Symposium on Precision Clock Synchronization for Measurement Control and Communication*, 2011.

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

[29] 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.

[30] 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.

[31] 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.

[32] 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.

[33] 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.

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

[35] G. P. Nychis, C. Fallin, T. Moscibroda, O. Mutlu, and S. Seshan. On-chip Networks from a Networking Perspective: Congestion and Scalability in Many-core Interconnects. In *SIGCOMM*, 2012.

[36] J. Perry, A. Ousterhout, H. Balakrishnan, D. Shah, and H. Fugal. Fastpass: A Centralized "Zero-queue" Datacenter Network. In *SIGCOMM*, 2014.

[37] 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.

[38] 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.

[39] B. Radunović and J.-Y. L. Boudec. A Unified Framework for Max-min and Min-max Fairness with Applications. *IEEE/ACM Trans. Netw.*, 15(5), 2007.

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

[41] 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.

[42] Telecordia. Synchronous Optical Network (SONET) Transport Systems: Common Generic Criteria, 2009.

[43] A. Vahdat. Computing at a Crossroads. Keynote at Interop 2016.

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

[45] M. Walraed-Sullivan, J. Padhye, and D. A. Maltz. Theia: Simple and Cheap Networking for Ultra-Dense Data Centers. In *HotNets*, 2014.

[46] H. Wu, Z. Feng, C. Guo, and Y. Zhang. Ictcp: Incast congestion control for tcp in data center networks. In *CoNEXT*, 2010.

[47] 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.

[48] 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.

[49] AFCOM. Data center Standards. `http://bit.ly/1KPoZOZ`.

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

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

[52] Boston Viridis Data Sheet. `http://bit.ly/1fBnsQ9`.

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

[54] 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`.

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

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

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

[58] Huawei proposed DC 3.0 architecture of future data center. `pr.huawei.com/en/news/hw-423134-3.0.htm`.

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

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

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

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

[63] Juniper EX Series. `http://www.juniper.net/us/en/products-services/switching/ex-series/`.

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

[65] Maze: A Rack-scale Computer Emulation Platform. `http://aka.ms/maze`.

[66] Mellanox Blog. `http://bit.ly/2d000ia`.

[67] Mellanox ConnectX-5 Adapter. `http://bit.ly/2cuggGQ`.

[68] Mellanox Innova Flex 4 Lx EN Adapter. `http://bit.ly/2cDkkCk`.

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

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

[71] The Next Platform Blog. `http://bit.ly/2cZXrwo`.