XFabric: A Reconfigurable In-Rack Network for Rack-Scale Computers

Sergey Legtchenko, Nicholas Chen, Daniel Cletheroe, Antony Rowstron, Hugh Williams, and Xiaohan Zhao, *Microsoft Research*

https://www.usenix.org/conference/nsdi16/technical-sessions/presentation/legtchenko

This paper is included in the Proceedings of the 13th USENIX Symposium on Networked Systems Design and Implementation (NSDI '16).

March 16–18, 2016 • Santa Clara, CA, USA

ISBN 978-1-931971-29-4

Open access to the Proceedings of the 13th USENIX Symposium on Networked Systems Design and Implementation (NSDI '16) is sponsored by USENIX.

XFabric: A Reconfigurable In-Rack Network for Rack-Scale Computers

Sergey Legtchenko
Microsoft Research

Nicholas Chen
Microsoft Research

Daniel Cletheroe
Microsoft Research

Antony Rowstron
Microsoft Research

Hugh Williams
Microsoft Research

Xiaohan Zhao*
Microsoft Research

Abstract

Rack-scale computers are dense clusters with hundreds of micro-servers per rack. Designed for data center workloads, they can have significant power, cost and performance benefits over current racks. The rack network can be distributed, with small packet switches embedded on each processor as part of a system-on-chip (SoC) design. Ingress/egress traffic is forwarded by SoCs that have direct uplinks to the data center. Such fabrics are not fully provisioned and the chosen topology and uplink placement impacts performance for different workloads.

XFabric is a rack-scale network that reconfigures the topology and uplink placement using a circuit-switched physical layer over which SoCs perform packet switching. To satisfy tight power and space requirements in the rack, XFabric does not use a single large circuit switch, instead relying on a set of independent smaller circuit switches. This introduces partial reconfigurability, as some ports in the rack cannot be connected by a circuit. XFabric optimizes the physical topology and manages uplinks, efficiently coping with partial reconfigurability. It significantly outperforms static topologies and has a performance similar to fully reconfigurable fabrics. We demonstrate the benefits of XFabric using flow-based simulations and a prototype built with electrical crosspoint switch ASICs.

1 Introduction

There is a trend in large-scale data centers towards higher per-rack server density. While a typical compute rack today is composed of 40 to 50 blade servers interconnected through a Top of Rack (ToR) switch, hardware vendors increasingly propose energy-efficient, high density micro-servers, designed for data center workloads [17, 27, 35]. Rack-scale computers, such as AMD SeaMicro [44], HP Moonshot [1] and Boston Viridis [8] are up to rack-scale high density clusters of micro-servers with tight integration of the network, storage and compute. For example, the Boston Viridis supports hundreds of SoCs in one standard data center rack. Rack-scale computers are optimized for commodity data center workloads and have significant power, cost and performance benefits over traditional racks [4, 21, 22, 38] and attract increasing research interest [5, 6, 14, 16, 39, 42].

Higher server density requires a redesign of the in-rack network. A fully provisioned 40 Gbps network with 300 SoCs would require a ToR switch with 12 Tbps of bisection bandwidth within a rack enclosure which imposes power, cooling and physical space constraints. For example, the peak power draw (for power, compute, storage and networking) is limited by the power distribution used in data centers and the amount of heat that the in-rack cooling is able to dissipate and is around 9-16 kW for a typical high density rack today [3]. To address these challenges, some proposed designs replace a ToR switch by a "distributed fabric" where the packets forwarding is done by the servers. If the system uses SoCs then a small packet switch can be embedded on the server's SoC. For example, Boston Viridis uses the Calxeda EnergyCore SoC which has an embedded packet switch supporting eight 10 Gbps lanes. Each SoC is connected to a subset of SoCs in the rack, forming a multi-hop, bounded degree topology, e.g. mesh or torus. Each SoC forwards in-rack traffic from other SoCs and ingress/egress traffic is tunneled through a set of SoCs that have direct uplinks to the data center network.

Distributed fabrics are cost effective, but lack the flexibility of a fully provisioned network. Bisection bandwidth and end-to-end latency in the rack are a function of the network topology, and the best topology depends on the expected workload. Ingress/egress traffic is forwarded through multiple hops in the rack to an uplink, interfering with in-rack traffic. Lack of flexibility leads to suboptimal performance and complicates the design. For example, the HP Moonshot has three independent

* while on internship from UCSB.

networks with different topologies within the same enclosure: a *radial fabric* for ingress/egress traffic, and multi-hop *storage* and *2D torus* fabrics for in-rack traffic.

XFabric is a rack-scale network that maintains the benefits of a distributed fabric but allows workload-specific reconfigurability of the topology and uplinks. XFabric is organized as a *packet-switched* network running over a *physical circuit-switched* network that allows the physical topology of the fabric to be dynamically reconfigured. This could be achieved by using a single large circuit switch that would provide *full reconfigurability*, so *any* two SoC ports in the rack can be directly connected. However, XFabric needs to operate within the space and power limitations of the rack.

To achieve this, XFabric uses *partial reconfigurability*. It partitions the physical layer into a *set* of smaller independent circuit switches such that each SoC has a port attached to each partition. Packets can be routed between the partitions by the packet switches embedded in the SoCs. The partitioning significantly reduces the circuit switch port requirements enabling a single crosspoint switch ASIC to be used per partition. This makes XFabric deployable in a rack at reasonable cost.

However, the challenge is that the fabric is no longer fully reconfigurable, as SoC ports attached to different crosspoint switch ASICs cannot be connected directly. XFabric uses a novel topology generation algorithm that is optimized to generate a topology and determine which circuits should be established per partition. It also generates the appropriate forwarding tables for each SoC packet switch. The algorithm is efficient, and XFabric can instantiate topologies frequently, e.g. every second at a scale of hundreds of SoCs, if required. Additionally, it is able to place uplinks to the data center enabling them to be efficiently reconfigured.

XFabric uses insights from extensive work on reconfigurable data center-scale networks that enable dynamically reconfigurable network links between ToR switches [13, 20, 23, 24, 41, 47]. Similar to prior work, e.g. OSA [13] and FireFly [24], the topology is reconfigured at the physical layer, and network traffic is forwarded through multiple hops over the reconfigurable topology. XFabric differs in that it has been designed to operate at a rack-scale with SoCs that have embedded packet switches with multiple ports. It neither relies on wireless technology that cannot be used in the rack, nor requires a single large circuit switch. Designed for cost-effective in-rack deployments, XFabric sacrifices full reconfigurability for partial reconfigurability and demonstrates that this still provides good performance.

We have a prototype cluster, which uses 27 servers emulating SoCs and an XFabric network built with custom 32-port switches using low cost commodity crosspoint switch ASICs. We evaluate XFabric using this prototype and a flow-based simulator at larger scale. The results show that under realistic workload assumptions, the performance of XFabric is up to six times better than a static 3D-Torus topology at rack scale. We also show it provides comparable performance to a fully reconfigurable network while consuming five times less power.

The rest of the paper is organized as follows. Sec. 2 motivates our design and Sec. 3 provides an overview of XFabric. Sec. 4 details the algorithms used by the controller. Sec. 5 describes our current implementation of XFabric. Sec. 6 evaluates the performance of XFabric. Finally, Sec. 7 and 8 present related work and conclude.

2 Partial Reconfigurability

Reconfigurable networks have been traditionally proposed at data center scale [13, 20, 23, 24, 41, 47]. In these networks, each ToR has d reconfigurable ports and the set of d ToRs to which each ToR is directly connected is dynamically adapted to match the traffic demand. This has been implemented either with wireless (e.g. RF-based or free-space optics) [23, 24] or Optical Circuit Switches (OCS) [13, 20, 41, 47]. In the latter case, all d ports of all the n ToRs are connected to the same OCS that acts as a circuit switch with $n \times d$ ports: *any* port can be connected to *any* port of *any* ToR and the network topology is *fully reconfigurable*.

XFabric is focused on providing reconfigurability at the rack-scale, which has unique challenges because of the additional constraints due to physical space, power and cooling limitations. At the densities that can potentially be achieved using SoCs, the number of ports is high. If the switch functionality is distributed across the SoCs and a distributed network fabric is used, the number of ports required will be even higher. For example, a fully reconfigurable distributed fabric with 256 SoCs and 6 ports per SoC for in-rack communication requires 1,536 ports. This port count is too high to use a single crosspoint switch ASIC. It is possible to build a circuit switch implemented as a folded Clos network with multiple crosspoint switches, however, a folded Clos to support $n \times d$ ports requires $5 \times n \times d$ ports to be provisioned [13], which, for this example setup would require $5 \times 256 \times 6 = 7,680$ ports.

Fitting this in the rack can be challenging, but powering and cooling it will be hard. The per-port power draw ranges from 0.14 W for a typical optical circuit switch [11] to 0.28 W for a 10 Gbps electrical circuit switch [12]. The switches would consume between 1.3 to 2.6 kW, representing a significant fraction of the power provisioned for a high density rack today [3]. Given that as we increase density the compute and storage power requirements will also increase we need to manage power

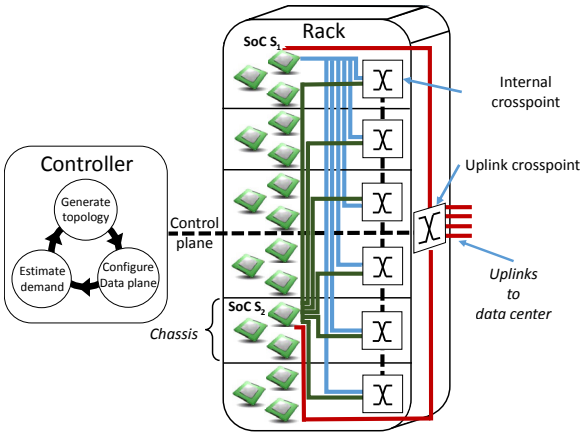


Figure 1: XFabric architecture, $d = 6$, $l = 4$ and $c = 6$.

resources carefully. A fully reconfigurable fabric is probably not acceptable.

In the rest of the paper, we refer to a circuit switch that is of a scale that can be implemented using a single crosspoint switch ASIC as simply a *crosspoint switch*, while one that is implemented as a folded Clos network of multiple crosspoint switches as a *Clos circuit switch*.

XFabric exploits the observation that full reconfigurability is not necessary. XFabric provides a *partially* reconfigurable fabric in which each SoC port can be connected to a *subset* of the SoC ports in the rack. XFabric has d independent physical networks each with a single circuit switch. Each SoC has a port attached to each of the d networks. This means that each SoC port can be connected to any other SoC. Hence, for 256 SoCs with 6 ports per SoC we would need 6 crosspoint switches each with 256 ports. Currently, commodity 160-port electrical crosspoint switches capable of switching 10 Gbps links are available [12]. We believe that a single crosspoint switch ASIC could be built to support approximately 350 ports. This is compared to requiring 7,680 ports for a full folded Clos circuit switch at this scale, requiring 22 crosspoint switch ASICs at 350 ports per ASIC.

Partial reconfigurability performs better in terms of cost and power. In terms of power, if we used a 256-port electrical crosspoint switch then for 256 SoCs with six 10 Gbps ports, XFabric would require 0.4 kW versus a fully reconfigurable fabric using a folded Clos circuit switch that would require 2.2 kW. In terms of cost, the per port cost is approximately $3, hence XFabric would have a cost of about $4.6K, while a fully reconfigurable fabric would cost $23K.

Partial reconfigurability limits the physical network topologies that can be instantiated, which potentially impacts performance. In the next section we describe XFabric in detail, and in Section 6 empirically show that the impact on performance is minimal.

3 XFabric Architecture Overview

The XFabric architecture combines a packet-switched layer 2 operating over a circuit-switched layer 1. It is assumed that each SoC has an embedded packet switch and exposes d ports for internal rack communication e.g. $d = 6$ [46]. Each of these ports is reconfigurable and is connected to a crosspoint switch called an *internal crosspoint*. Figure 1 shows the architecture, with $d = 6$. There are six internal crosspoints, and we highlight for SoCs S_1 and S_2, the links to each of the six internal crosspoints, which could be done on printed circuit board (PCB).

Each rack has l uplinks from the rack to the data center network to carry traffic for destinations outside the rack. The value of l is a function of the expected use for the rack; the Boston Viridis chassis has four 10 Gbps uplinks for 48 SoCs [8]. Each SoC has one *uplink port* to handle ingress/egress traffic to destinations outside the rack in addition to the d internal rack communication ports. Each SoC uplink port is connected to an *uplink crosspoint* which has a set of ports connected to the data center network. For example, in Figure 1 there are 4 uplinks. In operation, the SoCs that do not have their uplink port connected to one of the l external links tunnel their ingress/egress traffic to a SoC which is connected.

XFabric ensures that all d ports on each of the n SoCs are connected to other SoCs and all the l uplinks are connected to SoC uplink ports. We assume that each crosspoint has s ports, and that $s = n$. Due to the fact that a crosspoint with n SoCs connected can establish $n/2$ circuits, we assume that n is even. While our design is not fundamentally restricted to electrical circuit switching, this technology offers several benefits. Crosspoint ASICs are commodity low cost components [12, 50, 54] which are compact and have a low reconfiguration latency. For example, the M21605 crosspoint switch ASIC has 160 12.5 Gbps ports, is available in a 45 mm package and has a maximum reconfiguration latency of only 70 ns [12]. Uplink crosspoints connect SoCs' uplink ports directly to the data center network which requires the uplink crosspoint to support the PHY used outside the rack. The d internal ports can use the same or different PHY depending on the SoC implementation. It could be standard, e.g. backplane Ethernet [53] or proprietary [46], for example to reduce power consumption.

At layer 2, packets are forwarded by the SoCs over the instantiated physical topology using multi-hop routing. Packet switching operates independently from circuit switching, i.e. circuits are not established on a per-packet or per-flow basis. The physical topology reconfiguration is performed every interval t, where t is in the order of seconds. This removes the XFabric reconfiguration logic from the data path, simplifying the design and is motivated by the observation that circuits only

need to be reconfigured when the workload traffic pattern changes sufficiently to make reconfiguration beneficial. Layer 2 packet switching over layer 1 circuit switching forms the *data plane* of XFabric.

XFabric is managed by an in-rack controller that receives from each SoC estimates of its traffic demand to other SoCs and the uplink. Figure 1 shows the workflow of the controller. Periodically, it aggregates the information received from the SoCs into a rack-scale demand matrix and computes a new topology optimized for the demand. It then instantiates the topology in the data plane by establishing new circuits at the physical layer and updates the layer 2 forwarding tables. We assume that the packet switches on the SoCs support functionality to allow them to program their forwarding tables, e.g. OpenFlow [40]. The topology generation algorithm is lightweight and operates within the limitations imposed by the partially reconfigurable fabric, only producing topologies that can be instantiated by the network.

The SoC on which the controller executes needs to be connected to a micro-controller associated with each crosspoint ASIC through a *control plane* shown in dotted lines in Figure 1. Our current prototype supports Ethernet and USB control planes and we assume that a small fraction (e.g. 3) have their uplink ports connected to this network rather than an uplink crosspoint. The controller is designed to use only soft state and the reconfiguration process is resilient to the failure of the controller. If the controller fails then the network will be left in a consistent state and the controller can be started on another SoC which is connected to the control plane.

4 XFabric Configuration

XFabric needs to determine the mapping of the uplinks to SoCs and the internal fabric topology. The uplink mapping is performed first, because ingress/egress traffic induces load on the internal fabric while routed to the SoCs with external uplinks. Before describing the uplink mapping and internal topology generation algorithms, we describe how XFabric estimates the traffic demand.

4.1 Traffic Demand Estimation

For internal traffic, each SoC maintains a vector of length n and records the total number of bytes sent to each SoC in the rack. For external traffic, the SoC maintains two values, T_i and T_e, the total number of bytes sent and received, respectively. Periodically, this information is sent to the controller through the data plane and the counters are reset, and we call these *demand vectors*.

The controller maintains two vectors v_i and v_e of size n for ingress/egress traffic in which $v_i[S]$ is the number of bytes sent and $v_e[S]$ the number of bytes received by S during the interval. The controller aggregates the demand vectors into an $n \times n$ demand matrix, dm, such that $dm[S_1, S_2]$ represents a demand weight from S_1 to S_2, maintained using a weighted average.

4.2 Uplink Configuration Algorithm

The uplink configuration selects l SoCs that will be directly connected to the data center network and to which other SoCs need to tunnel their external traffic.

Conceptually, the controller partitions the n SoCs in the rack into l sets and places an uplink on one of the SoCs in each set. This SoC acts as a gateway to the data center network for the rest of the SoCs in the set. The controller aims to balance traffic between uplinks using the demand vectors v_i and v_e. Ideally, the aggregate external traffic demand is the same across all sets and for each set, the uplink is placed on the SoC with heaviest external traffic demand.

The placement algorithm operates in two stages. First, for each of the l uplinks, it selects the SoC S that has the highest demand $D_{ext}[S] = v_i[S] + v_e[S]$ and no uplink and places the uplink on S. In the second stage, the algorithm determines the sets of SoCs associated to each uplink. This is done by ordering SoCs without uplinks by their D_{ext} and iteratively assigning the SoC with highest demand to the set with the least aggregate demand. Ordering SoCs by demand ensures that SoCs with high demand will be fairly balanced across sets. Once all the SoCs have been assigned, source and destination SoCs for all external traffic are known. Based on this knowledge, the algorithm builds a traffic matrix dm_u in which $dm_u[S_1, S_2]$ is the ingress (and $dm_u[S_2, S_1]$ the egress) traffic demand between a SoC S_1 and its uplink placed on S_2. The algorithm then creates a matrix dm_{all} which is a sum of dm_{ext} and dm. This matrix is used by the topology generation algorithm to optimize the in-rack topology to both internal and external traffic.

4.3 Topology Generation Algorithm

This phase computes a topology optimized for dm_{all} by reducing the hop count between SoCs with high demand.

Forwarding high bandwidth traffic through multiple hops consumes bandwidth per link and incurs load on each SoC packet switch it traverses. Lower hop count thus results in lower link load and less resources spent on forwarding, improving network goodput [13]. For latency sensitive traffic, such as in-memory storage using RDMA [18], reducing the round trip time is important. A one hop latency of 1 microsecond versus a four hop latency of 4 microseconds is significant.

Conceptually, for each pair of SoCs in the rack, the algorithm assigns a weight based on their relative demand.

```
Input:
    socs ← SoC_list[n]
    dm_all ← demand_matrix
    port_map ← XbarToSoCPortMapping[c]
Output:
    PacketForwardingTables
    CircuitAssignment circuits[c]
1  topo ← Disconnected_Topology(socs)
2  SoC_pairs ← Order_By_Demand(socs, dm_all)
3  xbar_map ← To_Xbar(SoC_pairs, port_map)
4  while SoC_pairs ≠ ∅ do
5      partition_count ← 0
6      foreach soc in socs do
7          part[soc] = {soc}
8          partition_count ← partition_count + 1
9      foreach pair in SoC_pairs do
10         if part[pair.src] ≠ part[pair.dest] then
11             xbars ← xbar_map[pair]
12             xbar ← Best_Ranked(xbars, SoC_pairs)
13             xbars[xbar].Add_Circuit({pair.src, pair.dest})
14             topo.Add_Undirected_Edge(pair.src, pair.dest)
15             Merge(part[pair.src], part[pair.dest])
16             partition_count = partition_count − 1
17             foreach p in SoC_pairs do
18                 if xbar_map.Conflict(p, pair, xbar) then
19                     xbar_map[p].Remove(xbar)
20                     if xbar_map[p] = ∅ then
21                         xbar_map.Remove(p)
22                         SoC_pairs.Remove(p)
23                     else if p = pair then
24                         SoC_pairs.Reinsert(p, p.demand)
25             if partition_count = 1 then
26                 break
27 return {topo.ComputeForwardingTables(), circuits}
```

Algorithm 1: XFabric topology generation algorithm.

It then iteratively computes disjoint maximum weight spanning trees until all SoC ports in the rack have been assigned. The resulting topology is a union of maximum weight spanning trees and has three key properties. First, by construction it is fully connected, *i.e.,* there exists a path between each pair of SoCs. Second, it maximizes resource usage as all ports are assigned. Finally, as spanning trees are of maximum weight, SoC pairs with heavy traffic demand are satisfied in priority. A key challenge is to support partial reconfigurability and to do this within the constraints imposed by the physical topology.

Algorithm 1 describes the process in detail. The algorithm inputs are a list of SoCs and crosspoint ports to which each is attached (*socs* and *port_map*, which are initialized at boot time) and the demand matrix dm_{all}. It starts by initializing three data structures: $topo$, *SoC_pairs* and *xbar_map* (lines 1 to 3). The first is a fully disconnected topology in which each SoC in the rack is represented by a vertex and to which edges will be greedily added. We define the demand between a pair of SoCs $\{S_1, S_2\}$ as $D_{\{S_1,S_2\}} = dm_{all}[S_1][S_2] + dm_{all}[S_2][S_1]$ and *SoC_pairs* is a list of all pairs of SoCs that can be connected through each crosspoint, ordered by their demands in descending order (highest first). The last is a dictionary that associates each pair of SoCs to a set of crosspoints through which they can be connected. Initially, each pair of SoCs can be connected through any of the d crosspoints.

The main loop (lines 4 to 26) performs a sequence of spanning tree computations and stops when no more SoC pairs can be connected (line 4). The maximum weight computation is based on Kruskal's algorithm [32]: it starts with a set of partitions, one for each SoC (lines 5 to 8), and greedily reduces the number of partitions by connecting the two SoCs that are not in the same partition and have the highest demand (line 10). This results in two partitions being merged as their SoCs are no longer disconnected (lines 15-16). If only one partition remains, all SoCs are connected by a maximum weight spanning tree (line 25-26).

In order to connect a pair of SoCs, the algorithm selects one of the crosspoints through which the connection can be made (lines 11-12). Crossbar ports cannot be reused for multiple circuits simultaneously, therefore connecting a pair of SoCs $\{S_1, S_2\}$ through a crosspoint C implies that S_1 or S_2 can no longer be connected to other SoCs through C. It means that establishing a circuit in C negatively impacts its ability to satisfy remaining demand. In order to select a crosspoint in which connecting S_1 to S_2 has the least negative impact, a ranking between the crosspoints is performed (line 12). The ranking function computes the aggregate demand of all connections between S_1, S_2 to any other SoC that has a free port in C. This represents the demand that C would not be able to satisfy if $\{S_1, S_2\}$ was established, hence the crosspoint with the lowest value is selected. At that point, both the pair of SoCs and the crosspoint have been determined and the corresponding undirected edge and circuit are added (lines 13-14). Finally, the algorithm updates *SoC_pairs* and *xbar_map* (lines 17 to 24). It removes C from all the pairs in *xbar_map* that can no longer be connected through C (line 18-19). If the SoC pair can no longer be connected through any of the crosspoints, it is removed from *SoC_pairs* (lines 21 to 22). As two SoCs can be connected through multiple crosspoints at the same time, the pair that has just been connected is not removed from the *SoC_pairs* but reinserted after the last pair that has some unsatisfied demand (line 24). That way, if all pairs with demand have been connected, the algorithm can add secondary direct connections between high demand pairs, increasing the bandwidth.

The algorithm executes in polynomial time and once all circuits have been assigned, the topology is optimized for dm_{all}. The algorithm has the property that in addition to computing an optimized topology, it also finds the circuit assignment that instantiates that topology in the partially reconfigurable fabric. The result of the algorithm is a set of forwarding tables derived from the computed

topology and the circuit assignment that is merged with the uplink circuit assignment. The controller uses this information to reconfigure the data plane.

4.4 Reconfiguration

To instantiate a new topology, the XFabric controller needs to update the circuit switches (layer 1) and ensure all forwarding tables in each SoC are updated (layer 2). This cannot be achieved instantaneously, and can lead to instability during the update interval. The goal of reconfiguration is to minimize this window of instability.

We considered two general approaches. Inspired by SWAN [25], we experimented with incrementally changing the physical topology to ensure that packets can be successfully routed. This requires identifying a set of intermediate topologies, and then moving traffic off links that are to be reconfigured and then stepping through multiple different intermediate configurations. This approach leads to larger reconfiguration periods: the time taken to reconfigure is approximately constant and independent of the number of links being reconfigured, so migrating through x intermediate topologies takes x times the reconfiguration delay. Hence, we adopted the approach of performing a single reconfiguration.

Before triggering the reconfiguration, the controller sends new circuit assignments to every circuit switch through the control plane and new forwarding state to the SoCs through the data plane. Each circuit switch receives a `map` packet composed of a list of port mappings and a bitmap to indicate which ports need to be reconfigured. The micro-controller on the switch loads the circuit assignments into a set of registers on the crosspoint ASIC and acknowledges the controller, but does not instantiate the circuits. The physical topology must remain unchanged at this stage as the controller has no out-of-band mechanism to communicate with the SoCs. Each SoC receives its new forwarding tables together with the MAC address of the SoC that will be connected to each of its ports and a unique 64-bit version number for the configuration. This is efficiently encoded so the forwarding table, plus all the other information for a XFabric with 512 SoCs is less than 1 KB. Each SoC runs a process that receives and stores the update, but again does not reprogram any forwarding tables. Once all the SoCs have acknowledged the update information, all circuit switches and SoCs are ready for the reconfiguration.

The controller triggers the reconfiguration by transmitting a `reconfigure` packet to each circuit switch through the control plane. When received, the micro-controller on the circuit switch reprograms its crosspoint ASIC to the configuration specified in the `map`. At this point the physical network (layer 1) has been reconfigured, but the forwarding tables at the SoCs have not yet been updated. Once every circuit switch has acknowledged, the controller uses a simple flood-based mechanism to trigger the use of the new forwarding tables on each SoC. It sends on each of its ports a reconfiguration message which includes the new configuration version number. If a SoC with an old forwarding table receives the reconfiguration message, it starts using the new one and issues a reconfiguration message on all its ports. SoCs with new forwarding tables ignore reconfiguration messages. This process ensures rapid reconfiguration, in the worst case the number of rounds will be the diameter of the network.

It is essential that the data plane rapidly converges to a consistent state, even if the controller fails during the process. In particular, if the failure occurs after sending out a `reconfigure` to a subset of the circuit switches, the physical topology could be left in an inconsistent state. To address this, we ensure that each circuit switch that receives a `reconfigure` and has not yet reconfigured broadcasts the message through the control plane.

A failure of the controller before the broadcast of the reconfiguration in the data plane could lead to stale forwarding state at layer 2. To avoid this, we allow the SoCs to locally trigger the update of the forwarding tables. It does this by monitoring the local MAC addresses of SoCs attached to its ports: if a packet is received from a different MAC address than expected the SoC flushes the current forwarding table and uses the new one. After this local update, the SoC broadcasts a reconfiguration message, ensuring that the new forwarding state is propagated despite the failure of the controller.

During the reconfiguration of the switches any packets in flight at the switch can be corrupted or lost. However, thanks to the low switching latency of electrical crosspoint ASICs (see Section 3), we observe the packet loss to be low in practice (see Section 6) and rely on end-to-end transport protocols to recover from the packet loss.

5 XFabric Implementation

We have built a prototype XFabric platform, consisting of a set of seven electrical circuit switch units and 27 servers, each configured with eight 1 Gbps Ethernet NICs and a single Intel Xeon E5520 8-core 2.27GHz CPU running Windows Server 2008 R2 Enterprise. Each server emulates a SoC with an embedded packet switch that has six NIC ports for in-rack traffic. One NIC port is used as an uplink port and the last port is connected to a ToR switch for debug and experiment control.

Each circuit switch unit has 32 ports and each server is connected to all 7 circuit switches. Six serve as internal crosspoints and the last one is an uplink crosspoint with four uplinks ($l = 4$). The switches use the Analog Devices ADN4605 asynchronous fully non-blocking cross-

point switch ASIC [51]. Currently, they are connected to the servers using standard Ethernet cables, hence we need transceivers to convert the signal to and from the ASIC to 1000Base-T which is supported by the servers. This has significant cost and power overhead implications and is due to using standard servers instead of SoCs in the prototype platform. Each crosspoint ASIC is managed by an ARM Cortex-M3 micro-controller that configures it via an SPI serial bus and transceivers through I2C. The current design does not support 10 Gbps links, but we are in the process of designing a version with 160 10 Gbps ports using the Macom M21605 crosspoint ASIC [12]. In the experiments we use USB 1.1 to communicate with the control plane due to lack of spare Ethernet ports per server. Ethernet is supported by our switches and improves control plane latency by about an order of magnitude compared to USB.

The packet switch emulation is done in software, which allows us to understand the full functionality required before implementing it in hardware. The emulator uses two kernel drivers and a user-level process, and implements an OpenFlow-like API that provides access to the forwarding table, and callbacks on certain conditions. It binds to the six NIC ports used for internal traffic and the port used for the uplink. It also provides a virtualized NIC, to which an unmodified TCP/IP stack is bound to allow unmodified applications to be run on the testbed.

6 Evaluation

In this section we evaluate XFabric. First, we compare the performance of a reconfigurable fabric to static physical topologies. Then, we evaluate the efficiency of the algorithms used in XFabric. Finally, we show the benefits and overheads of XFabric dynamic reconfiguration.

In the experiments we evaluate XFabric using the prototype described in Section 5. In order to allow us to evaluate XFabric at scale we also use a simple flow-based simulator. We start by describing the fabric topologies, workloads and metrics used during the evaluation.

6.1 Topologies

We compare against three different topologies, two static and one dynamic. The first static topology is a 3D Torus (*3DTorus*). This topology has been widely used in HPC [15, 48] and has been proposed in data centers [2], in particular at rack scale [44]. It has the highest path length and lowest bisection bandwidth, but has a high disjoint path diversity and low cabling complexity.

The second one is a static random topology (*Random*). Random topologies have also been proposed for use in data centers [45] and are known to be "expander" graphs with high bisection bandwidth and low diameter.

We also use a dynamic reconfigurable network (*OSA*) inspired by OSA. In this network the topology is configured using the topology generation algorithm proposed in [13]. Our goal is to compare against the topology generation algorithm and we do not simulate additional features, such as the flexible link capacity described in [13]. Our implementation of the algorithm uses the same graph library as OSA [33]. This network uses a Clos circuit switch to which all the internal ports of all the SoCs are connected, so it is fully reconfigurable.

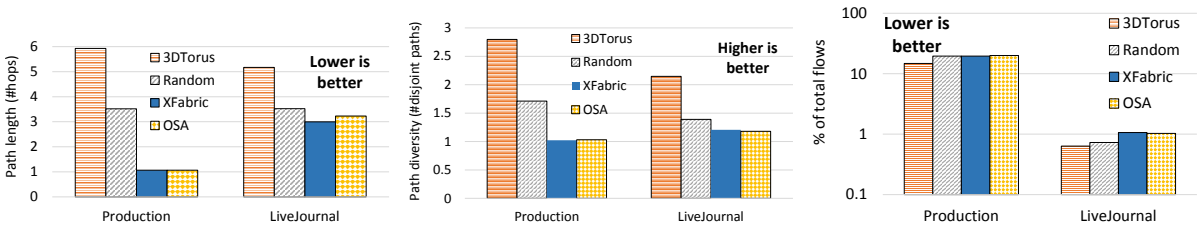
For the simulation results, unless otherwise stated, we assume the rack contains 343 SoCs each with $d = 6$ internal ports per SoC and one uplink port. In all cases we assume that the number of ports for the internal crosspoints is $s = n$ and for the uplink crosspoint $s = n + l$. We assume there are six internal crosspoints (as $d = 6$), and one uplink crosspoint with $l = 8$. Unless otherwise stated the uplinks are uniformly distributed across the SoCs and each SoC sends ingress/egress traffic to its nearest uplink in terms of path length.

We use 343 SoCs as it allows us to compare against a 3D Torus of size $7^3 = 343$. This is a challenge for XFabric, as a crosspoint with n ports establishes $n/2$ circuits so, if n is odd, one port cannot be connected to any other port on the same crosspoint. So, we will end up with 6 unused ports across all the SoCs. In practice, XFabric uses an even number of SoCs to avoid this issue. To handle the odd n configurations we form three pairs of crosspoints and in each pair statically connect a random SoC of one crosspoint to a different SoC in the second crosspoint. This means these static circuits are never reconfigured and results in strictly worse performance compared to allowing them to be reconfigured.

6.2 Workloads

We selected two workloads with well-identified traffic patterns, both based on real-world measurements.

Production cluster workload. This is a trace of 339 servers running a production workload in a mid-sized enterprise data center [7]. The data was collected over a period of 6.8 days and contains per-TCP-flow information including the source and destination IP address, the number of bytes transferred and a mapping from the servers' hostnames to the IP addresses of their NICs. We group flows based on source and destination hostnames. Flows in which the source or destination IP address does not correspond to a known hostname are considered as uplink traffic. The traffic is clustered, with heavy communication between servers with common hostname prefixes, and many-to-one traffic patterns: servers with a common hostname prefix often exchange traffic with a specific server with a different hostname prefix. This is consistent with the patterns described in [30].



(a) Path length (b) Average path diversity (c) Bottleneck link load

Figure 2: Performance summary of different fabrics for Production and LiveJournal workloads.

LiveJournal. Distributed platforms such as Pregel [34] or Tao [9] enable efficient processing of large graphs by partitioning the graph across a set of SoCs such that each SoC is assigned a set of vertices from the original graph. To generate a graph processing workload we use a trace collected from LiveJournal in December 2006 [36]. It includes 95.4% of users at that time, representing 5.2 million nodes and 48.7 million edges with an average edge degree of 18.7 edges per node. We shard the vertices into partitions using METIS, an offline graph partitioning algorithm [31] to uniformly partition the graph while minimizing traffic between partitions. There is one partition per SoC and we assume that the computation makes progress by message passing along the edges of the partitioned graph. The traffic is proportional to the number of the social graph edges between SoCs and is modeled as a constant bit rate over time.

For both workloads we map the workload onto the SoCs randomly. We also explored topology-aware workload placement using heuristic-based approaches for the static topologies [10]. For these we found that topology-aware placement always performs better than the random placement, but was always worse than the topology generated by XFabric. In practice, topology-aware placement is not easily feasible, and would often require migrating data between servers and is challenging to achieve dynamically. Due to lack of space, we only present results for random placement.

In the simulations each workload is mapped into a single traffic matrix *tm* such that for each pair of SoCs it stores the number of bytes sent and received between these SoCs. For the production trace, to scale beyond 339 SoCs, we augment the original trace by duplicating a random set of SoCs with non-zero traffic. For experiments with less than 339 SoCs, we subsample the trace by taking a random subset of the SoCs that have traffic.

6.3 Metrics

Across the experiments we use a number of metrics:

Path length. For each packet, we measure the path length from source to destination in number of hops. Since each hop adds a delay while forwarding traffic, this metric is a proxy for end-to-end latency.

Path diversity. This metric accounts for the fault tolerance of the topology, it measures the number of disjoint shortest paths that exist for each packet. Two paths are disjoint if they share no common link. Therefore, if there are k shortest paths for all the flows in the topology, $k-1$ links can fail without impacting the average path length of the traffic. Route diversity also improves traffic load balancing allowing traffic to be spread across disjoint shortest paths.

Bottleneck link load. The metric measures the congestion within the topology by measuring the link load on the most congested link in the topology.

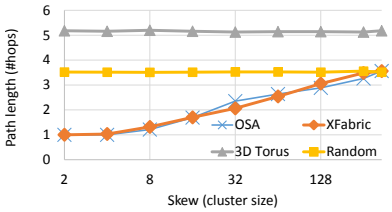
Fabric	# ports	Cost	Power draw
Clos Circuit Switch	10,290	$30.9K	2.9 kW
XFabric	2,058	$6.2K	0.6 kW

Table 1: Estimated cost & power, 343 SoCs, 6 ports/SoC.

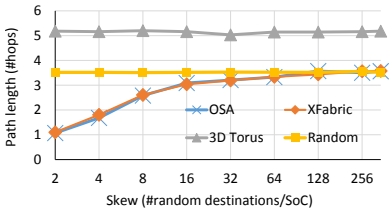
6.4 XFabric Performance

The two static topologies, 3D Torus and Random, do not require any additional hardware other than the switching functionality provided in the SoCs. Both the OSA and XFabric require additional ASICs to enable the reconfigurability. Any benefit obtained from being reconfigurable needs to be offset against the increased overheads this induces. Table 1 shows the number of ports required and the estimated cost and power consumption for XFabric and OSA assuming $3 per port and 0.28 W per port for 343 SoCs. OSA is a fully reconfigurable fabric supporting 2,058 ports, thus using a folded Clos. XFabric requires d crosspoint ASICs with n ports, connecting each SoC to each of the d crosspoints. This has a significant benefit in terms of cost and power.

This first simulation experiment evaluates the relative performance of the four topologies using the two workloads. For each configurable fabric we take the global demand matrix *tm* and optimize the network for *tm*. Fig-



(a) Clustered workload



(b) Random destinations workload

Figure 3: Impact of traffic skew on path length

ure 2(a) shows the average path length achieved by all the fabrics. Across both workloads the reconfigurable fabrics, XFabric and OSA, achieve shorter average path lengths than the static topologies. For Production, XFabric has an average path length of only 1.06 hops, which is 6 times less than 3D Torus and 3.7 times less than Random. For the LiveJournal workload, the path length for the reconfigurable fabrics is also lower but not by such a margin. As we will demonstrate later, the reason is due to the traffic skew. The LiveJournal workload has a lower skew. Comparing the performance of OSA and XFabric, we see for the Production workload that they both provide comparable performance. Notably for LiveJournal, even though OSA has a fully reconfigurable fabric it performs worse than XFabric. Having more flexibility in the fabric is insufficient, you also need an algorithm that can reconfigure the fabric to exploit the full flexibility.

Figure 2(b) shows the path diversity for all four fabrics with both workloads. These results show that the reconfigurable fabrics instantiate topologies with lower path diversity. The path length reduction benefits being shown in Figure 2(a) are achieved at the expense of reducing path diversity, shorter path lengths offer less opportunity for forwarding through different links. The 3D Torus has the highest path diversity, but also has the highest path length. This is an interesting trade-off where reconfigurability can provide benefit. Lowering path diversity can impact resilience to failure, and it also lowers the *aggregate* bandwidth available on the shortest paths between two SoCs. For reconfigurable fabrics, a link or SoC failure can be overcome by calculating a new topology that minimizes the impact of the failure. XFabric also can link multiple ports between the same SoCs, so providing multiple 1-hop links between two SoCs, and hence increase the aggregate bandwidth.

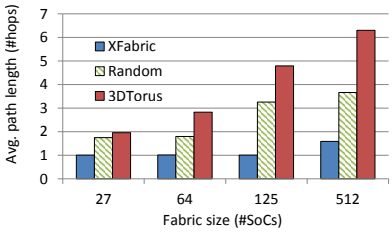
To understand this further, Figure 2(c) shows the number of flows that traverse each link. A flow from a to b is routed over the set of shortest paths in the topology between a and b and is registered on each link in the path. To achieve this each flow is split into f subflows of constant size, where f is much larger than the number of paths. The simulator estimates path congestion by counting the number of flows registered on the most loaded link in a path. To place a subflow, the simulator's transport layer checks if multiple shortest paths exist. If so two are randomly selected, and the simulator places the flow on the least congested one. This simulates traffic routing though multiple paths for any workload on top of any topology. Furthermore, the flow routing scheme ensures a good load balance of the traffic across links [37]. Figure 2(c) shows the percentage of flows that traverse the bottleneck link for each workload and topology. The most congested links on all topologies for both workloads have approximately the same load, except for the 3D Torus that benefits from its high path diversity.

The trade-off between path length and diversity also impacts the total network load across all links. The load imbalance across links is reduced when path diversity is high: in the 3D Torus the load is better balanced across links due to load balancing across multiple paths. However, because of the higher path length, the overall total load on links in the network is higher. The other topologies have a lower average total network link load than the 3D Torus, but a higher skew. However, XFabric aggressively reduces path length without significantly increasing load skew because optimization leads to links being shared across fewer source destination pairs.

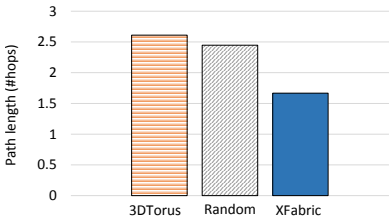
We now focus on the performance of XFabric with Production, our most realistic workload, to evaluate the performance of uplink placement. Figure 4(b) shows the path length of the ingress/egress traffic in the fabric before it reaches a gateway SoC with an attached uplink. It shows that XFabric efficiently places uplinks on SoCs with heavy ingress/egress traffic: the path length is on average reduced by 32% compared to the Random topology and 37% compared to the 3DTorus.

Finally, we vary the number of SoCs in the fabric to evaluate performance at different scales. Figure 4(a) shows the average path lengths for XFabric, Random and 3D Torus topologies when the number of SoCs is varied. In the worst case, for 512 SoCs, the average path length between SoCs in the rack is 1.6 hops only, showing that optimizing the topology at up to rack-scale is beneficial.

Reconfigurable fabrics perform better for workloads with high traffic skew. To understand this more we perform a parameter sweep across different traffic skews using two synthetic workloads. For the first, called *clustered*, we partition the SoCs into clusters and each SoC



(a) Path length vs. network size



(b) Uplink placement

Figure 4: Scalability and uplink placement performance.

communicates with all other SoCs in the same cluster. We vary the number of SoCs per cluster between 2 and 343. Intuitively, this results in a set of traffic matrices in which the traffic skew grows as cluster size drops. Figure 3(a) shows the path length as a function of the cluster size for XFabric and OSA with the clustered workload. The cluster size has no impact on static topologies because no reconfiguration is performed. When the skew is high, reconfigurable topologies are able to more efficiently optimize for the skew, up to the point when most of the traffic is sent through 1 hop. As the cluster size increases, the traffic pattern shifts to an all-to-all pattern and performance of reconfigurable fabrics becomes comparable to a Random topology. Notably, there is almost no difference between XFabric and OSA.

We run a second experiment with a different workload to evaluate the impact of the traffic pattern on path length. For this workload, called *random destinations*, each SoC sends traffic to a random set of k SoCs in the rack. For low values of k, the workload is very skewed and as it increases the workload progressively adopts an all-to-all traffic pattern. However, this results in a less clustered workload, even when traffic is very skewed. Figure 3(b) shows the path length as a function of the number of destinations per SoC for all fabrics. We observe the same trend as for the clustered workload, with both OSA and XFabric outperforming static topologies by up to a factor of 3.5 when the skew is high.

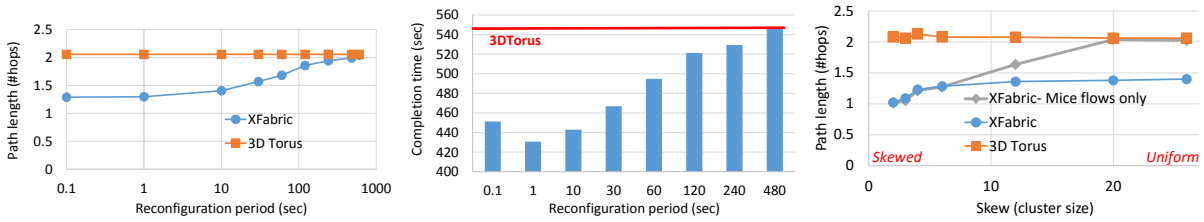
6.5 XFabric Prototype Performance

So far we evaluated the benefits of XFabric at scale using our simulator. In the next experiment we use our prototype platform to evaluate the dynamic reconfiguration performance of XFabric. Frequent XFabric reconfiguration is beneficial as it improves the responsiveness of the fabric to changes in traffic load, improving performance. However, too frequent reconfiguration induces overheads at the packet switching layer as it may result in packet loss. The reconfiguration of the crosspoints at layer 1 is not synchronized with layer 2. Too frequent packet loss can have a negative impact on the throughput at the transport layer, particularly if TCP is used.

We have created a test framework that uses unmodified TCP and replays flow-level traces derived from the Production workload. The framework opens a new socket for each flow and starts six flows per SoC concurrently, operating as a closed loop per SoC, so when one flow finishes the next is started on the SoC. In each experiment we configure the network as a 3D Torus and do not allow the network to reconfigure for the first 2 minutes. Unless otherwise stated the flow size is selected from the distribution of flow sizes in the Production workload, which is a typical heavy tailed distribution with a small number of elephant flows and a high number of mice flows, and an average flow size of 9.3 MB.

We first evaluate the impact of reconfiguration frequency on performance. We generate a trace with 250,000 flows and vary the reconfiguration period of XFabric between $t = 0.1$ to 480 seconds until the trace run is completed. Figure 5(a) shows the average path length of each packet as a function of the reconfiguration period. As expected, decreasing the reconfiguration period reduces the path length. When reconfigured every 30 seconds or less, XFabric achieves more than a 25% reduction in path length compared to the 3D Torus. The path length is reduced by approximately 37% (from 2.05 to 1.28 hops) for a reconfiguration interval of a second or less. This shows that even at small scale, reconfiguring the topology significantly reduces path length.

In order to understand how reconfiguring the fabric impacts goodput, Figure 5(b) shows the average completion time as a function of the reconfiguration interval. For each run we define completion time as the execution time from 2 minutes (when reconfiguration is enabled) to the end of the trace. The completion time for the 3D Torus is denoted by the red line and is constant as it does not reconfigure. We can see that for XFabric, shorter path length also reduces completion time, because each flow uses less network resources, increasing overall goodput. The completion time is reduced by 20% compared to the 3D Torus for a reconfiguration interval of 1 second. When reconfigured every 100 ms, the completion time increases compared to the 1 second interval, despite the path length being similar for both intervals. This shows the trade-off between the benefit of reconfiguration versus potential impact of packet loss on the transport layer.



(a) Path length (b) Completion time (c) Impact of workload skew on path length

Figure 5: Prototype performance.

We now explore the impact of traffic skew on performance. We set the reconfiguration period to 1 second and generate a set of traces in which SoCs are divided into clusters of fixed size c, with $c = 2$ to $c = 27$. Each trace has 250,000 flows and for each SoC, the destination of each flow is randomly selected in the corresponding cluster. Hence for $c = 27$ the traffic is uniform, and traffic skew increases as cluster size drops. Figure 5(c) shows the average path length as a function of the cluster size. As expected, for XFabric the path length is lower when the skew is high. In the extreme, when $c = 2$, the average path length is 1.02, which is more than a factor of 2 better compared to the 3D Torus. Notably, XFabric still has a 35% lower path length than the 3D Torus when the traffic is uniform. This is because many elephant flows live long enough to benefit from reconfiguration.

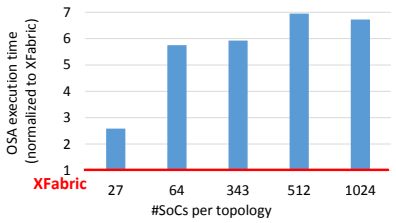
To quantify the impact of elephant flows, we generate a set of traces in which all flows are smaller than the median value from the Production flow size distribution. Each trace has 17 million flows with an average flow size of 129 KB and a maximum of 365 KB. SoCs are divided into clusters as previously but each SoC sends sequences of ten short flows to destinations in its cluster. Each SoC thus has a relatively stable flow rate, but per-destination traffic is bursty, which can be compared to real traffic patterns [43]. Figure 5(c) shows that when the traffic is skewed, XFabric is still able to accurately estimate the demand without elephant flows because each SoC has a limited number of destinations and the traffic pattern is predictable. However, as the traffic gets uniform, XFabric progressively loses the ability to accurately estimate the demand and the path length becomes comparable to the static topology.

6.6 Reconfiguration Overheads

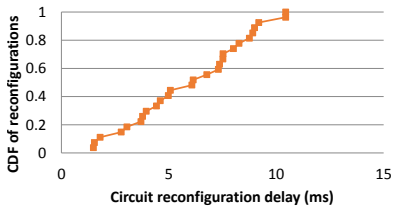
We now look at the overheads associated with XFabric reconfiguration. In the first experiment we calculate the average execution time across five runs for OSA and XFabric to generate a new topology for topology sizes ranging from 27 to 1024 SoCs. Figure 6(a) shows the time taken for OSA normalized to XFabric. In all cases XFabric significantly outperforms OSA. For 512 SoCs and below, XFabric generates topologies in less than 700ms, while for 1024 SoCs, the topology is generated in about 3 seconds. This shows that XFabric is able to optimize any rack-scale topology fast enough for dynamic reconfiguration in seconds or less. In comparison, it takes OSA about 20.5 seconds for the largest topologies, which is over 6 times longer.

The next experiment measures the end-to-end reconfiguration latency of XFabric. At the beginning, XFabric is configured as a 3D Torus and runs an all-to-all workload for 60 seconds to allow it to reach steady state. The controller then generates a new topology and reconfigures the data plane. Figure 6(b) shows the CDF of reconfiguration delays; each data point is the time taken for each server to have pushed a new forwarding table into the local packet switch from when the first crosspoint ASIC was reconfigured. Figure 6(b) shows that all servers are reconfigured within 11 ms. The latency for each micro-controller attached to the crosspoint ASIC to internally reconfigure it is approximately 40 microseconds. This delay is currently dominated by two factors, first the latency of the control plane interface which uses USB 1.1 and has a 1 ms delay, combined with the fact that the current prototype controller sequentially communicates with each of the micro-controllers, hence the last crosspoint ASIC is reconfigured 8 ms after the first. This latency would be removed if the controller used an Ethernet-based control plane.

We now measure the packet loss rate due to reconfiguration in the data plane. We run 5 experiments with the first workload described in Section 6.5 and a reconfiguration period set to 1 second. We count all Ethernet frames sent and received through each NIC on each SoC. Ethernet frames that are corrupted due to reconfiguration fail the CRC check and are dropped on the receiver before being counted. Hence the difference between the total number of frames sent and received by all SoCs accounts for the loss. We conservatively assume that all lost frames are due to reconfiguration. The average loss rate is 0.69 frames per full-duplex link per reconfiguration. The crosspoint ASICs we use have a switching time of



(a) Topology generation time



(b) Reconfiguration time

Figure 6: Reconfiguration overheads.

20 ns [51], which is the time to transmit 3 bytes at 1 Gbps and 25 bytes at 10 Gbps. With a minimal Ethernet frame size of 64 bytes [28], we expect the worst case loss on a full-duplex link to be 4 frames per reconfiguration for both 1 and 10 Gbps.

7 Related Work

The XFabric design is heavily influenced by the Calxeda SoC design, the first publicly available SoC that incorporates a packet switch. This SoC also explicitly provisioned ports for internal communication and a single Ethernet uplink port per SoC and we assumed this model for XFabric. We believe that this is a likely design point for other SoCs. Calxeda unfortunately collapsed, but we believe that other chip vendors will likely move in this direction. For example, the Xeon-D processor designed by Intel [52] is a low-power SoC with two 10 Gbps ports per SoC. Oracle recently announced their next-generation SPARC design with two 56 Gbps Infiniband controllers co-located with the CPU on silicon [26]. However, currently none of these designs yet supports embedded packet switches.

Optical Circuit Switching (OCS) has been proposed to establish physical circuits between ToR switches at the data center scale [20, 47]. They rely on MEMS-based switches and have high reconfiguration latency. To address latency sensitive traffic, c-Through and Helios rely on a separate packet switched network. Mordia [41] routes latency sensitive traffic through circuits by time-sharing the circuits between servers in a rack. XFabric differs from these architectures because they do not route packets over multiple circuits when a direct circuit is not available. The closest to our proposal is OSA [13] that allows multi-hop data forwarding between ToRs (servers in a rack use traditional packet switching). However, OSA does not address the issue of scaling beyond a single circuit switch and assumes all ToRs have all reconfigurable ports connected to the same switch. Compared to OSA, XFabric addresses a set of challenges unique to the rack scale. It reduces the space and power consumed by the reconfigurable fabric by using several smaller cross-point ASICs and deals with uplink management.

Halperin *et al.* [23] propose to augment standard data center networks with wireless flyways used to decongest traffic hotspots. Zhou *et al.* [49] improve the technique by bouncing the signal off the data center ceiling to overcome physical obstacles. However, the wireless technology has a set of physical constraints (e.g. signal interference) due to which only a subset of the links are reconfigurable while the rest of the traffic is still routed through the traditional network. FireFly [24] is a data center level architecture in which the physical layer is supported by lasers reflected using large ceiling mirrors. However, this technique is hard to leverage inside a rack.

In HPC, Kamil et al. uses an optical circuit switch to interconnect packet switches, which can then be pooled to increase available bandwidth between heavily communicating servers [29]. This work differs from XFabric as it considers one large circuit switch for all packet switches and leverages the high predictability of HPC workloads to compute efficient topologies.

AN3 [19] performs virtual circuit switching and allows speculative circuit establishment supported by custom switches implemented in FPGA. This system differs from our work as it establishes circuits at layer 2 of the network and operates over a static physical topology.

8 Conclusion

Emerging hardware trends and server densities are going to challenge the usual approach of connecting all the servers in a rack to a single ToR switch. One explored solution is to disaggregate the packet-switching functionality across SoCs. Based on the observation that different network topologies support different workloads we propose XFabric, a dynamically reconfigurable rack-scale fabric. It differs from prior work by addressing specific requirements that arise at rack scale, dealing with power and space constraints and managing uplink to the data center network. A prototype XFabric implementation demonstrates the reconfiguration benefits and shows that partial reconfigurability achieves the performance of full reconfigurability at lower cost and power consumption.

Acknowledgments

We are grateful to the anonymous reviewers, and in particular to our shepherd S. Keshav, for their feedback.

References

[1] HP Moonshot System: The World's First Software-Defined Server -Family guide, Jan. 2014.

[2] ABU-LIBDEH, H., COSTA, P., ROWSTRON, A., O'SHEA, G., AND DONNELLY, A. Symbiotic Routing in Future Data Centers. *ACM SIGCOMM Computer Communication Review 41*, 4 (2011), 51–62.

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

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

[5] ASANOVIC, K., AND PATTERSON, D. FireBox: A Hardware Building Block for 2020 Warehouse-Scale Computers. In *USENIX FAST* (2014).

[6] BALAKRISHNAN, S., BLACK, R., DONNELLY, A., ENGLAND, P., GLASS, A., HARPER, D., LEGTCHENKO, S., OGUS, A., PETERSON, E., AND ROWSTRON, A. Pelican: A Building Block for Exascale Cold Data Storage. In *Proceedings of the 11th USENIX conference on Operating Systems Design and Implementation* (2014), USENIX Association, pp. 351–365.

[7] BALLANI, H., JANG, K., KARAGIANNIS, T., KIM, C., GUNAWARDENA, D., AND O'SHEA, G. Chatty Tenants and the Cloud Network Sharing Problem. In *Proceedings of the 10th USENIX conference on Networked Systems Design and Implementation* (2013), USENIX Association, pp. 171–184.

[8] BOSTON. Boston Viridis Data Sheet. http://download.boston.co.uk/downloads/9/3/2/932c4ecb-692a-47a9-937d-a94bd0f3df1b/viridis.pdf.

[9] BRONSON, N., AMSDEN, Z., CABRERA, G., CHAKKA, P., DIMOV, P., DING, H., FERRIS, J., GIARDULLO, A., KULKARNI, S., LI, H., ET AL. Tao: Facebooks Distributed Data Store for the Social Graph. In *USENIX ATC* (2013).

[10] BURKARD, R., PARDALOS, P., AND PITSOULIS, L. The Quadratic Assignment Problem. In *Handbook of Combinatorial Optimization* (1998), Kluwer Academic Publishers, pp. 241–338.

[11] Calient S320 Optical Circuit Switch Datasheet. http://www.calient.net/download/s320-optical-circuit-switch-datasheet/.

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

[13] CHEN, K., SINGLA, A., SINGH, A., RAMACHANDRAN, K., XU, L., ZHANG, Y., WEN, X., AND CHEN, Y. OSA: An Optical Switching Architecture for Data Center Networks with Unprecedented Flexibility. *IEEE/ACM Transactions on Networking (TON) 22*, 2 (2014), 498–511.

[14] COSTA, P., BALLANI, H., RAZAVI, K., AND KASH, I. R2C2: A Network Stack for Rack-Scale Computers. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication* (2015), ACM, pp. 551–564.

[15] CRAY. CRAY XT3 Datasheet. http://www.craysupercomputers.com/downloads/CrayXT3/CrayXT3_Datasheet.pdf.

[16] DAGLIS, A., NOVAKOVIC, S., BUGNION, E., FALSAFI, B., AND GROT, B. Manycore Network Interfaces for In-Memory Rack-Scale Computing. In *Proceecidings of the 42nd International Symposium in Computer Architecture* (2015), no. EPFL-CONF-207612.

[17] Dell PowerEdge c5220 Microserver. http://www.dell.com/us/business/p/poweredge-c5220/pd.

[18] DRAGOJEVIĆ, A., NARAYANAN, D., HODSON, O., AND CASTRO, M. FARM: Fast Remote Memory. In *Proceedings of the 11th USENIX Conference on Networked Systems Design and Implementation, NSDI* (2014), vol. 14.

[19] ERIC CHUNG, ANDREAS NOWATZYK, TOM RODEHEFFER, CHUCK THACKER, AND FANG YU. AN3: A Low-Cost, Circuit-Switched Datacenter Network. Tech. Rep. MSR-TR-2014-35, March 2014.

[20] FARRINGTON, N., PORTER, G., RADHAKRISHNAN, S., BAZZAZ, H. H., SUBRAMANYA, V., FAINMAN, Y., PAPEN, G., AND VAHDAT, A. Helios: a Hybrid Electrical/Optical Switch Architecture for Modular Data Centers. *ACM SIGCOMM Computer Communication Review 41*, 4 (2011), 339–350.

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

[22] Google Ramps Up Chip Design. http://ubm.io/1iQooNe.

[23] HALPERIN, D., KANDULA, S., PADHYE, J., BAHL, P., AND WETHERALL, D. Augmenting Data Center Networks with Multi-Gigabit Wireless Links. In *ACM SIGCOMM Computer Communication Review* (2011), vol. 41, ACM, pp. 38–49.

[24] HAMEDAZIMI, N., QAZI, Z., GUPTA, H., SEKAR, V., DAS, S. R., LONGTIN, J. P., SHAH, H., AND TANWER, A. FireFly: a Reconfigurable Wireless Data Center Fabric using Free-Space Optics. In *Proceedings of the 2014 ACM conference on SIGCOMM* (2014), ACM, pp. 319–330.

[25] HONG, C.-Y., KANDULA, S., MAHAJAN, R., ZHANG, M., GILL, V., NANDURI, M., AND WATTENHOFER, R. Achieving High Utilization with Software-Driven WAN. In *ACM SIGCOMM Computer Communication Review* (2013), vol. 43, ACM, pp. 15–26.

[26] Oracles Sonoma Processor. `http://www.hotchips.org/archives/2010s/hc27/`.

[27] HP ProLiant m800 Server Cartridge. `http://bit.ly/1JxM9Zr`.

[28] IEEE. 802.3-2012 IEEE Standard for Ethernet. `http://standards.ieee.org/findstds/standard/802.3-2012.html`.

[29] KAMIL, S., PINAR, A., GUNTER, D., LIJEWSKI, M., OLIKER, L., AND SHALF, J. Reconfigurable Hybrid Interconnection for Static and Dynamic Scientific Applications. In *Proceedings of the 4th international conference on Computing frontiers* (2007), ACM, pp. 183–194.

[30] KANDULA, S., SENGUPTA, S., GREENBERG, A., PATEL, P., AND CHAIKEN, R. The Nature of Data Center Traffic: Measurements & Analysis. In *Proceedings of the 9th ACM SIGCOMM conference on Internet measurement conference* (2009), ACM, pp. 202–208.

[31] KARYPIS, G., AND KUMAR, V. Multilevel Algorithms for Multi-Constraint Graph Partitioning. In *Supercomputing* (1998).

[32] KRUSKAL, J. B. On the Shortest Spanning Subtree of a Graph and the Traveling Salesman Problem. *Proceedings of the American Mathematical society 7*, 1 (1956), 48–50.

[33] LEMON Graph Library. `http://lemon.cs.elte.hu/trac/lemon`.

[34] MALEWICZ, G., AUSTERN, M. H., BIK, A. J., DEHNERT, J. C., HORN, I., LEISER, N., AND CZAJKOWSKI, G. Pregel: a System for Large-Scale Graph Processing. In *SIGMOD* (2010).

[35] Microservers Powered by Intel. `http://www.intel.com/content/www/us/en/servers/microservers.html`.

[36] MISLOVE, A., MARCON, M., GUMMADI, K. P., DRUSCHEL, P., AND BHATTACHARJEE, B. Measurement and Analysis of Online Social Networks. In *IMC* (2007).

[37] MITZENMACHER, M. The Power of Two Choices in Randomized Load Balancing. *Parallel and Distributed Systems, IEEE Transactions on 12*, 10 (2001).

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

[39] NOVAKOVIC, S., DAGLIS, A., BUGNION, E., FALSAFI, B., AND GROT, B. Scale-Out NUMA. *ACM SIGARCH Computer Architecture News 42*, 1 (2014), 3–18.

[40] OpenFlow Specification. `http://archive.openflow.org/documents/openflow-spec-v1.1.0.pdf`.

[41] PORTER, G., STRONG, R. D., FARRINGTON, N., FORENCICH, A., SUN, P., ROSING, T., FAINMAN, Y., PAPEN, G., AND VAHDAT, A. Integrating Microsecond Circuit Switching into the Data Center. In *ACM SIGCOMM 2013 Conference, SIGCOMM'13, Hong Kong, China, August 12-16, 2013* (2013), D. M. Chiu, J. Wang, P. Barford, and S. Seshan, Eds., ACM, pp. 447–458.

[42] PUTNAM, A., CAULFIELD, A. M., CHUNG, E. S., CHIOU, D., CONSTANTINIDES, K., DEMME, J., ESMAEILZADEH, H., FOWERS, J., GOPAL, G. P., GRAY, J., ET AL. A Reconfigurable Fabric for Accelerating Large-Scale Data Center Services. In *Computer Architecture (ISCA), 2014 ACM/IEEE 41st International Symposium on* (2014), IEEE, pp. 13–24.

[43] ROY, A., ZENG, H., BAGGA, J., PORTER, G., AND SNOEREN, A. C. Inside the Social Network's (Datacenter) Network. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication* (2015), ACM, pp. 123–137.

[44] SEAMICRO, A. AMD SeaMicro SM15000 Fabric Compute Systems. `http://www.seamicro.com/sm15000`.

[45] SINGLA, A., HONG, C.-Y., POPA, L., AND GODFREY, P. B. Jellyfish: Networking Data Centers Randomly. In *NSDI* (2012), vol. 12, pp. 17–17.

[46] SUDAN, K., BALAKRISHNAN, S., LIE, S., XU, M., MALLICK, D., LAUTERBACH, G., AND BALASUBRAMONIAN, R. A Novel System Architecture for Web-Scale Applications using Lightweight CPUs and Virtualized I/O. In *High Performance Computer Architecture (HPCA2013), 2013 IEEE 19th International Symposium on* (2013), IEEE, pp. 167–178.

[47] WANG, G., ANDERSEN, D. G., KAMINSKY, M., PAPAGIANNAKI, K., NG, T., KOZUCH, M., AND RYAN, M. c-Through: Part-Time Optics in Data Centers. *ACM SIGCOMM Computer Communication Review 41*, 4 (2011), 327–338.

[48] WWW.HPCRESEARCH.NL. IBM BlueGene P&Q. `http://www.hpcresearch.nl/euroben/Overview/web12/bluegene.php`.

[49] ZHOU, X., ZHANG, Z., ZHU, Y., LI, Y., KUMAR, S., VAHDAT, A., ZHAO, B. Y., AND ZHENG, H. Mirror Mirror on the Ceiling: Flexible Wireless Links for Data Centers. *ACM SIGCOMM Computer Communication Review 42*, 4 (2012), 443–454.

[50] Analog Devices ADN4612. `http://www.analog.com/media/en/technical-documentation/data-sheets/ADN4612.pdf`.

[51] Analog Devices ADN4605. `http://www.analog.com/en/products/switches-multiplexers/digital-crosspoint-switches/adn4605.html`.

[52] Intel Xeon Processor D-1500 Family Product Brief. `http://www.intel.com/content/www/us/en/processors/xeon/xeon-processor-d-brief.html`.

[53] 10GBase-KR FEC Tutorial. `http://www.ieee802.org/802_tutorials/06-July/10GBASE-KR_FEC_Tutorial_1407.pdf`.

[54] Vitesse VSC3144. `https://www.vitesse.com/products/product/VSC3144`.