Larry: Practical Network Reconfigurability in the Data Center

Andromachi Chatzieleftheriou, Sergey Legtchenko, Hugh Williams, and Antony Rowstron, *Microsoft Research*

https://www.usenix.org/conference/nsdi18/presentation/chatzieleftheriou

This paper is included in the Proceedings of the 15th USENIX Symposium on Networked Systems Design and Implementation (NSDI '18).

April 9–11, 2018 • Renton, WA, USA

ISBN 978-1-939133-01-4

Open access to the Proceedings of the 15th USENIX Symposium on Networked Systems Design and Implementation is sponsored by USENIX.

Larry: Practical Network Reconfigurability in the Data Center

Andromachi Chatzieleftheriou, Sergey Legtchenko, Hugh Williams, Antony Rowstron
Microsoft Research

Abstract

Modern data center (DC) applications require high cross-rack network bandwidth and ultra-low, predictable end-to-end latency. It is hard to meet these requirements in traditional DC networks where the bandwidth between a Top-of-Rack (ToR) switch and the rest of the DC is typically oversubscribed.

Larry is a network design that allows racks to dynamically adapt their bandwidth to the aggregation switches as a function of the traffic demand. Larry reconfigures the network topology to enable racks with high demand to use underutilized uplinks from their neighbors. Operating at the physical layer, Larry has a predictably low traffic forwarding overhead that is adapted to latency sensitive applications. Larry is effective even when deployed on a small set of racks (e.g., 4) because rack traffic demand is not correlated in many DC workloads. It can be deployed incrementally and transparently co-exist with existing non-reconfigurable racks. Our prototype uses a 40 Gbps electrical circuit switch we have built, with a simply local control plane. Using multiple workloads, we show that Larry improves tail latency by to 2.3x for the same network cost.

1 Introduction

There is a rapid adoption of high bandwidth networking in the DC. It is now common to deploy 40 Gbps to the server [59], and 50-100 Gbps is becoming popular [1]. An increasing number of applications are capable of consuming that bandwidth [22, 24, 34, 36, 42, 51] and require low and predictable latency [24, 41, 59]. Emerging techniques such as disaggregation of DRAM and non-volatile memory are also sensitive to latency and packet queuing [26]. This is a challenge because a large fraction of the traffic for these applications is not rack local [44, 59], and rack uplink bandwidth is typically oversubscribed [28, 47] which leads to rack-level congestion.

This paper presents Larry, a network design that addresses rack uplink congestion by dynamically adapting the aggregate uplink bandwidth of the rack to its traffic demand. For that, racks with congested uplinks use spare uplink bandwidth from physically adjacent racks. Larry targets workloads in which rack traffic is bursty and loosely correlated across racks, and we observe these properties in traces from our DCs.

Using local resources for traffic offloading has been first proposed by GRIN [18]. However, GRIN offloads traffic through multiple hops at layers 2 or 3. This typically adds 200-500 ns *per hop* even in cut-through mode and without queuing [6, 26, 60]. For some applications that require round trip times of a few microseconds [24, 26], this could represent a non-negligible latency overhead. In contrast, Larry is reconfigured at the physical layer, and only adds a predictable end-to-end forwarding overhead of a few nanoseconds. This *local reconfigurability* differentiates Larry from prior work on fully reconfigurable networks [23,25,27,29,30,39,40,43,50,54–56]. These systems typically redesign the *entire DC network*, making them efficient, but hard to deploy in practice, especially in existing data centers.

Larry uses a custom electrical circuit switch and exploits unused ports on the ToRs. Larry is designed for small-scale deployments of physically adjacent racks (e.g., 4 to 6) called *rack sets*. In a rack set, the ports used on *each* ToR to connect to the aggregation switches are instead connected to the circuit switch. The circuit switch is then connected to the set of aggregation switches that would have been connected to the ToRs. Any non-cabled ToR ports are also connected to the circuit switch. This does not change the number of uplinks to the aggregation switches, but creates a network reconfigurable at the physical layer and local to the rack set. Once configured, traffic between ToRs and aggregation switches is transparently forwarded by the circuit switches at the physical layer. Within the rack set, a rack with high traffic demand can hence forward its traffic

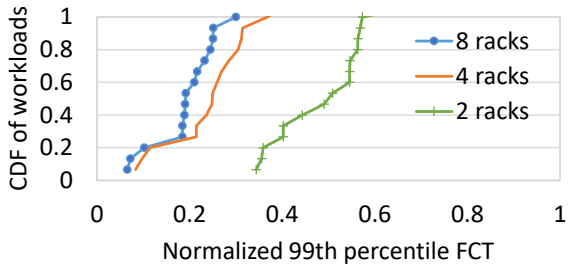


Figure 1: Impact of local reconfigurability on tail FCT.

through underutilized uplinks on other racks. The properties of our design are the following:

Incremental Deployment. The smallest unit of deployment is the rack set. Therefore, if a cloud provider suffers from congestion in a specific service (e.g., storage racks), local reconfiguration can be deployed only on the racks used for that service.

Transparency. Larry works with the DC network in place: any existing global controllers see links failing and being established, as reconfiguration occurs at the physical layer. No reconfiguration-specific routing state is needed outside the rack set and reconfiguration events are only visible to ToR and aggregation switches. We use off-the-shelf ToR and aggregation switches, and no software or hardware changes are required on the servers to use the extra uplink bandwidth.

Cost efficiency. All the racks in the rack set are physically close to each other, which allows, even at 100 Gbps, to use cheaper passive cables for all additional connectivity. No extra connectivity is required in the rest of the DC network. Further, our circuit switch design uses simple, commercially available crosspoint ASICs [7].

Low latency. The crosspoint ASIC forwards the incoming signal from the high-speed serial data link without processing any packets, hence the forwarding latency of the circuit switch is within a few nanoseconds [7].

Our evaluation shows that uplink reconfiguration can be done with low overhead and augmenting a traditional oversubscribed topology with Larry increases the performance per dollar by up to 2.3x.

The rest of the paper is organized as follows. Section 2 describes the benefits and challenges of local reconfigurability. Section 3 details the design of Larry that implements local reconfigurability at the physical layer. Section 4 discusses the practicality of our approach. Section 5 evaluates our design. Finally, Section 6 presents the related work and Section 7 concludes.

2 Local Reconfigurability

We now describe the benefits and challenges of local reconfigurability.

2.1 Is Local Reconfigurability Useful?

Local reconfigurability reduces rack uplink congestion by using underutilized uplink bandwidth from a small set of adjacent racks. We now show that some key DC workloads do exhibit low traffic correlation across racks and can benefit from local reconfigurability. For our analysis, we assume that any uplink can be used by any rack in a rack set composed of m physically adjacent racks. We also assume that the core network is fully provisioned. These optimistic assumptions allow to estimate an upper bound on the performance of local reconfigurability and will be refined in latter sections.

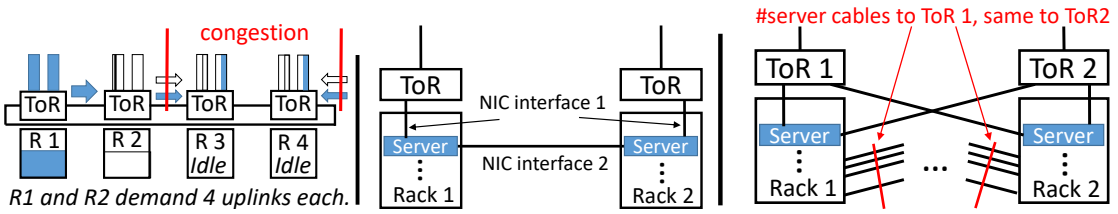
We use two DC traces and show that they exhibit low rack traffic correlation. The first trace contains all files accessed by a large-scale cloud service over a week in mid-2016. The traces do not differentiate between local accesses to disk and remote accesses over the network. To measure the impact of data transfers on the network, we simulate a small local write-back cache at the compute node to which all accesses are made. Files that have not been accessed for more than a day are de-staged to the DC storage tier over the network. On a cache miss, a file is read from the storage tier over the network. We group the network transfers by rack, then form groups of eight randomly selected racks[1]. For every group of racks, we replay all the network transfers between the storage tier and the cache in a flow-based simulator that computes the flow completion times (FCT) assuming max-min fairness bandwidth allocation and one flow per file access.

In the simulated topology, each rack is provisioned with u uplinks, such that a rack set with m racks has $u \times m$ uplinks. In the rack set, at any point in time, each rack gets a subset of the uplinks that is proportional to its demand. We compute the tail FCT for each sub-trace for $u = 4$ and $m = 1, 2, 4, 8$ and use $m = 1$ (all racks are independent) as a baseline. Figure 1 shows the CDF of the 99^{th} percentile FCT for all the simulated sub-traces for different rack set sizes, normalized to the baseline. We can see that tail FCT is improved for *all* rack set sizes, showing that local reconfigurability reduces congestion. The results also show that for this workload, there is only a marginal benefit of having rack sets larger than 4 racks.

The second trace was obtained from the authors of [27], and covers four clusters from a large cloud provider. As described in [27], the clusters have between 100 and 2,500 racks and run a mix of workloads. The results qualitatively show a similar trend: when operating at a scale of 4 to 8 racks, local reconfigurability can reduce most of the uplink bandwidth congestion during peak demand.

Notably, in the first trace, demand is not correlated de-

[1] The trace lacks rack placement information.



(a) Congestion in ToR-to-ToR topology. (b) Cross-rack GRIN. (c) Servers connected to neighboring ToR.

Figure 2: Challenges of implementing local reconfigurability at layers 2 or 3.

spite the fact that all racks host the same workload (storage). This suggests that rack traffic decorrelation is a property of the workload. We hence expect the analysis to hold if racks were chosen according to their physical proximity. Other studies of workloads [28, 40, 44] show the skew at rack-level. In the public cloud, services and applications are used by a large number of clients with decorrelated demand, and services often use several generations of rack hardware. We expect that local reconfigurability can be beneficial to a number of key workloads in that context.

2.2 Design Challenges

Several network designs can enable local reconfigurability. This section discusses the challenges of implementing local reconfigurability at layer 2, thereby motivating the use of layer 1 circuit switching. The number of aggregations switch ports connected to the core network *per rack* is the same for all described designs.

Local reconfigurability can be implemented at layer 2 by enabling non-shortest path routing between ToR and aggregation switches. This can be done by either adding extra packet switches between ToRs and the aggregation layer, or by interconnecting spare ToR ports in a multi-hop topology. Figure 2(a) shows one simple example of the latter, with four ToRs interconnected into a ring topology[2]. A Software-Defined Network (SDN) controller monitors the uplink bandwidth usage and balances traffic in the rack set. However, this introduces extra forwarding latency as store-and-forward packet switching adds approximately 1 μs per hop. Cut-through packet forwarding reduces the latency down to 200 to 500 ns per hop [6, 26, 60] but is not supported by all switches. Gao et al [26] describe emergent applications with a 3μs round trip latency budget, for which extra packet forwarding could increase round trip latency by 30% or more. In addition, for ToR-to-ToR topologies, there will be worst case scenarios with fate sharing of the ToR-to-ToR links leading to congestion. In Figure 2(a), racks have two uplinks, but each of 1 and 2 demands 4 uplinks. These racks hence forward half of their demand to the idle racks 3 and 4. This causes congestion on the ToR-to-ToR links such that 1 and 2 cannot use all the available uplink bandwidth. Finally, this increases the unpredictability of the network, as the latency of a flow depends on the uplink through which it is forwarded, and its throughput is impacted by flows from other racks on the ToR-to-ToR links. In the control plane, this requires custom routing, fine-grain bandwidth management and consistent updates to sets of SDN-enabled switches. This increases the probability of software bugs in the controller [46] and reconfiguration latency [31].

GRIN [18] implements local reconfigurability *within* a rack by interconnecting spare ports on servers' NICs and allowing a busy server to forward traffic through idle servers in the rack. This approach can be extended to enable local reconfigurability across racks by interconnecting servers on neighboring racks as shown in Figure 2(b). In addition to the extra hop latency, the flexibility is limited by the number of spare ports per NIC. At 40 Gbps per port and beyond, NICs with more than two ports (or multiple NICs per server) are not common, which limits the approach to *pairs* of servers. Packet forwarding and traffic prioritization between NIC ports need to be done in hardware, which is not supported by all NICs.

Finally, avoiding extra latency is possible with extra cables and ports at ToR or aggregation switches. For example, spare NIC ports on the servers can be connected to neighboring ToRs, as proposed by Liu et al [40] (see Figure 2(c)). However, in that case, the number of ports required for server connectivity increases by at least a factor of p (if p-port NICs are used) and it is unlikely that ToRs have enough spare ports. Alternatively, all spare ports on the ToRs can be connected directly to aggregation switches, which either requires additional aggregation switches and extra optical cables, or increased oversubscription at the aggregation layer.

3 Overview of Larry

We enable low, predictable forwarding latency by ensuring that any extra forwarding is done at the physical

[2] Alternative topologies are also possible.

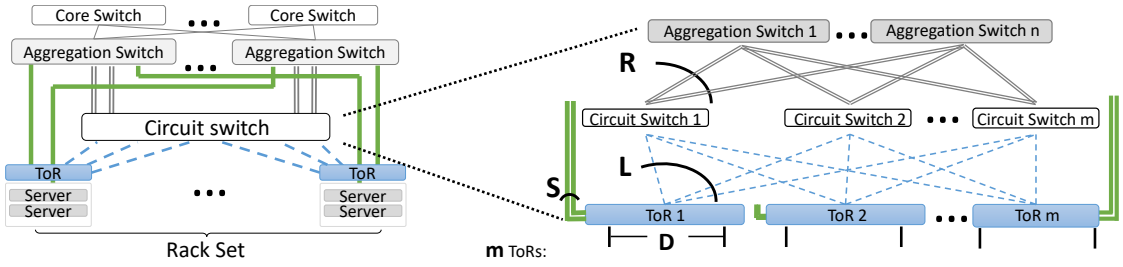


Figure 3: Architecture overview.

layer. We achieve this by using a *reconfigurable fabric* at the rack set that operates at the physical layer and allows uplinks to be *migrated* from one rack to another. The fabric enables racks to adapt their uplink bandwidth while appearing as a collection of independent racks to the rest of the DC. Uplinks are used to forward all the traffic from the ToRs to the aggregation switches. When a ToR is experiencing congestion to the core network, a local controller can physically migrate uplinks to that ToR from a non-congested ToR in the rack set. For example, the ToR could have 8 uplinks, and then have 2 additional uplinks migrated to it. The packet-switched network literally observes the two uplinks as disconnecting, and then connecting to the new ToR. We use Equal-Cost Multi-Path (ECMP) routing on the ToRs and aggregation switches to balance traffic across uplinks [33]. This is done at the flow granularity, such that for each packet, the switches hash the TCP five tuple to determine the port for the next hop. Uplink migration is enabled by deploying a custom low-cost low-radix electrical circuit switch in each rack to complement the existing ToR packet switches.

3.1 The circuit switch

An electrical circuit switch forwards the *electrical signal* received on one port through a circuit established to another port on the switch. The switching cannot be done *per packet*: once established, a circuit is expected to exist for hundreds of milliseconds or longer. The circuit switch does no packet header inspection or buffering, as a result the latency of transferring the signal from one port to another is on the order of 2 ns [7]. This is low enough to be transparent to the upper layers of the network. The packet switches connected to the circuit switch are unaware of its presence, and perceive the link as being a cable. A relatively simple hardware design using crosspoint ASICs [2, 7, 10] available today can have up to 48 100 Gbps ports. Section 5.1 describes the design of a prototype switch we have built.

Cost. The hardware architecture of crosspoint ASICs is typically simpler than merchant silicon. Crosspoint ASICs have no packet buffers or packet processing logic. In addition to that, because they do not need to process packets, the incoming signal from the high-speed serial data link does not need to be de-multiplexed to lower speed lanes, as done in packet switches. Crosspoint ASICs hence do not require SerDes at the inputs and outputs[3] which are typically challenging to design at high speed rates [15]. The switch core itself is hence simpler and switching is done at the line rate. In equivalent manufacturing processes, crosspoint ASICs should take less die area and have a lower power consumption for the same number of I/O lanes.

While the cost of the switch depends on the cost of its ASIC, it is harder to do a fair comparison of packet- and circuit- switches due to the cost of the other components, the difference of production volume, etc. Section 5.2.3 includes a sensitivity analysis on cost.

3.2 The reconfigurable fabric

Figure 3 shows the topology of a reconfigurable fabric that connects a set of racks to the aggregation switches. Each rack contains a set of servers connected to an unmodified ToR. ToRs and aggregation switches operate at layer 2 and above. For simplicity, in this section we assume that the reconfiguration is done by a single large circuit switch, as depicted on the left-hand side of Figure 3. In practice using a single circuit switch has scaling and deployment issues, and we distribute the circuit switch using one small-radix circuit switch per rack. The circuit switches are interconnected as shown on the right-hand side of Figure 3. We will describe this in more detail in the next section.

In a traditional network design, a ToR is connected to all the servers in the rack and has several (e.g., 8) uplinks connected to the aggregation switches. In our design, only a fraction of these uplinks is directly connected to the aggregation switches. We keep at least two (full green lines in Figure 3) to ensure the rack is still connected to the core if an aggregation switch fails while no uplinks are available through the circuit switch. The

[3] A SerDes is a pair of functional blocks that convert data between a fast serial link and slower parallel interfaces in each direction.

remaining ToR ports (e.g., 6), that would have been connected to aggregation switches are instead connected to the circuit switch. For each of these, a port on the circuit switch is then connected to the aggregation switch. Finally, any unused ports on each ToR are also connected to the circuit switch. Unused ports typically exist for topological reasons (e.g. due to oversubscription or mismatch between the number of servers per rack and the switch radix). The links from the circuit switch to the aggregation switches are active (optical) cables, and are shown as gray double lines in Figure 3. All the links from the ToR to the circuit switch use passive (copper) cables (dashed blue in Figure 3). At current price points, this reduces the cabling costs: at 100 Gbps, the transceivers account for a large part of the cable cost. While transceiver costs are currently dropping, passive cables are cheaper than the active ones [45]. At 100 Gbps, passive cables are available up to 5 m [16], which is long enough for rack sets of at least 4 racks. As the data rate increases, the loss of the passive cable is typically reduced by increasing the thickness of the cable. We have successfully used our 40 Gbps circuit switch prototype with long passive cables to interconnect a row of 4 racks. An important aspect of our design is that all the extra connectivity is done at lower cost using spare ToR ports, passive cables and circuit switch ports. The number of aggregation switch ports and active cables used in our design is the same as in a traditional topology with the same level of oversubscription.

The circuit switch can connect any of its uplinks to the aggregation switches to any ToR in the rack set. On the circuit switch, the number of uplinks is lower than the number of links to the ToRs. Therefore, a fraction of the ToR ports will not be connected by a circuit to an aggregation switch port at a given point in time. Such *disconnected* links have no PHY established and are seen as disconnected ports by the ToR. These links are used by the ToR to get extra bandwidth on demand. Consider a scenario where each rack has on average 8 uplinks to the aggregation switches, and this is not sufficient to satisfy the demand on one rack. With the reconfigurable fabric, any links that are underutilized on other racks in the rack set can be migrated to the rack experiencing high demand. The number of extra uplinks that can be migrated is a function of demand across the racks, and is at most the number of extra unused ports connected to the circuit switch from each ToR. More formally, if each rack has on average U uplinks to the aggregation switches, a ToR with S directly connected uplinks and L links to the circuit switch will be able to have between S and $S+L$ uplinks ($S \leq U \leq S+L$). The uplink bandwidth allocation on each ToR is managed by a local controller that monitors the set of ToRs in the rack set and reconfigures the circuit switches through a separate control plane. We will describe this in Section 3.4.

Physical layer reconfiguration enables low forwarding latency on all uplinks regardless of their physical location in the rack set. It also improves performance and predictability: there is no fate sharing of uplinks by multiple racks in the rack set. So far, we considered for simplicity that each rack set had a single large circuit switch. However, this is unpractical as even at small scales (e.g., 4 racks), the number of reconfigurable fabric ports exceeds the port count of the largest electrical circuit switch capable of 100 Gbps. This makes the cost prohibitive. Further, using a single circuit switch per rack set introduces a single point of failure.

3.3 Distributing the circuit switch

To simplify deployment and management, we use one circuit switch per rack. Each ToR needs to be attached to *all* the circuit switches in the rack set. The links to aggregation switches are distributed across the circuit switches, ideally with (at least) one link to each aggregation switch per circuit switch (see Figure 3 right hand side). This reduces the flexibility compared to a single large circuit switch: some pairs of ports cannot be connected by a circuit. For example, a port attached to one circuit switch cannot have a circuit to a port on another circuit switch. *Is this flexible enough to operate efficiently?*

To answer this, the key insight is that allowing any pairs of ports to be connected provides more flexibility than it is necessary for our design. The reconfigurable fabric can be represented as a bipartite graph in which vertices are ToR and aggregation switch ports and edges are circuits established between them. As the traffic demand changes, the circuit switch instantiates the bipartite graph that is the best adapted to the demand. Intuitively, since the graph is bipartite, there is no need to guarantee that *any* two ports can be connected, but only that each ToR has the required number of uplinks.

To show that the latter can always be achieved, we denote L the number of links from each ToR to the circuit switches and R the number of links from each circuit switch to the aggregation switches (see right hand side of Figure 3). We have $L \geq m$ because each ToR is connected to every circuit switch and there is one per rack. In addition to that, as described in the previous section, on each circuit switch the number of links to the ToRs is equal or higher to the number of links to the aggregation switches, i.e. $L \geq R$. Intuitively, these constraints over L mean that: *(i)* an uplink on a circuit switch can be connected to *any* ToR in the rack set and *(ii)* there are enough ports to ToRs on each circuit switch to allow all uplinks to be connected at the same time.

Therefore, for any uplink assignment in the rack set,

there exist a circuit configuration on the circuit switches that will instantiate the assignment. Furthermore, this configuration can be easily found. Intuitively, if each circuit switch is associated with a color, this problem can be expressed as an edge f-coloring of the bipartite graph [57]. The f-coloring problem is NP-complete in general, however f-coloring of *bipartite* graphs can be done in polynomial time [57]. It means that our set of circuit switches can instantiate any uplink assignment. In the evaluation, we show that our fabric achieves the performance of a single large circuit switch.

The number of ports required on each ToR and circuit switch is shown in Figure 3 (right hand side). On a ToR, our design needs $D+S+L$ ports: D to servers in the rack, S directly attached to aggregation switches and L to the circuit switches. On a circuit switch, $L+R$ ports are required: L to the ToRs in the rack set and R to aggregation switches. For example, assuming both 32-port ToR and circuit switches, $D = 16$ ports to servers and $S = 2$ static uplinks per ToR, our design can scale to $m = 14$ racks per rack set. In this case, the limiting factor is the number of ports on the 32-port ToRs, because half of the ToR ports are used for in-rack connectivity. In practice, assuming a standard hot-/cold-aisle DC layout, we are limited to 4-6 racks by 5 m passive cables. For the rest of the paper, we conservatively consider rack sets with 4 racks only.

3.4 Controller

Each rack set has an independent lightweight controller that monitors the network load within the rack set, decides when to migrate the uplinks and manages the reconfiguration. The controller and circuit switches communicate through a control plane. In the prototype, we use a pre-existing management switch connected to board management controllers in the rack [14]. The controller can specify a new port mapping for each circuit switch, and read out the current port mapping. To ease deployment, the controller is designed as a soft-state process. When started, it is provided with a rack set configuration that describes the racks, circuit switches, ToRs and aggregation switches associated with the rack set. It then reads the current mapping from each circuit switch.

A key property of this controller is that it is *local*: it requires no global information from the core network. It only relies on the information that comes from the ToRs in the rack set, not even from the aggregation switches. We assume that ToRs enable a mechanism to query per port traffic statistics, e.g., OpenFlow [12]. This design simplifies deployment and reduces the impact on any existing global SDN controllers used in the core data center network. We assume that the core network is configured such that any flow-granularity traffic management is orthogonal to our design, but we do need the core network to use a mechanism, e.g., ECMP, to efficiently spread the network load across all possible paths to a rack[4].

When the controller migrates an uplink, there are two approaches to handling this from the perspective of the core network. The clean approach is to ensure no packet loss, which can be done by signaling to the global controller that the link will fail before it is remapped. The global controller then removes all routes that currently use the link. After that, the circuit switches are reconfigured and the PHYs are established over the new circuits. We assume that the ToRs can report physical layer changes to the rack set controller. When the new link is established, it is reported to the global controller that starts to route traffic over it. This approach can be for example easily implemented using off-the-shelf OpenFlow-enabled ToRs and aggregation switches [4,9].

The dirty approach is to allow rack set controllers to operate independently without communicating with the global controller. This relies on the existing core network monitoring and management mechanisms to see the reconfiguration as a link failure followed by a link repair. However, this can lead to a small number of packets being lost on the failed links. The reconfiguration typically occurs when the link is not highly utilized. In the evaluation, we show that reconfiguration delays are low, and links are reconfigured infrequently, suggesting that this approach will not cause significant traffic disruption.

The goal of the controller is to migrate uplinks to decongest ToRs with high demand. We do not modify the end-systems, and the controller has no global visibility, so it has no understanding of potential future demand for either egress or ingress traffic to the racks. We therefore use a greedy algorithm that periodically classifies each ToR in the rack set as being underutilized or potentially congested. To do this, the controller uses per-port byte count metrics obtained from the ToRs. This can be done at a fine-granularity: our prototype shows that if the controller is managing a rack set with 4 racks, polling all 4 ToRs every millisecond would generate only in the order of 10 Mbps of traffic.

The algorithm takes uplinks from ToRs which are not being utilized, and migrates them to ToRs that could potentially use more bandwidth. The greedy algorithm computes the aggregate uplink utilization for each ToR, in terms of ingress and egress bandwidth. It then determines the number of links from the ToR to the aggregation switches required to support that demand, considering a link fully utilized at 85% of its nominal capacity. If it determines that a ToR could support the workload with one or more links fewer than it currently has, the ToR is marked as underutilized. If all the links to the aggregation switches are fully utilized, it is marked as congested.

[4]For example by using groups of type `select` in OpenFlow 1.3 [12, 53].

The controller knows the full topology of the rack set and has a list of the circuit switches. For each circuit switch, it greedily assigns an uplink from the most underutilized ToR to the most overloaded ToR, records the corresponding circuit configuration and re-computes the link utilization. This happens until no more uplinks can be reassigned, because either there are no more underutilized uplinks, or all overloaded ToRs reached their maximum number of uplinks.

The controller is currently configured to be conservative to minimize the impact of reconfigurations on performance. If all ToRs in the rack set are underutilized no uplinks are migrated. If no ToRs are underutilized, no uplinks are migrated even if other ToRs are congested except if congestion is detected on a ToR that has less than its fair share of the aggregation links attached to the circuit switch. If so, we allocate the ToR the fair share of links. This ensures that all ToRs get at least their baseline uplink bandwidth under congestions across multiple racks. The output of the algorithm is a circuit configuration for each circuit switch that instantiates the new uplink assignment.

4 Discussion

This section discusses the feasibility of deploying Larry in production DCs. We first focus on the features that facilitate deployment:

Good failure resilience. A failure in one rack set does not lead to failures in other rack sets or elsewhere in the core network. The failure of the controller or any circuit switch does not disconnect racks from the core network as racks have direct connections to aggregation switches and controller failures do not lead to inconsistent network state. The rack set configuration state is stored in persistent storage and the internal local state is soft. We assume that a data center-wide service can monitor the liveness of the controller, and restart it upon failure.

Transparency. The reconfiguration is scoped within a rack set and is transparent to the end-systems. At the physical layer, PHY loss and establishment events only occur on the ToRs and aggregation switches. These events are managed by a local rack set controller that updates the mapping of links between the aggregation switches and the rack set ToRs. Larry can operate without modifications to the end-system operating system or applications, or the firmware or hardware design of the ToR or aggregation switches.

Support for incremental deployment. Larry does not require any changes to the core network management or operation. Existing cabling from the aggregation switches to the racks can be used. Within a data center, some sets of racks can have local reconfigurability provisioned (e.g., storage racks), while other racks can be deployed without it. It is even possible, to retrospectively fit this to deployed racks if needed.

Figure 4: Prototype circuit switch.

Ease of deployment. State of the art reconfigurable topologies can be hard to deploy and operate [48]. In contrast, the sensitivity of Larry to environmental factors is negligible, and no specific operator experience is required. Each rack needs a 1U slot for a production version of the circuit switch.

Deployment Challenges. Larry requires extra cabling across racks, which complicates rack provisioning. We ensure that the additional cabling complexity is limited. The cabling is scoped to a rack set and is symmetric: all racks have the same number of extra cables to the other racks in the rack set. Additional cables originate on the ToRs and target the circuit switch. With a rack set size of m it is feasible to have just m extra cables per rack, where each cable carries multiple lanes similar to original 100 Gbps cables that bonded ten 10 Gbps lanes.

The reconfigurability of the network in the rack set can increase the link churn and create topological asymmetry. Link churn and asymmetry exist in DCs today, and prior work has already been done on efficient load balancing and neighbor discovery mechanisms to address the associated challenges [20, 47]. Larry also increases the routing state update rate on the switches. While reconfiguration in the rack set is expected to be infrequent (see Section 5.2.4), this could induce overhead on core network (T2) switches. Finally, the performance of Larry can be improved by rethinking existing DC components. In particular, the PHY negotiation mechanism on the ToR and aggregation switches should be tuned to minimize the link downtime.

5 Evaluation

Our evaluation aims to explore the benefits and overheads of deploying our design in a data center. We aim to answer the following questions: *(i)* What are the overheads of reconfiguring the physical link? *(ii)* How does Larry compare to static DC topologies with respect to performance metrics and cost efficiency? and *(iii)* What are the properties of the reconfigurable fabric?

For that, we first evaluate reconfiguration overheads

using a prototype circuit switch. Then, we explore the properties of our design with data center workload traces using discrete event simulation. Overall, our results show that local reconfigurability improves the performance per dollar of the network. The reconfiguration is infrequent, mainly adapting the network to macro changes in the workload and reconfiguration overheads are low.

5.1 Prototype

To demonstrate the viability of our design, we have built a prototype circuit switch (see Figure 4). The circuit switch has a 2U form factor with 40 front-facing QSFP+ ports. We use the M21605 asynchronous fully non-blocking crosspoint switch ASIC from MACOM [7]. The ASIC supports up to 160 lanes at 10 Gbps per lane and can connect any two lanes with an internal circuit. Each QSFP+ port is internally connected to four lanes on the crosspoint ASIC enabling 40 Gbps per port. The crosspoint ASIC is controlled by custom firmware on an ARM Cortex-M3 micro-controller on the switch. The micro-controller has an Ethernet link to an external control plane and a parallel interface to the ASIC.

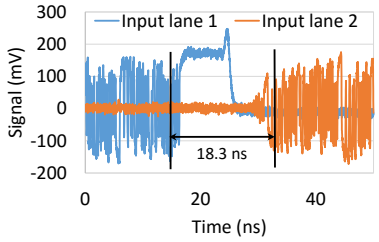
We emulate the egress/ingress traffic between a ToR and its aggregation using two Arista 7050X packet switches with 32 40 Gbps ports [3]. The packet switches are both attached to the circuit switch and to each other using passive copper cables. For traffic generation, we connect 2 servers per packet switch. Each server has a Mellanox ConnectX-3 40 Gbps NIC [5] and a dual Intel Xeon E5-2660 v3 CPU at 2.6 GHz running Windows Server 2016. All switches are connected to an Ethernet control plane and controlled by a separate server. The packet switches support OpenFlow and we use the Floodlight OpenFlow controller to reconfigure the routing state of the switches during reconfiguration [13].

When we designed the switch, 100 Gbps network components were not widely available. Today, this can be implemented using for example the MAXP-37161 crosspoint ASIC that supports 25 Gbps per lane [8]. A 100 Gbps zQSFP+ port [17] uses 4 lanes so it is possible to build a circuit switch with 16 ports at 100 Gbps by using four MAXP-37161 ASICs in parallel and routing each lane of a zQSFP+ connector to a separate crosspoint ASIC then back to another zQSFP+ connector. While 16 ports are enough to implement all the topologies described in the evaluation, there exist up to 48-port crosspoint ASICs operating at the same bandwidth per lane.

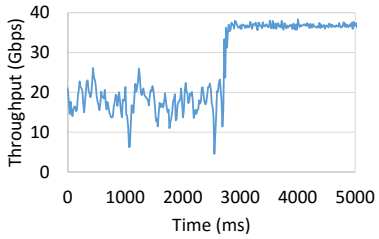
5.1.1 Micro-benchmarks

We evaluate the reconfiguration overheads by measuring the circuit switching time and its impact on throughput.

Switching time: We measure the time taken by the crosspoint ASIC to establish a circuit. For that, we connect a high frequency oscilloscope to two spare ports on the circuit switch. The oscilloscope measures the voltage output on lane 0 of each port. We use one of the ports attached to a packet switch as a traffic generator. We first set up a circuit between the packet switch and the first port of the oscilloscope. We then connect the packet switch to the second port and measure the time taken by the signal to appear on the second port. Figure 5(a) shows the voltage generated by the signal over time on both ports during reconfiguration. Initially, the signal appears on the first port while the second port is idle. After a transition period of 18.3 ns on average, the signal appears on port 2, while port 1 becomes idle. We ran the experiment 10 times and observed a reconfiguration delay of 19.5 ns in the worst case, with a median of 18.27 ns. This is lower by about an order of magnitude or more compared to other circuit switching proposals for DC networks [27, 50] because electrical circuit switching requires no physical movement in the switch.



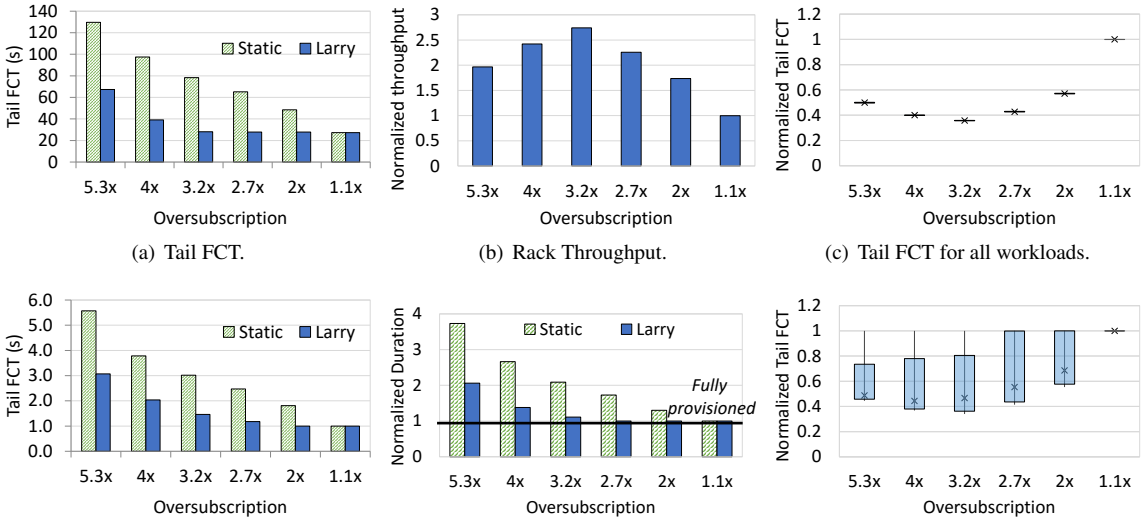
(a) Signal under reconfiguration.



(b) TCP throughput on reconfiguration.

Figure 5: Prototype micro-benchmarks.

Throughput: We now measure the impact of adding uplinks on application throughput. We form two source-destination pairs using the four servers such that traffic for both pairs must traverse both packet switches. We use `NTttcp` [11] to saturate the NIC bandwidth by creating 20 TCP flows, one per core, from each source to its destination and measure the throughput at the destination with a 15 ms interval. Initially, there are no circuits on the circuit switch, so packet switches can only use one 40 Gbps link for all traffic. The link is fair-shared across both source-destination pairs. We then set up a circuit that creates an additional link between the packet switches and configure them such that each source desti-



(a) Tail FCT. (b) Rack Throughput. (c) Tail FCT for all workloads.

(d) Tail FCT. (e) Impact of queuing on experiment duration. (f) Tail FCT for all workloads.

Figure 6: Base performance with the Storage (top) and Production (bottom) workloads.

nation pair uses its own link. Figure 5(b) shows the receive throughput on one receiver over time. We can see that the application is able to benefit from reconfiguration and doubles its average throughput. This is achieved transparently without modifications to packet switches or servers. The reconfigurable uplink is similar to a physical cable: we observe no packet loss on any of the links, and both uplinks achieve the same throughput.

5.2 Simulations

We now demonstrate that deploying our design in existing DCs offers performance and cost benefits. At the high level, we aim to answer the question: *what is the benefit of adding Larry to a static topology?*

To scale our evaluation to our traces, we use a simple flow-level simulator that represents each TCP flow by a single flow in the topology graph. We obtained the simulator that has been used in [21, 37, 38]. It has been cross-validated with real hardware in the context of storage [21] and offers a good trade-off between scalability and accuracy. Each flow is routed through one randomly chosen shortest path between the source and the destination. Network components, including ToRs, circuit switches and servers, are represented as vertices, while edges simulate links. Each link is bidirectional, has a bandwidth and keeps track of all the flows traversing it at any time. The simulator computes the bandwidth dedicated to each flow using max-min fairness. On flow creation or completion, the throughputs of all impacted flows are recomputed. Overall, this simulates the high-level behavior of a typical TCP network with ECMP routing. During reconfiguration, the topology graph is updated, and all flow throughputs are recomputed. Unless otherwise stated we do not simulate the downtime on links during reconfiguration as it typically occurs only a few times per hour (see Section 5.2.4).

Topologies. Our baseline topology, denoted as *static*, has m racks directly attached to aggregation switches. The performance of the static topology depends on the oversubscription at the ToRs. For Larry, we augment the rack with a circuit switch and interconnect the circuit and packet switches as described in Section 3. For both topologies, we use 32-port 100 Gbps ToR and aggregation switches and have 32 servers per rack with 50 Gbps per server as described in Section 3.3. We use rack sets of 4 racks; the controller monitors traffic every 500 ms and sets the link capacity threshold to 0.85. Evaluation of alternative values showed no qualitative impact for the examined workloads.

Performance metrics. The metrics of interest are *FCT*, *rack throughput* and *workload duration*. The FCT is the time span between the creation of the flow and its completion and directly impacts application performance. In the following experiments, we focus on 99^{th} percentile FCT, denoted as *tail FCT*. We also measure the throughput of each ToR every second by aggregating throughputs of all the flows sent or received by the ToR and taking the maximum between ingress and egress throughput. We compute the average throughput achieved by the racks during the periods for which the rack is not idle. Finally, we measure the *workload duration* as the timespan between the beginning of the first and the completion of the last request. We now detail our evaluation workloads.

5.2.1 Workloads

Storage (open loop workload). We use the traces described in Section 2.2. The data written to the storage tier is stored on 159 storage racks. For each rack, the writes are controlled by the storage service and are hence uniformly distributed over the day. Reads are very bursty and dominate over writes during peaks.

Production (closed loop workload). We use the traces from [27] as described in Section 2.2 that are representative of a typical production cluster. The traces contain the number of bytes sent each second between ToR source destination pairs, without flow- or request-level information. We model this as a closed loop workload with at most one outstanding request between each source destination pair. For a source-destination pair, each second in which $x > 0$ bytes were sent corresponds to a request. The request is sent at a throughput of x Bps. For large requests that exceed 200 MB we use multiple flows to leverage ECMP across all the uplinks. In a topology provisioned for peak, each request finishes within a second and the achieved throughput equals the throughput observed in the trace. We detect the fully provisioned bandwidth for each workload by down-scaling the link bandwidth to the minimal bandwidth that allows every request to finish on time for a given workload. Then, we oversubscribe the network and examine the impact on performance. Namely, oversubscription introduces queuing delays since there is only one outstanding request per source destination pair. We generate multiple *workloads* by mapping randomly selected rack-level traces to individual racks in a rack set. Within a rack, requests are randomly distributed across servers.

5.2.2 Base performance

We now compare our design to the static topology at different oversubscription ratios. We first describe the results for the Storage workloads shown in Figure 6. Figure 6(a) shows the tail FCT for both static and reconfigurable topologies in function of the oversubscription for one representative set of four racks. The tail FCT is relatively high even for well-provisioned topologies, showing that peak bandwidth demand can be high. As expected, the tail FCT drops as the oversubscription decreases because both topologies get more network resources. However, Larry has low tail FCT even when oversubscription is high. For example, the tail FCT for Larry with 3.2x oversubscription is about 64% lower compared to a static topology with the same oversubscription. Despite the 3.2x oversubscription, it has about 42% lower tail FCT than the static topology with 2x oversubscription and within 18% of a fully provisioned network. At the highest oversubscription, Larry improves tail FCT by a factor of 2.

This happens because Larry can reconfigure its topology to efficiently allocate bandwidth to the traffic. Figure 6(b) shows the average rack throughput of Larry normalized to the static topology in function of the oversubscription ratio for the same workload. At the highest oversubscription, the throughput per rack is 1.96 times higher for Larry. As the oversubscription decreases, the number of reconfigurable uplinks increases, which improves performance compared to static topologies. However, eventually, the bandwidth per rack provisioned on static topologies gets high enough to satisfy the demand, and the reconfigurability makes less difference. Overall, Larry has higher rack throughput for all oversubscriptions and up to 2.7 times higher at 3.2x oversubscription.

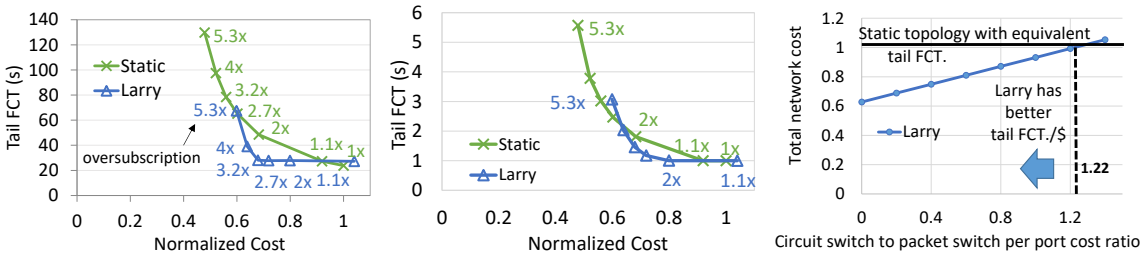
Figure 6(c) examines the generality of our findings depicting 10^{th} and 90^{th} percentiles as error bars, 25^{th} and 75^{th} percentiles as the box and the median as the cross in the box across all the workloads. The low variance reveals that our observations are consistent across all the simulations for the Storage workload.

We now focus on a representative Production workload. Figure 6(d) shows the tail FCT across different oversubscription ratios. Like in the Storage workload, we observe that reconfigurability improves tail FCT by up to approximately a factor of 2. We attribute this behavior to reduced queuing of requests. Namely, due to oversubscription, a delayed request can result in queuing of subsequent requests, which delays the overall completion of the experiment. Figure 6(e) measures the duration of the workload for Larry and static topologies normalized to the actual duration in a non-blocking network. Varying the oversubscription, we show that Larry manages to complete the workload almost in time with up to 3.2x oversubscription while static topologies take much longer to complete. Finally, Figure 6(f) shows the variance of our results across all the workloads, represented as in Figure 6(c). We observe higher variance across simulations because some workload instances have extremely low load. However, median values exhibit the same trends as for the Storage workloads.

5.2.3 Performance per dollar

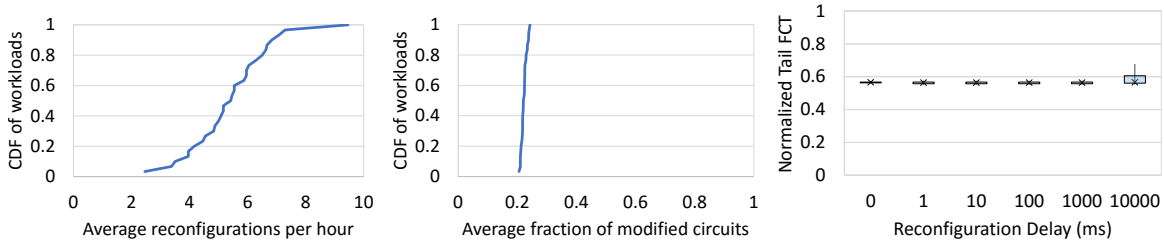
We now examine the performance per dollar of Larry. Intuitively, our design improves performance but requires additional ports and cables at the rack level, and we aim to determine whether its performance is worth the extra cost. We evaluate the deployment cost for the entire data center network using a cost model from a large DC provider that includes the volume costs of cables, merchant silicon [5], and server NICs. Figures 7(a) and 7(b) examine the tail FCT of two representative Storage and Production workloads in function of the cost across static

[5] In the model, packet switch costs are *per device* and not per port.



(a) Storage workload, tail FCT vs cost. (b) Production workload, tail FCT vs cost. (c) Varying circuit switch cost.

Figure 7: Performance as a function of cost.



(a) Reconfiguration frequency. (b) Circuit changes per reconfiguration. (c) Varying circuit downtime.

Figure 8: Reconfiguration overhead.

and reconfigurable topologies. Each data point refers to different oversubscription, ranging from 1x to 5.3x in increasing order from right to left (see the data labels). For confidentiality reasons, we normalize the costs on the x-axis to the cost of a fully provisioned topology. Based on our estimations for the hardware costs, in this analysis we assume a circuit switch to packet switch per port cost ratio of 0.3. In both workloads, Larry has a better performance per dollar than static topologies for oversubscription ratios between 2x and 4x. For example, in Figure 7(a) a static topology with 2x oversubscription has a normalized cost of 0.68, which is approximately the same cost as Larry with an oversubscription of 3.2x that has 43% lower tail FCT. These oversubscription ratios are common in current DCs, meaning that the performance improvements described in Section 5.2.2 compensates for the cost of adding reconfigurable hardware.

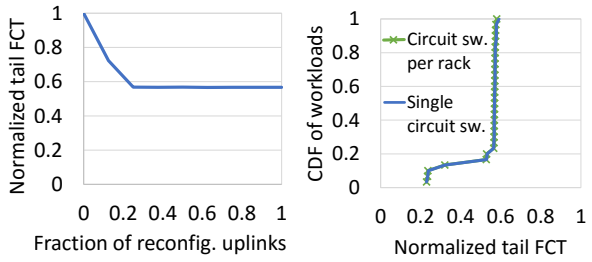
The cost of Larry depends on the cost of the circuit switches. Figure 7(c) shows the total network cost of Larry as a function of the circuit- to packet- switch per port cost ratio, for 2.7x oversubscription, which is a sensible design point in today's DC topologies. We compare it to the cost of a static topology with 1.1x oversubscription that has an equivalent 99^{th} percentile FCT for a representative Storage workload. We can see that if the circuit switch port cost is below 1.22, Larry is cheaper while offering the same performance. This means that Larry remains cost-effective even if the circuit switch has the same per port cost as a packet switch.

5.2.4 Properties of reconfiguration

We now focus on the Storage workloads and explore the properties of the reconfiguration. Larry incurs overheads in both control and data planes, so we first estimate the reconfiguration frequency. Figure 8(a) shows the CDF of the average reconfiguration frequency. The frequency is only a few times per hour for all workloads. The highest rate is 9 reconfigurations per hour, for a median of 5. This corresponds to one reconfiguration every 7 minutes for the worst case observed. Furthermore, when reconfiguration occurs, it only affects a small fraction of all the circuits. Figure 8(b) shows the fraction of all circuits changed per reconfiguration. On average, about 22% of the circuits are modified, and 24% in the worst case. This happens because of the conservative reconfiguration policy of the controller: reconfiguration happens only when imbalance across racks is high.

We now evaluate the impact of the circuit downtime on performance. The PHY negotiation delay is expected to dominate over the circuit reconfiguration, hence we conservatively vary the downtime between 1ms and 10s. Figure 8(c) shows that downtimes up to one second have low impact on the tail FCT because during peak demand FCT is dominated by queuing at the 99^{th} percentile.

We now vary the number of reconfigurable uplinks per rack, focusing on 2x oversubscription. With this oversubscription ratio, a rack has 8 uplinks, so the number of reconfigurable uplinks is varied between 0 and 8, and



(a) Restricting reconfigurability. (b) Distributing the circuit switch.

Figure 9: Design flexibility.

having 0 reconfigurable uplinks is equivalent to the static topology. Figure 9(a) shows the tail FCT for each setup, normalized to the tail FCT of the static topology with 2x oversubscription. Larry efficiently reduces tail FCT with as little as 2 reconfigurable uplinks per rack. This is because when a rack is congested, the other 3 are typically not. They can hence give away 6 uplinks to the congested rack, nearly doubling its aggregate uplink bandwidth.

Finally, to evaluate the overhead of using multiple circuit switches, we rerun all the Storage workloads using our uplink assignment algorithm on single large circuit switch. The algorithm then has no constraints on how the circuits are assigned and assigns uplinks only based on the traffic. Figure 9(b) shows how the tail FCT using a single switch compares to our design for all workloads. We can see that both lines overlap, showing that our greedy algorithm efficiently allocates uplink bandwidth in function of the rack demand.

6 Related Work

Local reconfigurability was described in GRIN [18] and Subways [40], showing that network resources are likely to be available locally in the DC. Both systems offload traffic at layers 2 or 3, with oveheads described in 2.2.

There has been extensive prior work on network reconfigurability using optical circuit switches [25, 39, 43, 50,54,55], free-space optics [27,30] and 60GHz wireless radios [23,29,58]. As in our design, packets sent by ToRs are forwarded over a fabric reconfigurable at the physical layer. Flat-tree is a DC network design can also adapt to different workloads by dynamically changing the topology between Clos [19] and random graphs [49] via small port-count packet or circuit switches [56]. However, these systems require substantial changes to the data plane or/and control plane of the entire DC network. Larry is not aiming to provide direct links between any ToRs in the DC, or change DC-wide network properties. Instead, it targets local uplink congestion on the ToRs. It hence does not require a DC-wide controller and can be deployed incrementally and transparently to the DC.

Flexibility can be achieved without reconfiguring the network topology. Hedera and Swan perform traffic engineering at the control plane [20, 32] and dynamically redirect traffic though least congested paths in the network. Expander topologies, such as Jellyfish [49] and Xpander [52] directly interconnect ToRs via static links and dynamically change the routing protocol in function of the load [35]. Expander topologies are not transparent, incremental deployments. They require re-cabling and custom routing policies for the entire DC network (or a large fraction of it). Larry is orthogonal to these designs and can interface with them if they are deployed.

ReacToR is a hybrid ToR switch design which combines circuit and packet switching [39]. A local classifier directs high-bandwidth flows to a set of uplinks connected core network circuit switches, while the rest traffic is directed to packet switches. ReacToR relies on host-side buffering of bursts until the circuits become available. Given the recent trend towards increasing server density per rack, XFabric proposes a hybrid design for rack-scale computers, where intra-rack traffic is forwarded by packet switches embedded on the servers' SoC over a circuit-switched physical layer [37]. Larry can be incrementally deployed without involving changes to any existing network components.

7 Conclusion

DC applications increasingly have high bandwidth demand and tight latency requirements cross rack. Larry is a network design that dynamically adapts the aggregate uplink bandwidth on the ToRs as a function of the rack demand. It ensures a low and predictable forwarding latency overhead by reconfiguring the network at the physical layer instead of re-routing traffic at layers 2 and 3. Larry can be deployed incrementally in existing DCs at the scale of a few racks, and transparently co-exist with the DC-wide controllers. It is hence well-suited for targeted deployments that improve the performance of specific tiers or services in existing DCs. We have built a prototype that uses a custom 40 Gbps electrical circuit switch and a small local controller. Using workload traces, we show that Larry improves the performance per dollar of the traditional oversubscribed networks by up to 2.3x.

Acknowledgments

We are grateful to Monia Ghobadi and Ratul Mahajan for sharing their traces. We also thank our shepherd George Porter and the anonymous reviewers for their helpful feedback.

References

[1] 100 Gbps revenues dominate in the cloud. https://aka.ms/P27bgu.

[2] Analog Devices Digital Crosspoint Switches. http://www.analog.com/en/products/switches-multiplexers/digital-crosspoint-switches.html.

[3] Arista 7050X Series. https://www.arista.com/en/products/7050x-series.

[4] Arista 7160 Series. https://www.arista.com/assets/data/pdf/Datasheets/7160-Datasheet.pdf.

[5] ConnectX-3 User Manual. http://bit.ly/2iA7Ken.

[6] Latency in Ethernet Switches. http://www.plexxi.com/wp-content/uploads/2016/01/Latency-in-Ethernet-Switches.pdf.

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

[8] Macom MAXP-37161 Crosspoint Switch. https://www.macom.com/products/product-detail/MAXP-37161.

[9] Mellanox SN2000 Series. http://www.mellanox.com/page/products_dyn?product_family=251.

[10] Microsemi Digital Crosspoint Switches. http://www.microsemi.com/products/switches/digital-cross-point-switches.

[11] NTttcp Utility. https://gallery.technet.microsoft.com/NTttcp-Version-528-Now-f8b12769.

[12] OpenFlow Switch Specification. http://bit.ly/2kk3Wyo.

[13] Project Floodlight. http://www.projectfloodlight.org/floodlight/.

[14] Project Olympus Universal Motherboard. http://bit.ly/2jAeUOM.

[15] SerDes signal integrity challenges at 28Gbps and beyond. http://bit.ly/2BiqCq8.

[16] zQSFP+ Cable Assembly, 5.0m Length. http://bit.ly/2kk47FO.

[17] zQSFP+ Interconnect System. http://www.molex.com/molex/products/family?key=zqsfp_interconnect_system.

[18] AGACHE, A., DEACONESCU, R., AND RAICIU, C. Increasing Datacenter Network Utilisation with GRIN. In *Proceedings of the 12th USENIX Conference on Networked Systems Design and Implementation* (2015), pp. 29–42.

[19] AL-FARES, M., LOUKISSAS, A., AND VAHDAT, A. A Scalable, Commodity Data Center Network Architecture. In *Proceedings of the ACM SIGCOMM 2008 Conference on Data Communication* (2008), pp. 63–74.

[20] AL-FARES, M., RADHAKRISHNAN, S., RAGHAVAN, B., HUANG, N., AND VAHDAT, A. Hedera: Dynamic Flow Scheduling for Data Center Networks. In *Proceedings of the 7th USENIX Conference on Networked Systems Design and Implementation* (2010), pp. 19–19.

[21] 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).

[22] CALDER, B., WANG, J., OGUS, A., NILAKANTAN, N., SKJOLSVOLD, A., MCKELVIE, S., XU, Y., SRIVASTAV, S., WU, J., SIMITCI, H., HARIDAS, J., UDDARAJU, C., KHATRI, H., EDWARDS, A., BEDEKAR, V., MAINALI, S., ABBASI, R., AGARWAL, A., HAQ, M. F. U., HAQ, M. I. U., BHARDWAJ, D., DAYANAND, S., ADUSUMILLI, A., MCNETT, M., SANKARAN, S., MANIVANNAN, K., AND RIGAS, L. Windows azure storage: A highly available cloud storage service with strong consistency. In *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles* (2011), pp. 143–157.

[23] CUI, Y., XIAO, S., WANG, X., YANG, Z., ZHU, C., LI, X., YANG, L., AND GE, N. Diamond: Nesting the Data Center Network with Wireless Rings in 3D Space. In *Proceedings of the 13th USENIX Symposium on Networked Systems Design and Implementation* (2016), pp. 657–669.

[24] DRAGOJEVIĆ, A., NARAYANAN, D., NIGHTINGALE, E. B., RENZELMANN, M., SHAMIS, A., BADAM, A., AND CASTRO, M. No Compromises: Distributed Transactions with Consistency, Availability, and Performance. In *Proceedings of the 25th Symposium on Operating Systems Principles* (2015), pp. 54–70.

[25] 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. In *Proceedings of the ACM SIGCOMM 2010 Conference on Data Communication* (2010), pp. 339–350.

[26] GAO, P. X., NARAYAN, A., KARANDIKAR, S., CARREIRA, J., HAN, S., AGARWAL, R., RATNASAMY, S., AND SHENKER, S. Network Requirements for Resource Disaggregation. In *Proceedings of the 12th USENIX Symposium on Operating Systems Design and Implementation* (2016), pp. 249–264.

[27] GHOBADI, M., MAHAJAN, R., PHANISHAYEE, A., DEVANUR, N., KULKARNI, J., RANADE, G., BLANCHE, P.-A., RASTEGARFAR, H., GLICK, M., AND KILPER, D. ProjecToR: Agile Reconfigurable Data Center Interconnect. In *Proceedings of the ACM SIGCOMM 2016 Conference on Data Communication* (2016), pp. 216–229.

[28] GREENBERG, A., HAMILTON, J. R., JAIN, N., KANDULA, S., KIM, C., LAHIRI, P., MALTZ, D. A., PATEL, P., AND SENGUPTA, S. VL2: A Scalable and Flexible Data Center Network. In *Proceedings of the ACM SIGCOMM 2009 Conference on Data Communication* (2009), pp. 51–62.

[29] HALPERIN, D., KANDULA, S., PADHYE, J., BAHL, P., AND WETHERALL, D. Augmenting Data Center Networks with Multi-gigabit Wireless Links. In *Proceedings of the ACM SIGCOMM 2011 Conference on Data Communication* (2011), pp. 38–49.

[30] 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 ACM SIGCOMM 2014 Conference on Data Communication* (2014), pp. 319–330.

[31] HE, K., KHALID, J., GEMBER-JACOBSON, A., DAS, S., PRAKASH, C., AKELLA, A., LI, L. E., AND THOTTAN, M. Measuring Control Plane Latency in SDN-enabled Switches. In *Proceedings of the 1st ACM SIGCOMM Symposium on Software Defined Networking Research* (2015), p. 25.

[32] HONG, C.-Y., KANDULA, S., MAHAJAN, R., ZHANG, M., GILL, V., NANDURI, M., AND WATTENHOFER, R. Achieving High Utilization with Software-driven WAN. In *Proceedings of the ACM SIGCOMM 2013 Conference on Data Communication* (2013), pp. 15–26.

[33] HOPPS, C. Analysis of an Equal-Cost Multi-Path Algorithm, 2000.

[34] KALIA, A., KAMINSKY, M., AND ANDERSEN, D. G. FaSST: Fast, Scalable and Simple Distributed Transactions with Two-sided (RDMA) datagram RPCs. In *Proceedings of the 12th USENIX Symposium on Operating Systems Design and Implementation* (2016), pp. 185–201.

[35] KASSING, S., VALADARSKY, A., SHAHAF, G., SHAPIRA, M., AND SINGLA, A. Beyond fat-trees without antennae, mirrors, and disco-balls. *Proceedings of the ACM SIGCOMM 2017 Conference on Data Communication* (2017).

[36] KLIMOVIC, A., KOZYRAKIS, C., THERESKA, E., JOHN, B., AND KUMAR, S. Flash storage disaggregation. In *Proceedings of the Eleventh European Conference on Computer Systems* (2016), ACM, p. 29.

[37] LEGTCHENKO, S., CHEN, N., CLETHEROE, D., ROWSTRON, A., WILLIAMS, H., AND ZHAO, X. XFabric: A Reconfigurable In-Rack Network for Rack-Scale Computers. In *Proceedings of the 13th USENIX Symposium on Networked Systems Design and Implementation* (2016), pp. 15–29.

[38] LEGTCHENKO, S., LI, X., ROWSTRON, A., DONNELLY, A., AND BLACK, R. Flamingo: Enabling Evolvable HDD-based Near-Line Storage. In *Proceedings of the 14th USENIX Conference on File and Storage Technologies* (2016), pp. 213–226.

[39] LIU, H., LU, F., FORENCICH, A., KAPOOR, R., TEWARI, M., VOELKER, G. M., PAPEN, G., SNOEREN, A. C., AND PORTER, G. Circuit Switching Under the Radar with REACToR. In *Proceedings of the 11th USENIX Symposium on Networked Systems Design and Implementation* (2014), pp. 1–15.

[40] LIU, V., ZHUO, D., PETER, S., KRISHNAMURTHY, A., AND ANDERSON, T. Subways: A Case for Redundant, Inexpensive Data Center Edge Links. In *Proceedings of the 11th ACM Conference on Emerging Networking Experiments and Technologies* (2015), pp. 27:1–27:13.

[41] MARANDI, P. J., GKANTSIDIS, C., JUNQUEIRA, F., AND NARAYANAN, D. Filo: Consolidated Consensus as a Cloud Service. In *Proceedings of the 2016 USENIX Conference on Usenix Annual Technical Conference* (2016), pp. 237–249.

[42] MITCHELL, C., GENG, Y., AND LI, J. Using one-sided rdma reads to build a fast, cpu-efficient key-value store. In *USENIX Annual Technical Conference* (2013), pp. 103–114.

[43] PORTER, G., STRONG, R., FARRINGTON, N., FORENCICH, A., CHEN-SUN, P., ROSING, T., FAINMAN, Y., PAPEN, G., AND VAHDAT, A. Integrating Microsecond Circuit Switching into the Data Center. In *Proceedings of the ACM SIGCOMM 2013 Conference on Data Communication* (2013), pp. 447–458.

[44] 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), pp. 123–137.

[45] SCHMIDTKE, K. Facebook Network Architecture and Its Impact on Interconnects. In *Proceedings of 23rd Annual Symposium on High-Performance Interconnects* (2015), IEEE.

[46] SCOTT, C., WUNDSAM, A., RAGHAVAN, B., PANDA, A., OR, A., LAI, J., HUANG, E., LIU, Z., EL-HASSANY, A., WHITLOCK, S., ET AL. Troubleshooting Blackbox SDN Control Software with Minimal Causal Sequences. *ACM SIGCOMM Computer Communication Review 44*, 4 (2015), 395–406.

[47] SINGH, A., ONG, J., AGARWAL, A., ANDERSON, G., ARMISTEAD, A., BANNON, R., BOVING, S., DESAI, G., FELDERMAN, B., GERMANO, P., KANAGALA, A., PROVOST, J., SIMMONS, J., TANDA, E., WANDERER, J., HÖLZLE, U., STUART, S., AND VAHDAT, A. Jupiter Rising: A Decade of Clos Topologies and Centralized Control in Google's Datacenter Network. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication* (2015), pp. 183–197.

[48] SINGLA, A. Fat-FREE Topologies. In *Proceedings of the 15th ACM Workshop on Hot Topics in Networks* (2016), pp. 64–70.

[49] SINGLA, A., HONG, C.-Y., POPA, L., AND GODFREY, P. B. Jellyfish: Networking Data Centers, Randomly. In *Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation* (2012), vol. 12, pp. 17–17.

[50] SINGLA, A., SINGH, A., AND CHEN, Y. OSA: An Optical Switching Architecture for Data Center Networks with Unprecedented Flexibility. In *Proceedings of the 9th USENIX Symposium on Networked Systems Design and Implementation* (2012), pp. 239–252.

[51] THERESKA, E., BALLANI, H., O'SHEA, G., KARAGIANNIS, T., ROWSTRON, A., TALPEY, T., BLACK, R., AND ZHU, T. IOFlow: a Software-defined Storage Architecture. In *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles* (2013), pp. 182–196.

[52] VALADARSKY, A., SHAHAF, G., DINITZ, M., AND SCHAPIRA, M. Xpander: Towards optimal-performance datacenters. In *CoNEXT* (2016), pp. 205–219.

[53] WANG, A., GUO, Y., HAO, F., LAKSHMAN, T., AND CHEN, S. Scotch: Elastically scaling up sdn control-plane using vswitch based overlay. In *Proceedings of the 10th ACM International on Conference on Emerging Networking Experiments and Technologies* (2014), pp. 403–414.

[54] WANG, G., ANDERSEN, D. G., KAMINSKY, M., PAPAGIANNAKI, K., NG, T. E., KOZUCH, M., AND RYAN, M. c-Through: Part-time Optics in Data Centers. In *Proceedings of the ACM SIGCOMM 2010 Conference on Data Communication* (2010), pp. 327–338.

[55] XIA, Y., SCHLANSKER, M., NG, T. S. E., AND TOURRILHES, J. Enabling Topological Flexibility for Data Centers Using OmniSwitch. In *Proceedings of the 7th USENIX Workshop on Hot Topics in Cloud Computing* (2015), pp. 4–4.

[56] XIA, Y., SUN, X. S., DZINAMARIRA, S., WU, D., HUANG, X. S., AND NG, T. S. E. A tale of two topologies: Exploring convertible data center network architectures with flat-tree. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication* (2017), pp. 295–308.

[57] ZHOU, X., AND NISHIZEKI, T. Edge-coloring and f-coloring for various classes of graphs. *Journal of Graph Algorithms and Applications 3*, 1 (1999), 1–18.

[58] 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.

[59] ZHU, Y., ERAN, H., FIRESTONE, D., GUO, C., LIPSHTEYN, M., LIRON, Y., PADHYE, J., RAINDEL, S., YAHIA, M. H., AND ZHANG, M. Congestion Control for Large-scale RDMA Deployments. In *ACM SIGCOMM Computer Communication Review* (2015), vol. 45, ACM, pp. 523–536.

[60] ZILBERMAN, N., GROSVENOR, M., POPESCU, D. A., MANIHATTY-BOJAN, N., ANTICHI, G., WÓJCIK, M., AND MOORE, A. W. Where Has My Time Gone? In *International Conference on Passive and Active Network Measurement* (2017), Springer, pp. 201–214.