Subways: A Case for Redundant, Inexpensive Data Center Edge Links

Vincent Liu, Danyang Zhuo, Simon Peter, Arvind Krishnamurthy, Thomas Anderson
{vincent, danyangz, simpeter, arvind, tom}@cs.washington.edu
University of Washington

ABSTRACT

As network demand increases, data center network operators face a number of challenges including the need to add capacity to the network. Unfortunately, network upgrades can be an expensive proposition, particularly at the edge of the network where most of the network's cost lies.

This paper presents a quantitative study of alternative ways of wiring multiple server links into a data center network. In it, we propose and evaluate Subways, a new approach to wiring servers and Top-of-Rack (ToR) switches that provides an inexpensive incremental upgrade path as well as decreased network congestion, better load balancing, and improved fault tolerance. Our simulation-based results show that Subways significantly improves performance compared to alternative ways of wiring the same number of links and switches together. For example, we show that Subways offers up to 3.1× better performance on a MapReduce shuffle workload compared to an equivalent capacity network.

CCS Concepts

•Networks → Data center networks;

Keywords

Data center network; Datacenter fabric

1. INTRODUCTION

Keeping pace with ever increasing network demand is an ongoing challenge for data center operators. Network demand is surging at a rapid rate due to faster and more capable servers, increasing application network-to-compute ratios [17], and kernel bypass techniques [8, 26]. For example, Google recently reported that it increased the bisection bandwidth of its data center networks by three orders of magnitude between 2004 to 2012, on average doubling every 10 months [32]. Making matters worse, the network is a large and growing portion of the total cost of the data center [17]. Because many data center applications are highly sensitive to tail latencies, networks must be configured with relatively low average link utilization, further increasing costs.

Operators often prefer an incremental approach to adding capacity while the existing network continues to carry traffic [7, 32]. While it is also possible to take down the data center and forklift in a new faster network, this process can require extensive downtime. Instead, adding multiple network links per server has become one way to support upgrades. In principle, a network operator could double capacity by doubling the amount of network hardware, wiring each server in parallel to dual Top-of-Rack (ToR) switches; those switches in turn can be wired in parallel to a replicated aggregation layer, and so forth.

In this paper, we present the counterintuitive result that it is possible to achieve better than a proportional performance improvement when upgrading a data center network for typical workloads. In other words, a doubling of network capacity can result in much better than a 2× performance improvement on the same hardware. A key insight is that nearby servers exhibit communication locality, where physically co-located servers often communicate at the same time, both with each other and with the rest of the network [9, 22, 31], making a denser local interconnect attractive.

Our approach, Subways, is a co-design of the network wiring, routing, and load balancing algorithms when there are multiple links per server. Instead of always wiring servers to switches within a physical rack, we cross-wire some server links to the ToR switches of adjacent racks in an overlapping pattern. We similarly cross-wire ToR switches of adjacent racks into different locations in the aggregation layer. This has four benefits. First, we can arrange a larger number of nodes to be reachable within a locality region, e.g., to be able to directly communicate through a single ToR switch, thereby reducing traffic load and congestion at the aggregation layer. Second, by wiring correlated servers into distinct parts of the data center network fabric, we achieve better statistical multiplexing properties than more basic wiring pat-

CoNEXT '15, December 01-04, 2015, Heidelberg, Germany.

ACM 978-1-4503-3412-9/15/12.
http://dx.doi.org/10.1145/2716281.2836112

terns. Third, in our topology, differential load balancing becomes more effective than in a standard FatTree topology, allowing Subways to reduce hotspots at the ToR and aggregation switch layer. Finally, our topology allows servers to detour traffic to remote ToRs without using the data center backbone. Because of this, servers can utilize the uplink capacity of many ToRs—not just their own.

Our goal is to characterize the space of Subways-style topologies and show how addressing, routing, and load balancing can be accomplished in these networks. The options offer a range of tradeoffs in complexity and performance, and we aim to quantify the performance benefits of the different choices. Using a simulation-based methodology, we show that Subways offers substantial performance benefits for popular application workloads: up to a 3.1× speedup in MapReduce and a 2.5× throughput improvement in memcache for a fixed average request latency, relative to an equivalent-bandwidth network that differs only in its wiring.

Of course, network cost is as important as network performance. It is beyond the scope of the paper to fully characterize the installation and operational costs of a Subways network. In practice, these costs depend on proprietary volume discounts for optical and electrical cables, switches, network interfaces, etc. However, the largest added cost for Subways is likely to be the added cable length from the server to the adjacent ToR switches. We develop a set of wiring algorithms that show that physical wire lengths can be kept short enough to be implemented with copper and also made relatively easy to install.

2. BACKGROUND AND MOTIVATION

Data centers can vary greatly from one deployment to another, sometimes even within the same company. Even so, there are several features that are common among most state-of-the-art data centers. These best practices are necessitated by scalability, cost, and practical limitations.

The first is that servers and the network switches that connect them are stacked in physical racks to simplify installation and management. The second is that data center network switches are constructed using mass-produced merchant silicon switching chips, each with only a tiny fraction of the required network switching capacity. These chips are packaged to provide various port counts and speeds, subject to the constraints of the underlying chip. Because these building blocks have limited capacity, the third common feature is a multi-level, multi-rooted tree of switches. Typically, this tree is some variant of a Folded-Clos topology [5, 14]. The canonical version has three layers: (1) ToR switches, which connect a single physical rack of several dozen servers, (2) aggregation switches, which connect the ToRs of tens of racks into a single cluster, and (3) core switches, which connect the clusters of the data center together. Each layer has many physical switches operating in parallel.

Typically, inexpensive electrical cables are used between servers and ToRs, even for the high-bandwidth connections needed by modern servers. In addition, some data center networks are wired with a single, fast link between each server and the ToR switch; others wire links in parallel from each server to the ToR switch, using Link Aggregation Groups (LAGs) to spread packets as evenly as possible across the links and allowing the parallel links to behave as a virtual fast link. Likewise, two or more ToR switches with identical down and up connectivity can be grouped together into a larger virtual switch using Multi-Chassis LAGs (MC-LAGs).

More expensive fiber optic links are used in the network fabric: from ToR nodes to the aggregation layer, and from there to the core. This is because electrical signalling is not feasible for cables that need to span the data center, or even a cluster. Because optical connections are expensive, the tree is often thinned immediately above the ToR level. That is, the aggregate bandwidth from servers into a ToR is often a small multiple of that of the ToR's optical uplinks. Higher levels of the tree are often even more oversubscribed, again for cost reasons [14, 32].

2.1 Inter-rack Communication is Growing

With oversubscription at higher levels of the tree, one approach to job scheduling is to keep communication local where possible. For instance, if we could pack all nodes from a MapReduce job into a single rack, shuffle traffic would never need to traverse the oversubscribed core network. Indeed, this technique can be quite successful—75% of measured traffic in one network stayed within a rack [9].

Despite this, inter-rack communication is often unavoidable. Sometimes, this distribution is by design, e.g., network storage blocks are often stored across power failure domains for fault tolerance. More commonly, jobs are often too large to fit within a single rack [31]. In a recent Google trace [29], 63% of the workload was for jobs that required multiple racks, even assuming optimal bin-packing. The increased prevalence of large analytics implies that inter-rack communication will continue to grow at a very fast pace [17].

2.2 Server Traffic is Often Correlated

Data center traffic is very bursty, particularly on links closer to the edge [6, 9, 16]. This is partly a consequence of the desire to preserve traffic locality. To reduce inter-rack communication, multiple servers in the same rack can be assigned to the same job, but by virtue of performing related tasks, when they do send inter-rack traffic, they are more likely to do so simultaneously.

A second reason is that servers/jobs with similar purposes are often placed in the same rack. Facebook, for example, expands its memcache and web frontend infrastructure by rolling out an entire rack of servers that are optimized for that particular service [25]. Although this approach significantly reduces operational complexity, it implies that the traffic patterns of servers within a rack can be correlated; as traffic increases, all of them get hot simultaneously.

Correlated server behavior is a problem because of over-

subscription. The aggregate capacity of the uplinks from the ToR switch define how much the servers in that rack can send or receive at any time, gating application performance.

2.3 ToRs Are a Single Point of Failure

In addition to demand growth, the network is also becoming a fault tolerance bottleneck. Data center operators go to great lengths to ensure that their systems are resilient to failures. In fact, it is common to see techniques such as redundant power supplies, a great deal of network path diversity, and even redundant SDN controllers. In most data centers (e.g., those without MC-LAGs), there is no such redundancy for the ToR switch—it remains as one of the few remaining single points of failure in the data center. This is particularly important as ToRs have relatively high failure rates compared to other network devices [13].

3. DESIGN OVERVIEW

We introduce Subways, a family of data center edge interconnect architectures that use redundant links between servers and ToRs. Common among these designs is a wiring pattern where servers in a rack are wired to adjacent ToRs in addition to their own.[1] Clever usage of multiple server-ToR connections can mitigate many fundamental issues in today's data centers:

- *Simpler upgrades:* A typical edge capacity upgrade requires large amounts of up-front investment and/or rewiring of both server-ToR links and links in the backbone. By augmenting connectivity and allowing servers to connect to adjacent ToRs, we enable cheap, potentially incremental upgrades that reduce or eliminate the need for rewiring.
- *Less backbone traffic:* An overlapping connection pattern creates shorter paths for more destinations, keeping traffic off the data center backbone.
- *Smoother hot spots:* Traffic is very bursty, particularly at the ToRs. By connecting servers to multiple, differently-loaded ToRs and clusters rather than just one, we gain better load balance.
- *Fault tolerance:* Redundant server-ToR links remove one of the remaining single-points-of-failure in data centers.

The Subways design varies on two dimensions, which we discuss in the following sections. First is topology: how are the server links distributed among ToRs, and how are those ToRs distributed into clusters? These choices have a large impact on reducing and smoothing traffic at each layer. The second dimension is load balancing: Subways can work with uniform load balancing mechanisms like ECMP, but it can also benefit from adaptive mechanisms that shift traffic away from overloaded ToR switches and from detour routing, which shunts traffic to remote ToRs.

[1] In our work, different servers in the same *physical rack* can be wired to different sets of ToR switches. We refer to the set of servers wired identically as a *logical rack*, and we use the term rack to refer to a logical rack unless otherwise specified.

Var.	Definition
N	# of end hosts in the data center
p	# of ports per server
q	# of downward facing ports per ToR switch
r	# of servers per logical rack ($\frac{q}{p}$)
c	# of clusters over which a loop is mapped (Type 2)
l	# of racks in a single Subways loop (Types {1,2})

Table 1: The variables that define a Subways architecture. Some only apply to a subset of the wiring methodologies in Sec. 4.

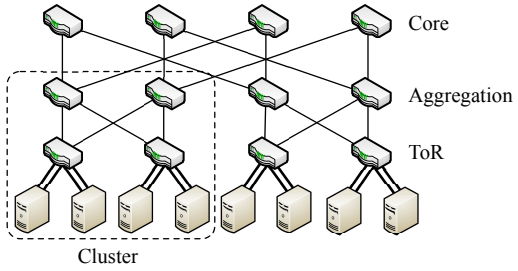


Figure 1: A FatTree with Type 0 Subways, $p = 2$. Three layers of switches connect servers together. Each server has two ports that are both connected to the same ToR switch.

As every data center instantiation may have different numbers of links per server or servers per rack, we parameterize our discussion using the notation sketched in Table 1. Because Subways is an edge architecture, it is compatible with *any* aggregation and core topology, although for concreteness we concentrate on FatTrees in this paper.

4. WIRING TYPES

We begin by describing a baseline topology, Type 0 Subways, corresponding to the current, industry-standard approach to using a p-port server where all ports are connected to the same switch. We then introduce two new wiring types that link servers to adjacent ToRs. To keep the discussion concrete, we assume for now that all topologies use a simple ECMP-like load balancing algorithm; we relax that assumption in the next section. We defer a discussion of the implications of Subways for physical cable lengths until Sec. 7.

4.1 Starting Point: Type 0 Subways

The simplest wiring model is to trunk p connections from each server to the same ToR switch, as shown in Fig. 1. A Link Aggregation Group (LAG) can be used to treat the multiple physical links as a single logical connection using a protocol like LACP. This simplifies routing but requires the entire trunk to terminate at a single, physical switch.

As a result, although the network core has ample redundant paths for load balancing and fault tolerance, all paths to and from a rack feed through a single ToR switch. Likewise, the server itself has redundant links, but if the switch fails or is overloaded, all p connections suffer.

When using Type 0 Subways as an upgrade path, operators are required to rewire many existing connections. For instance, let us assume she wants to double capacity while keeping the number of servers (N_s) and oversubscription ratio constant. Any such upgrade requires N_s new server-ToR

connections and double the number of switches in the network. In addition, Type 0 requires her to rewire $\frac{N_s}{2}$ of the existing server-ToR links so that all links from a single server go to the same switch, and then to rewire/expand the existing interconnect to ensure that new ToRs have connectivity to the old ones.

Perhaps most importantly, most trunking protocols are limited to links of the same speed. This prevents heterogeneous upgrades, e.g., where a 25 Gbps link is added to an existing 10 Gbps server connection, something allowed with the wiring patterns we discuss next.

4.2 Type 1: Shared ToRs Within a Cluster

One way to improve fault tolerance is to wire the p ports of each server to p different ToR switches at the top of the physical rack. With a multihoming approach like MC-LAG, these multiple physical switches can be aggregated into a single, virtual ToR. To make routing and failover completely transparent, the physical switches are typically wired identically into the network fabric.

Instead, we propose to wire each group of servers to a distinct, overlapping set of nearby ToR switches. In doing so, we gain an extra degree of freedom and, as we describe later, considerable performance improvement relative to both Type 0 and multihoming. For example, with $p = 2$, we wire each server to its ToR plus the one for the closest (logical) rack to the left. In this way, neighboring logical racks share at least one ToR. With Type 1, each server is connected to ToR switches that connect into the same aggregation-level cluster; we relax this assumption for Type 2.

This simple act of sharing ToRs has far-reaching benefits. Using $p = 2$ as an example, two adjacent logical racks are connected to three ToRs, instead of just two. Each server has a high-capacity path to 50% more servers than before, improving performance and reducing the traffic reaching the oversubscribed data center network. When those servers do need to send data through the backbone, they have, in aggregate, 50% more peak throughput than with Type 0.

Topology. Conceptually, each logical rack still has an associated ToR. However, instead of connecting servers to their own ToR p times, servers connect to their own ToR and their $p-1$ closest ToRs exactly once. This overlapping chain of racks and ToRs is wrapped around to eventually form a *loop* of l racks/ToRs, where each server is connected to its own ToR, the $\lfloor \frac{p-1}{2} \rfloor$ ToRs clockwise, and $\lceil \frac{p-1}{2} \rceil$ ToRs counterclockwise from it. A loop is thus a connected component in the server-ToR topology. In the degenerate case where the number of server links equals the loop size, $p = l$, we have multihoming. We show that performance improves with increased loop size, but there are practical limitations due to physical wiring constraints that we discuss in Sec. 7. For simplicity, we assume a single loop length l for all loops in the data center. Fig. 2 shows a single loop of 3-port servers and a length of 9. Fig. 3a shows two loops where the loops are the same size as the clusters.

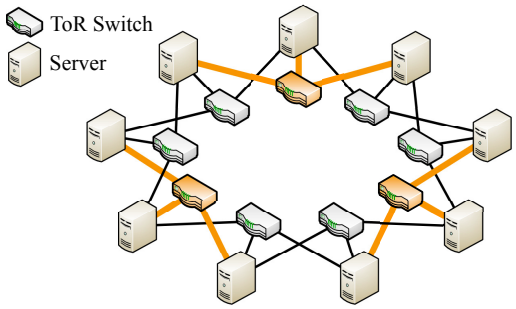


Figure 2: Example of the server-ToR connection in Type 1 with a single loop of 9 3-port servers. Bold links/switches are added to upgrade from a 2-port to a 3-port configuration without rewiring.

Upgrades. One of the interesting things about Subways-style server-ToR connections is that adding capacity to servers does not require rewiring their existing connections. Fig. 2 shows an example of this, where an upgrade from 2 ports to 3 requires adding new ToRs, but leaves the existing ToR connections untouched. Note, however, that to make room for the added ToRs, an operator may still need to rewire the upper layers of the data center network to ensure that all of a server's ToRs are contained within the same cluster.

Routing. In vanilla Type 1, we keep routing similar to today's data centers, implemented entirely with existing protocols. In particular, because all of a server's ToRs are contained within a single cluster, routing in most of the network does not change—traffic to a particular server still flows to a single cluster, typically through longest-prefix matching.

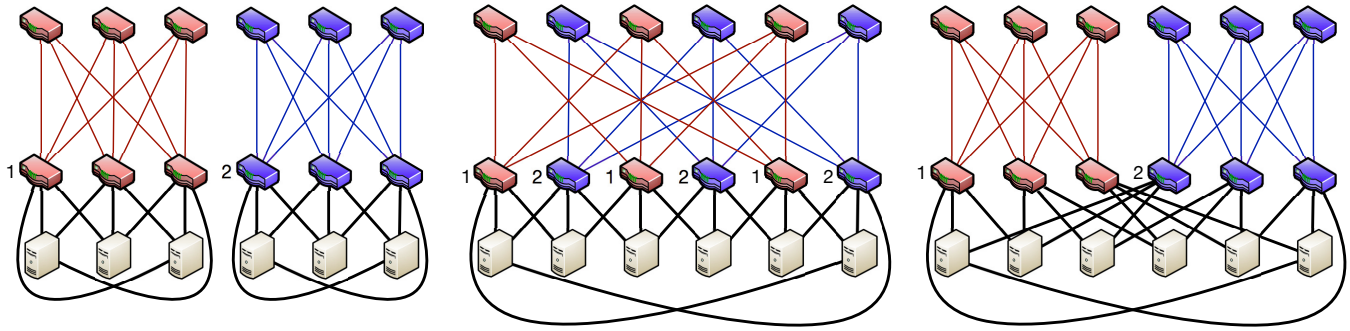
Servers and aggregation switches have an additional responsibility to assign connections randomly to the available ToRs. This can be done using ECMP [33]. In ECMP, routing table entries can map to multiple possible output ports, to which connections are randomly assigned. Assignment is done on a per-connection basis to avoid packet reordering.

Only nodes that are directly connected to ToRs (i.e., servers and aggregation switches) need to install extra ECMP rules. Servers only need one such rule with p options: one for each ToR. Aggregation switches need t rules with p options each, where t is the number of racks in its cluster; they do not need any rules for traffic destined for other clusters beyond what is needed today.

4.3 Type 2: ToRs in Different Clusters

Subways provides one more degree of freedom: adjacent ToR switches in the same loop can be wired into different clusters. In a traditional multi-rooted tree, there is no benefit to declustering. In our case, however, it can spread load more evenly across clusters, and it increases the number of servers that can be reached without going through the core layer. These shortcuts allow a greater degree of oversubscription, for a given level of performance, or equivalently, less congestion for those jobs whose traffic must traverse the core.

Topology. Every server in a Type 2 Subways is connected to multiple clusters as evenly as possible. The number of clusters that are connected to a single loop is configurable



(a) Type 1 (b) Type 2 (c) Type 2 (alternate representation)

Figure 3: A two cluster topology for Type 1 (a) and Type 2 (b, c) with $p = 3$ and $l = 3$ and 9, respectively. We omit the core layers and color/number ToRs according to their cluster.

and depends on factors such as how much mixing is needed, loop length, and physical constraints.

Let us assume that we have c clusters whose ToRs should be connected to a single Subways loop. In order to find a mapping between the ToRs in Fig. 2 and the clusters' ToRs, pick an arbitrary ToR and assign it to the first cluster. Move to the next ToR in clockwise order and assign it to the second cluster. Continue assigning each ToR to a cluster in this manner, looping back to cluster 1 after assigning a ToR to cluster c. Note that the position within a cluster does not matter as all ToRs within a cluster are logically equivalent. It also does not matter that a server is not necessarily connected to all c clusters, as long as assignment is as even as possible.

Fig. 3b and Fig. 3c show two equivalent examples where $c = 2$. Note that servers still only connect to nearby ToRs (Fig. 3b), and at the same time, the cluster topology is still amenable to cable bundling (Fig. 3c)—we only change the interface between these two topologies.

Upgrades. Because we no longer require all of a server's ToRs to be in the same cluster, an upgrade in Type 2 does not actually require any rewiring of existing links. In the case of augmentation with a parallel backbone, we do not even need to touch the existing core layer. In the absence of rewiring, it is possible to perform a capacity upgrade without any disruption in service. Further, the upgrades can be heterogeneous.

As an example, in an entirely 10 GbE data center, an operator could choose a single cluster of servers and augment them with a parallel 25 GbE FatTree, cross-wired so that each logical rack of servers connects to a different, overlapping pair of 10 GbE and 25 GbE ToR switches. Instead of wiring every server in a physical rack to the same two 10 and 25 GbE switches, we wire half of the servers to one 25 GbE switch (at the top of the physical rack) and half to the 25 GbE switch at the top of the neighboring rack.

Addressing/Routing. Regardless of whether servers are connected to parallel interconnects or different clusters in the same interconnect, the networks are configured as they are today. Each interconnect has its own address space, and each of its clusters has a subnet within that space. Routing tables can be set up for longest-prefix matching with no increase in size. The primary change is that each server has p addresses.

Servers spread flows over possible paths using a simple, static version of ECMP or WCMP [36]. For example, servers with parallel 10 and 25 GbE connections would place flows on those networks with probability $^2/_7$ and $^5/_7$, respectively.

5. ADAPTIVE LOAD BALANCING

Wiring servers to ToRs in an overlapping pattern is by itself enough to provide significant performance benefits, but it also opens up the possibility of more advanced load balancing. In this section, we introduce two load balancing mechanisms; they can be combined with either Type 1 or Type 2 topologies. For simplicity, we assume $l > p$ in this section.

As a motivating example, consider the case where $p = 2$ and two adjacent racks are simultaneously hot—a situation that could be more likely as jobs are placed to take advantage of short paths in Subways. With ECMP, the shared ToR between the racks would have twice the load of the two non-shared ToRs. This hurts tail latency: flows placed on the shared ToR take twice as long as the other flows. Using adaptive load balancing, we can equalize utilization across all three ToRs. Note that with Type 0, ToR switches can become overloaded, but load balancing does not help: all paths to the same set of servers lead through the same ToR, and in the absence of a switch or link failure, all paths are identical.

We introduce two possible solutions: a multipath transport-layer protocol or controller-based scheduling. We describe both approaches, but our evaluation assumes the latter.

5.1 Multipath Transport-layer Protocols

Transport-layer protocols like MPTCP [35] allow each end host to utilize multiple paths through the network to maximize resource usage. Like TCP, MPTCP detects congestion and varies the amount of traffic sent along a given path. However, the specific protocol is orthogonal to our design, as long as it is able to utilize multiple paths effectively.

In the context of Subways, a multipath transport-layer protocol would allow the system to adaptively balance load over all available paths. This is done in a distributed fashion at

each end host and thus can adapt to changes in load relatively quickly and at a fine granularity. In the example described above where adjacent racks are hot, a multipath protocol would let each server split all connections over both ToRs. Instead of a situation where some connections have half the throughput of others, every connection would receive an approximately equal portion of the available bandwidth—a 50% decrease in tail latency.

Exposing multiple paths to the servers. To support MPTCP, we need to expose the multiple paths to each server. For each server-server pair, there are up to p^2 combinations of ToRs through which a path can pass—p ToRs at the source and p at the destination. Note that we assume path diversity in the core interconnect is handled separately.

The source ToR is chosen by directing traffic out of the associated physical port. The destination ToR is chosen by giving each server p addresses, just as in Sec. 4.3. This is necessary regardless of whether the topology is Type 1 or 2; in Type 1, however, each address will come from the same subnet so that routing in the middle of the network remains the same.

5.2 Weighted-ECMP

An alternative approach that does not require changes to the end host TCP implementation is to use a locally adaptive variant of the previous work on Weighted-ECMP [4, 10, 23, 36]. In these proposals, senders periodically obtain current utilization information from a centralized controller and use that information to change the probability that flows are placed on a given next hop.

The problem is much simpler here. Because the ToR tends to be the bottleneck, we only need to look at ToR utilization (and not entire paths) when making traffic engineering decisions. Because of this, load balancing calculations are local; ToR utilization information can be maintained by a sharded set of controllers, rather than a single centralized server. This makes statistic collection and load balancing calculations faster and more efficient. Loops in particular represent a natural sharding boundary.

Initial flow placement. Like traditional FatTree routing, the source will prefer shorter paths if available. If there are one or more paths that pass through a shared ToR (and avoid using the data center backbone), the source will choose one of them with equal probability. Otherwise, it must route the flow through one of its ToRs and one of the destination's ToRs. Each is chosen independently and the decision is based on current congestion information. The source ToR is always chosen by the source server, while the destination ToR is chosen by the destination-side aggregation switch or source server in Type 1 and Type 2 topologies, respectively.

Setting weights for source ToRs. The probability that a flow is placed on any particular ToR is based on the remaining capacity of that ToR. For the choice of source ToR, we care about the remaining capacity in the outbound direction.

More formally, each server obtains from its loop's controller the values $[U_1,U_2,..,U_p]$. These represent, for each of the server's ToRs, an exponentially-weighted moving average of the fraction of the ToR's uplinks that are utilized. From utilization, we can calculate the remaining capacity on each ToR: $V_x = B_x(1-U_x)$, where B_x is the total uplink capacity of ToR x. The probability of placing a new flow on ToR i is then simply $\frac{V_i}{\sum_{j=1}^{p} V_j}$.

Setting weights for destination ToRs. The main difference between choosing a destination ToR and a source ToR is that we care about *inbound* traffic, rather than outbound. In particular, for each destination, we have $[D_1,D_2,..,D_p]$, the per-ToR downlink utilization of the Aggregation-ToR links.

The resulting weights for a server's ith ToR is $\frac{E_i}{\sum_{j=1}^{p} E_j}$, where E_x is the remaining downlink capacity, $B_x(1-D_x)$. In Type 2, these weights are provided to each server for each communication partner. In Type 1, they are provided to each aggregation switch for their own cluster. Alternatively, these could be implemented using pushback mechanisms like pause frames and ECN.

Subsequent load balancing. We periodically reschedule existing flows to adapt to changing traffic conditions in the presence of long-lived flows. Each end host and/or aggregation switch will rebind all existing connections every T_{rebind}.

The first step in this process is to calculate a target utilization for each ToR, which for the outgoing direction, is $U_{target} = \frac{\sum_{i=1}^{p} B_i U_i}{\sum_{i=1}^{p} B_i}$, i.e., the total amount of traffic divided by the total bandwidth. With these targets in mind, a server will shift its current utilization to achieve a more even split.

Specifically, let $[u_1,u_2,..,u_p]$ be the current fraction of traffic the server is sending over each ToR. Based on the target ToR utilization and the server's current split, it will calculate a target split for its own traffic. Toward ToR i, its target is $t_i = cu_i\frac{U_{target}}{U_i} + (1-c)u_i$, where c is a scaling factor that determines the aggressiveness our algorithm. In our evaluation, we use $c = 0.5$.

The probability that a flow is rebound to the ith ToR is then $P[i] = \frac{t_i}{\sum_{j=1}^{p} t_j}$. Note that the target traffic split might not be possible due to the bandwidth of the server-ToR link, b_i. Ignoring short paths for simplicity, this happens when $P[i](\sum_{j=1}^{p} u_j b_j) > b_i$. In this case, we set $P[i] = \frac{b_i}{\sum_{i=1}^{p} u_i b_j}$ and spread the excess load evenly over the remaining links. Also note that reordering can be prevented by waiting for flowlet inactivity gaps [20].

6. DETOUR ROUTING

An interesting side effect of the Subways wiring pattern is that each loop is a connected graph even without the use of the inter-ToR network. Because of this, whenever a server's ToRs are overloaded, it is possible for servers to detour traffic through adjacent racks (using only Subways links) in order to shunt excess load to remote ToRs.

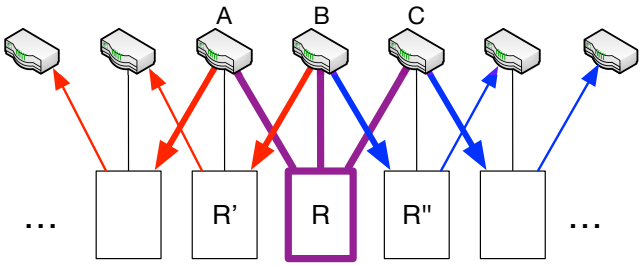


Figure 4: A rack-level diagram showing the detour paths of a single hot rack of servers R. Boxes represent racks. Colors differentiate the direction of the detour (red for left, blue for right).

As an extreme example, consider the topology in Fig. 4. Assume that rack R has 20 servers and all of them wish to fully utilize their three 10 GbE links. Despite R's 600 Gbps of demand, its ToRs A, B, and C have only 40 Gbps of total uplink capacity apiece—a 15:1 oversubscription ratio. Even with adaptive load balancing, the ToRs can satisfy at most $^1/_5$th of R's offered load. If R instead bounced traffic through adjacent racks, it could enlist its neighbors to handle all 600 Gbps. Here, some portion of the offered load would be sent through A, B, and C while the remainder is detoured away from the loaded racks, i.e., through R' and R''. This happens recursively, with each successive ToR egressing a portion of the remaining traffic until all of it is handled.

In the common situation where just a single rack is hot, the only limit to the amount of traffic we can detour is the rack's NIC capacity. In other words, if all server-ToR links have the same capacity, detouring can, in principle, achieve full burst bandwidth to/from the loaded rack *regardless of the network's oversubscription ratio*.

Design overview. To ensure that detours are both efficient and do not interfere with adjacent servers, we rely on a per-loop scheduler in the same mold as the one in Sec. 5. These can be one in the same. When a group of servers is bottlenecked by oversubscription in the network, the scheduler can allocate a set of detour paths for them. These detours can extend to as many remote ToRs as necessary, limited in 3 ways:

- *Loop size:* detours can only use remote ToRs in the same loop.
- *Other hotspots:* for simplicity, multiple distinct hotspots are not allowed to detour through the same remote ToR.
- *Path dilation:* operators can limit the associated increase in latency and additional server utilization by imposing a cap on path length.

The scheduler will provide each server with an equal share of the available capacity, but will limit the usable uplink capacity of each remote ToR based on their current utilization. In our experiments, we set the usable capacity to be 60% of the remaining bandwidth (before any detours) so as to provide sufficient headroom for momentary spikes in traffic.

Note that for simplicity, we focus on outbound detours in this section; the process for inbound detours is similar. Also note that we start by assuming isolated hotspots and discuss groups of adjacent, hot racks at the end of this section.

When to detour. The scheduler initiates a detour whenever there is persistent congestion at the ToR level. Persistent congestion is measured by gathering pre-detour ToR utilization statistics every T_{detect} ms. Whenever a ToR is using $> 60\%$ of its total uplink capacity (incoming or outgoing) over an entire period, it is considered "hot" for a minimum of $T_{duration}$ ms. If all of a source rack's ToRs are hot in the outgoing direction, the rack is considered "hot" and warrants detour paths. Likewise, when all of a destination rack's ToR uplinks are hot in the incoming direction, the scheduler will allocate detour paths and notify the sources of the traffic. Notifications continue to arrive every T_{detect} ms until the congestion subsides.

Computing detour paths. As an example, consider Fig. 4. In this case, we have a hot rack of servers and wish to shunt traffic through neighboring racks. To shunt traffic left, neighboring racks will take traffic from their rightmost $h = \lfloor \frac{p}{2} \rfloor$ ToRs and forward it along their leftmost h ToRs. Similarly, neighboring ToRs will take traffic from their rightmost h connected racks and forward it to their leftmost h racks. Rightward detours proceed in a similar fashion. Each intermediate ToR egresses a portion of the detoured traffic until all of it exits the Subways server-ToR network.

More concretely, from the ToR utilizations, a per-loop scheduler should be able to deduce the set of overloaded racks and the uplink utilization of every ToR in the loop. A hot rack of servers will manifest itself as p contiguous, heavily-utilized ToRs. From this information, it can determine the set of rack-level detour paths and the usable capacity on each:

1. Consider each group of hot ToRs.
2. Take the 2 ToRs on either side of the set under consideration. These ToRs will be used as egress points for some number of detour paths.
3. If one of the ToRs is already being used for a detour, do not consider it or any further ToRs in that direction.
4. Otherwise, for each of these edge ToRs, calculate the set of detour paths that should use the ToR's spare uplink capacity. We can do this by iteratively backtracing the paths and pruning based on a maximum path length. Specifically, the left ToR should be fed by its rightmost $\lfloor \frac{p}{2} \rfloor$ connected racks, which should in turn be fed by their rightmost $\lfloor \frac{p}{2} \rfloor$ connected ToRs. The detours are allowed to use a total of 60% of the measured residual capacity. The paths split this usable capacity equally.
5. Repeat 2-4 for every group of hot ToRs.
6. Repeat 2-5 until we either run out of capacity on the original sources' server-ToR links or no additional ToRs are under consideration.

Servers source route along these paths using recursive encapsulation with headers that specify the next hop in reverse order. Forwarding can be implemented using software switches that are increasingly common (e.g., Open vSwitch [27]) and with NICs that are beginning to include similar functionality for forwarding among VMs [18]. While these

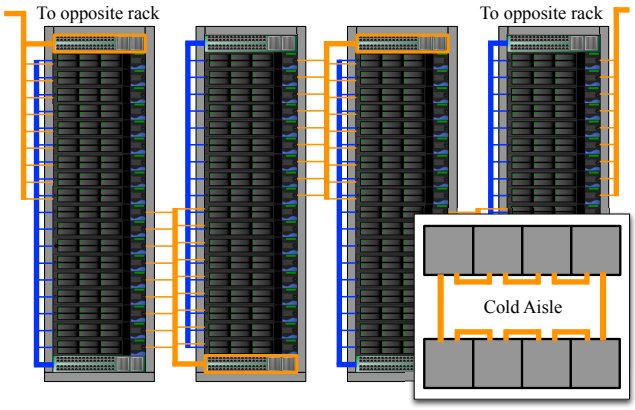


Figure 5: An example physical topology for Type 1 and up. Blue links are part of the initial 1-port configuration. Orange links and outlined switches denote hardware that is added as part of the upgrade to 2 ports. The main image is the frontal view of a row of racks, while the inset is an aerial view of the entire Subways loop.

solutions are primarily aimed at providing a communication bridge among various virtual machines executing on a server, they also have the capability to switch among server NIC ports. In Sec. 8.8, we test the software overhead of forwarding and show that it is small.

Detouring for groups of racks. As in Sec. 5, shortcut-aware job scheduling policies can lead to cases where multiple adjacent racks are simultaneously loaded. The scheduler can detect this case when it notices that more than p contiguous ToR switches are heavily loaded. When this occurs, a couple of modifications must be made to the above algorithm.

The first is that not all servers can detour in all directions. For some, their detours would pass through other loaded racks, which would create contention. As a result of this restriction, only the $\lfloor \frac{p}{2} \rfloor$ leftmost racks in a contiguous group can detour traffic to the left, while the $\lfloor \frac{p}{2} \rfloor$ rightmost racks can detour traffic to the right.

The second modification concerns fairness. Because only the edges of the group can utilize detours, we can achieve a slightly more even division of capacity by forcing the edges—those racks that can utilize detours—to not send any traffic through directly connected ToRs that other members of the group could use. Doing so can improve tail job completion time for the entire group.

7. PHYSICAL CONSIDERATIONS

As with any topology, physical design considerations can affect the practicality of an architecture. While every data center instantiation has unique physical constraints, we present here one example of how a data center could be upgraded with Subways links while keeping wire lengths short and complexity low. Other wiring patterns (including one with much shorter wire lengths) and a more general discussion of physical concerns can be found in our tech report [1].

Upgrades. Fig. 5 shows an example physical topology. Blue links are the initial connections, while orange links and switches represent hardware that is added during the upgrade. This design takes advantage of the fact that, in a real data center, servers have multiple ToRs that are physically "adjacent"—not just those in the next rack in the row.

More concretely, this design starts simply: operators connect all servers in a rack to a ToR switch in the same rack. ToRs are placed at the top or bottom of the rack in an alternating fashion, with an empty 2U slot at the other end. When upgrading, the operator will install a new ToR in each rack that is used to connect either (1) adjacent racks within a row of racks or (2) racks that are separated by a cold aisle. In this way, an operator creates a Subways loop using two parallel rows of racks.This example is optimized for expansion to $p = 2$, but similar techniques can be used to handle higher port counts. With three or more switches per physical rack, it is possible to weave a loop along a single row of racks (without crossing any aisles) while still only using connections between adjacent racks [1].

Wire length. Assuming a rack height of 2 m and cold aisle width of 1.2 m [2], the longest intra-row wire is about 2 m. If we also assume we can place wires on both overhead trays and beneath a raised floor, then the length of the cross-aisle wires will be ~ 2.2 m. These lengths are not much more than is required for a single rack and, for all currently available link speeds, are easily implemented using copper cables.

Cabling Complexity. An advantage of this design is that the initial configuration requires little to no work beyond what is done today: racks can still be preconfigured and there are no cross-rack wires. The upgrade step requires some additional cable installation beyond what is traditional; however, this is such a structured, symmetric topology that we anticipate this complexity to be manageable. Further, it can be simplified by bundling the wires between racks into a single cable and connector in a manner similar to the pin headers in modern desktop computers.

8. EVALUATION

To evaluate Subways, we implemented the following:

- A packet-level simulator that we used to test medium to large deployments of Subways.
- A small Cloudlab [30] testbed to validate our simulator.
- A server detour implementation to test the feasibility of software-based detouring through servers.

In this section, we denote Subways variants with adaptive load balancing as Type 1LB and 2LB, while detour variants are denoted as Type 1D and 2D. Our evaluation attempts to illustrate the key aspects of Subways, but our results are limited to the specific workloads and parameters that we tested. A complete assessment of workload and parameter sensitivity is left for future work.

8.1 Simulator Implementation

We used a modified version of a packet-level simulator that we previously used in the evaluation of other systems [23, 28]. It implements both low-level switch behavior and all of

	Type 0	Type 1
Testbed	179.1 s	47.5 s
Simulator	171.3 s	47.6 s

Table 2: Comparison of the tail flow completion time of the simulator and the testbed with the same topology and workload. Type 1 is faster because senders and receivers share ToRs for this workload.

our Subways protocols. The Layer-3 switches use drop-tail queues and flow-level ECMP. The queues are per-port with size based on the bandwidth-delay product of the network. Switching latency along with network propagation delays total to 60 µs per hop. Our workload varies by experiment. For experiments that use TCP, we implemented TCP New Reno in the end hosts using the MPTCP codebase [35] as a reference.

We simulate a standard 3-layer FatTree topology in which ToR switches have 36 10 GbE ports and all other switches have 12 10 GbE ports. Each cluster consists of 12 racks and up to 6 aggregation switches. In some of the following experiments, we evaluate the sensitivity of Subways to different configurations by beginning with a default configuration and varying one parameter at a time.

Our default configuration has two ports per server and 15 servers per rack. For Type 0, we wire both ports to the same ToR switch; for Type 1 and Type 2, we wire servers to overlapping ToRs with a loop size of 12—the size of an entire cluster. The ToR layer has a 5:1 oversubscription ratio while the aggregation layer has a ratio of 4:1. When varying the oversubscription ratio at a given layer, we do so by removing links and aggregation/core switches. For instance, we create our 5:1 ToR-layer oversubscription ratio by only including 6 aggregation switches per cluster and adjusting the number of core switches accordingly.

We assume that ToRs have per-port packet counters that are aggregated by local controllers. Every T_{rebind} = 10 ms, the controllers collect all the packet counters and disseminate them to any subscribed end hosts. For detouring, we use T_{detect} = 10 ms, $T_{duration}$ = 100 ms, and limit the number of hops to at most two intermediate servers.

8.2 Validation Using Our Testbed

We validate our simulator using a small Cloudlab [30] testbed. The testbed emulates 16 dual-ported servers, 4 ToRs, and 2 aggregation switches connected through a 2-layer FatTree topology with an oversubscription ratio of 4:1. Because Cloudlab's network topology and queuing characteristics differ from our system, we emulated the ToR and aggregation switches using servers. For this purpose, we have added support for flow-based ECMP routing to the Linux kernel version 3.13. ECMP was implemented using the same consistent hash function that memcache uses to map keys to servers [19]. To prevent the emulated switches from becoming overloaded, we limited link rates to 500 Mbps. For comparison purposes, we replicated this setup in our simulator.

For validation, we emulated a scenario where three racks of servers all send traffic to the servers in a single rack. Since a rack has 4 servers, there are a total of 12 senders and 4 receivers. Each sender sends 4 simultaneous 100 MB flows to each receiver. In the testbed, this was accomplished using `iperf`. In the simulator, we started 192 simultaneous TCP flows. In both cases, we record tail flow completion time.

Table 2 shows the time from the first flow start to the last flow completion for simulation and emulation, for Subways Type 0 and Type 1. With Type 0, all flows traverse the aggregation layer and are bottlenecked at the downlink into the ToR switch for the receivers. With Type 1, most senders have shortcuts through a shared ToR to the receiver rack. The flows from the remaining senders traverse the aggregation layer, but with less contention. For both topologies, our simulator and testbed results matched each other closely. Some of the remaining difference can be attributed to kernel scheduling latency and jitter resulting from our need to emulate switches in software.

8.3 Speeding up a MapReduce Shuffle

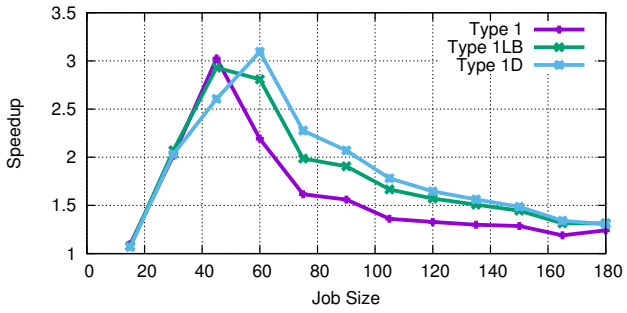
Our first evaluation considers the effect of Subways on the performance of an all-to-all MapReduce-like shuffle.

Experiment. Using the default configuration described above, we tested a range of shuffle job sizes. We measured the speedup in completion time of different Subways types compared to Type 0 with the same hardware. For each job, all of the servers in several contiguous racks act as both mappers and reducers. We minimize the number of racks used and group them together as much as possible in order to promote locality. Mappers initiate flows to reducers with a shuffle size derived from [11], which, in this experiment, is 15 MB between each pair. We mark the job as completed when the last TCP flow finishes and compare the completion time with that of Type 0. Note that we ignore the effects of cross-traffic.

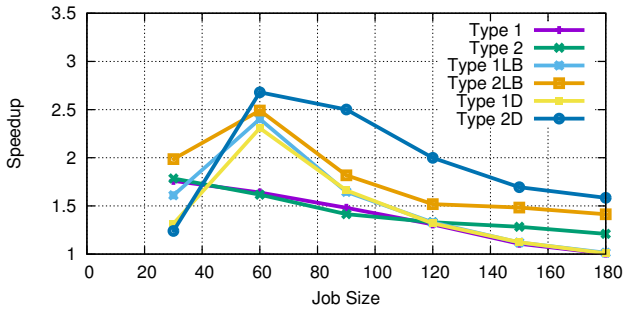
We evaluate two situations. The first involves job sizes of up to 180 nodes (12 racks) within a single cluster. For these, we only test Type 1 and its load balancing variants as Type 2 provides very little additional benefit for a purely intra-cluster and intra-loop workload. The second situation also tests several different job sizes, but splits those servers equally between two different Subways loops. For this case, we test both Type 1 and Type 2 variants.

Results. Fig. 6 shows results for both intracluster and cross cluster configurations. In most cases, Subways significantly outperforms a Type 0 architecture using the same amount of hardware. This is most pronounced for mid-sized jobs (up to a 3.1× speedup for 60 nodes in an intracluster job) with the speedup tapering off for larger jobs. The primary reasons for these performance benefits are (i) more short paths that avoid the network backbone, (ii) the ability to spread uplink/downlink traffic to the backbone across more ToRs, and (iii) for Type 2, increased cross cluster bandwidth.

In the intracluster case, these benefits peak at a job size of 45/60, i.e., 3 or 4 racks of servers. In these cases, most of the traffic can be transmitted directly through shared ToRs; this is in contrast to Type 0 and traditional rack-based architec-



(a) Intracluster



(b) Cross cluster

Figure 6: The effect of Subways on an all-to-all MapReduce shuffle workload and a range of job sizes. For both intracluster and cross cluster configurations, we show the speedup of different Subways wiring and load balancing variants compared to Type 0.

tures where short paths only exist within a single rack. Load balancing has a greater relative effect—with $p = 2$, 3 racks can spread their uplink/downlink traffic across 4 ToRs versus 3 ToRs with Type 0. Adaptive load balancing and detours only serve to enhance and extend this benefit. As the job size grows to encompass an entire loop, the load balancing effect disappears as there are no free racks for either load balancing or detours. This represents a worst case for our load balancing algorithms. However, even so, short paths continue to provide a modest benefit (about a 1.3× speedup).

Like the intracluster configuration, our cross cluster experiments show significant benefits for small to medium job sizes, peaking at a speedup of 2.7× for Type 2D compared to Type 0. For large, cross-cluster jobs, the bottleneck becomes the core network For this reason, Type 2 with its greater amount of cross-cluster bandwidth is particularly effective for this workload. Adaptive load balancing and detours provide further benefits for Type 2 for the same reason.

8.4 Comparison of Upgrade Paths

Next, we examine the effect of the above performance improvements on different upgrade paths.

Experiment. We reconsider the MapReduce shuffle pattern of the previous section, but focus on a job size of 90 nodes all contained within a single cluster.

Our experiment measures speedup in job completion time versus a baseline where every server has a single 10 GbE link and the backbone is also made of 10 GbE links. From this baseline, we evaluate four potential upgrade paths: a full network replacement with 25 GbE links, one with 40 GbE links, a Subways-style 10 GbE augmentation, and a Subways-style 25 GbE augmentation. Note that for both the single port and Subways configurations, we maintain a 15 server rack and add/remove aggregation and core switches to keep the oversubscription ratio the same across upgrade paths. Because this is a heterogeneous configuration, we use Type 2 wiring.

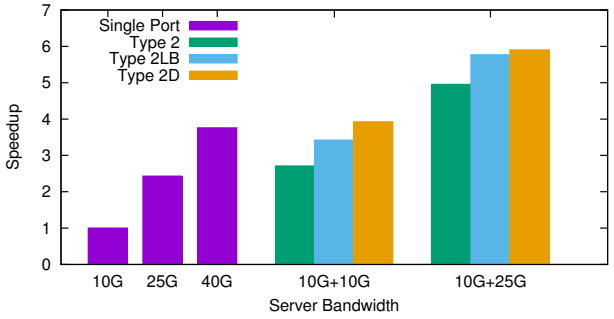


Figure 7: Speedup of a MapReduce shuffle for different upgrade paths. The baseline is a configuration with a single 10 GbE port on every server.

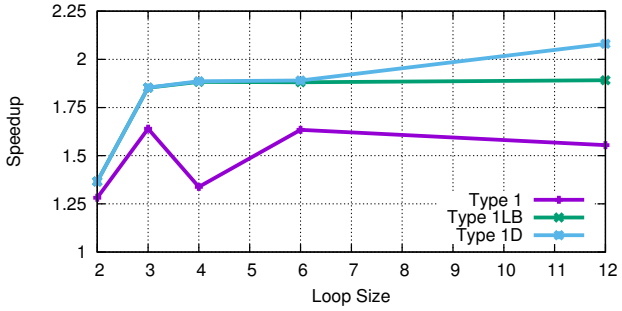


Figure 8: Speedup of a MapReduce shuffle for different loop sizes compared to Type 0.

Results. Fig. 7 compares the upgrade paths. As expected, performance of the 25 GbE and 40 GbE network replacements provide ∼ 2.5× and ∼ 4× speedup respectively. In contrast, with a Type 2D design, a 10 GbE augmentation provides about a 4× speedup. Despite the fact that the servers only have 20 GbE of total bandwidth, the performance is on par with a 40 GbE network because of decreased inter-ToR traffic and better load balancing. Augmenting with a 25 GbE link shows additional performance benefits.

8.5 The Effect of Loop Size

In this experiment, we evaluate the effect of loop size on performance. In particular, we look to answer two questions: (1) Is Subways better than a multihoming solution ($l = p$)? and (2) How sensitive is Subways to loop size?

Experiment. Again we consider a shuffle pattern with 90 contiguous nodes. We evaluate Subways Type 1, with and without load balancing, on loop sizes ranging from 2, essentially a multihomed configuration, to 12, spanning an entire cluster. The remainder of the topology adheres to our default configuration.

Results. Our results are shown in Fig. 8. From the graph, we

can see a few interesting effects. First, all Subways Type 1 variants benefit significantly from $l > p$. Multihoming provides a 1.3× speedup on this workload because it has short paths between more pairs of servers. However, a loop size of 3 provides a 1.64× speedup for Type 1 and a 1.85× speedup for Type 1LB and Type 1D, because of even more short paths and opportunities to spread backbone traffic across more ToRs.

Second, ECMP is sensitive to interactions between the job- and loop-size. When $l = 3$, the 6 racks of MapReduce servers fit perfectly in 2 loops and their ToRs are therefore naturally balanced when all flows are operating at full capacity. At $l = 4$, we must split the job into two groups of racks that do not fit perfectly in the loop; with ECMP, this leads to load imbalance at some ToRs as described at the beginning of Sec. 5. Adaptive load balancing fixes this issue. Finally, as expected, detours improve with loop size once the loop is large enough to encompass both the busy and free ToRs.

8.6 The Effect of Port Count

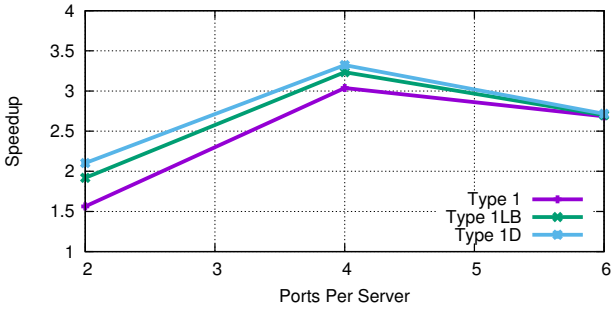
Our default configuration has two ports per server because it is the simplest Subways upgrade step for many data centers. In this section, we extend our discussion to study the effect of further increases in the port count.

Experiment. In this experiment we again use a shuffle workload. We begin with the default configuration and vary the number of ports per server. More ports increases the bandwidth per server. To keep the oversubscription ratio constant, each configuration includes a different number of aggregation switches. For $p = 2$, there are 6 aggregation switches per rack, for $p = 4$, 12, and for $p = 6$, 18. Within a given port count and job size, all Subways types operate under the same constraints.

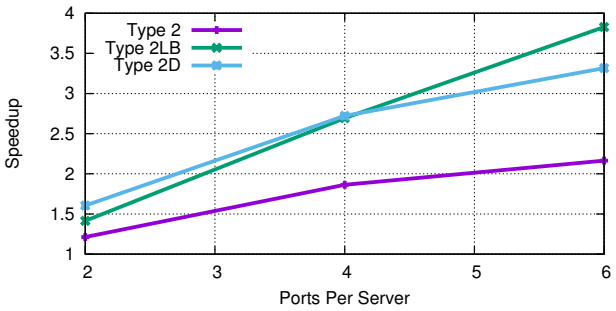
Like Sec. 8.3, we test two job configurations. The first involves 90 nodes across 6 racks within a single cluster to evaluate Type 1 and its variants. The second involves 180 nodes across 12 racks split equally between two different Subways loops. For this case, we only test Type 2 variants.

Results. Fig. 9a depicts the results for the intracluster configuration. By creating more short paths to more nodes, increasing the number of ports per server also increases the speedup of Subways. There are diminishing returns for very high port counts. With a job size of 6 racks, 6 ports is overkill—servers will prefer the short paths while rarely using their ToR uplinks. For the same reason, the relative benefits due to adaptive load balancing and detours decrease with 6 ports since most of the traffic bypasses the data center backbone.

The cross cluster results in Fig. 9b provide a complementary view of the effect of port counts on Subways performance. Because the job is divided amongst two loops, short paths will never dominate as they do in the intracluster case. Instead, the bottleneck is in the cross cluster network, where more ports leads to a greater degree of interconnection and speedup compared to Type 0. Here, load balancing becomes even more effective with more ports.



(a) Intracluster



(b) Cross cluster

Figure 9: Speedup of a MapReduce shuffle for different per-server port counts compared to Type 0.

8.7 Faster Memcache with Less Hardware

We also look at the effect of Subways on the throughput of a Facebook-like memcache deployment. In particular, we test a range of oversubscription ratios to evaluate the degree to which we can achieve the same performance as a Type 0, but with less hardware.

Experiment. We model this experiment on Facebook's memcache architecture [25]. Each rack within a single cluster consists of either memcache or web servers, but not both. Out of a cluster with 12 racks, 2 are memcache racks while the rest hold web servers. The web servers perform lookups for random keys spread across the memcache servers. Requests are done with UDP packets of 50 bytes, while responses are 1500 bytes. Because the nodes reside in a single cluster, we only evaluate Type 1 and its variants.

As in [25], the metric we use is the maximum sustainable memcache request rate. More specifically, all web servers in a cluster send requests at a constant rate to all memcache servers in the same cluster. We then record the average latency for responses after the system enters a steady state. To find the maximum sustainable throughput, we increase each web server's request rate until the average latency over all requests goes above 1 ms.

Results. Fig. 10 shows our results. We draw two conclusions from the graph. The first is that, for a given oversubscription ratio, Type 1 and each successive load balancing variant provides higher memcache throughput while maintaining a fixed latency. This benefit remains fairly stable across oversubscription ratios, with Type 1 hovering at around a 1.2× speedup compared to Type 0 and Type 1LB at around 1.6×.

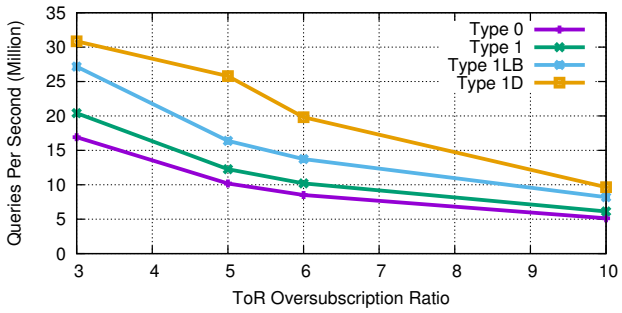


Figure 10: Throughput of memcached for various Subways variants for a range of oversubscription ratios. For a given ratio, each variant including Type 0 has the same amount of hardware. Throughput is the maximum number of queries per second while preserving an average response latency of 1 ms.

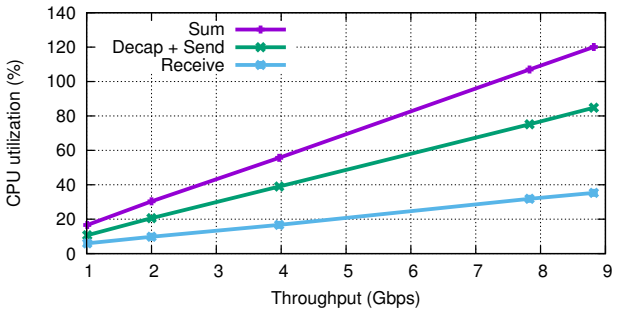


Figure 11: Detour throughput vs. server CPU utilization. Two cores are involved in detouring. We show the utilization of each individually, and the sum.

The relative benefit of Type 1D fluctuates between 1.8 and 2.5×, rising as the ToR becomes the bottleneck and falling due to our limit on detour path length. We note that, despite the fact that this experiment does not have any intra-rack locality, it does exhibit intra-*cluster* locality that allows Subways to decrease traffic in the data center backbone. Because responses are much larger than requests, the bottleneck is the outgoing capacity of each memcache rack. Load balancing and detours improve performance by spreading the load more evenly across the available ToRs.

The second conclusion is that, for a target latency and number of queries per second (qps), we can achieve equivalent performance with less hardware. For instance, to achieve a target qps of 10 million and a 1 ms average latency, Type 0 requires a ToR oversubscription ratio of at most 5. On the other hand, Type 1D can provide the same performance with a ToR oversubscription ratio of 10—a factor of two less backbone capacity.

8.8 Detour Forwarding Overhead

How much load does software detouring place on servers? We benchmark a prototype of our detour protocol and measure CPU utilization. Our prototype is implemented on the Arrakis high-performance server OS [26]. We note that any solution providing low-latency access to the network would have been sufficient (e.g., a Linux kernel module).

We conduct our experiment on a six machine cluster consisting of 6-core Intel Xeon E5-2430 systems at 2.2 GHz. Each system has an Intel X520 dual-port 10 Gb Ethernet adapter. Both ports of each machine are connected to a single Dell PowerConnect 8024F 10 Gb Ethernet switch. One machine is the detour server under scrutiny. The other machines generate detour traffic to one port of the server. The server decapsulates each detour packet and forwards it along its other port to the next hop.

Our prototype uses a simple pipeline of two server cores: the first core receives packets from the input NIC port, checks whether they are detour packets and, if so, puts a pointer to each one in a shared memory queue. For each queue entry, the second core decapsulates the corresponding packet and sends it to the other NIC port. When done, it uses another shared memory queue to inform the first core that the packet buffer can be reused. We do not copy the packet payload.

Fig. 11 shows the average CPU utilization for each pipeline step over a period of 5 s. We vary the detour load between 1 and 10 Gbps, the line rate of a single NIC port. We see that total utilization grows linearly with the detour throughput, with decapsulation contributing most to the load. Although our prototype requires two cores, in principle, a single core would be able to sustain a detour workload of up to 7 Gbps.

9. RELATED WORK

There has been a large body of work on the topology and management of the data center backbone, above the ToR switch level [5, 14, 23, 24]. Our work is complementary to these, in that wiring servers to overlapping sets of ToR switches can be combined with any of them.

BCube, Hypercube, and Torus [15, 21] are backbone topologies based on an assumption of multi-port servers, e.g., by connecting each server port along a different dimension of a hypercube. Although performance in Subways is better with higher values of p, we demonstrate performance benefits for $p = 2$, a port count far smaller than in this earlier work. Further, since we preserve the ToR switch layer, our work is compatible with existing data center operational constraints.

Port trunking (e.g., LACP (802.1ax), MC-LAG, and Trill [34]) is a well-known technique for connecting multiple server links to the same physical (LAG) or virtual (MC-LAG) switch. These architectures are equivalent to Type 0, or in the case of MC-LAG, Type 1 with $p = l$. We show that Subways provides significantly better performance than port trunking, along with further benefits when combined with adaptive load balancing and detour routing.

GRIN [3] is an interesting proposal that looks at wiring adjacent servers together directly, in addition to the ToR switch. Traffic through the adjacent server is detour routed through that server's ToR link. If the servers are in the same logical rack, this has the effect of increasing the burst bandwidth into each server, at low cost. If the servers are in adjacent racks, GRIN becomes a way to wire each server indirectly into multiple ToR switches, at the cost of dedicating CPU cycles and server link bandwidth to forwarding traf-

fic. Our work is complementary to GRIN, in that our wiring topologies, adaptive load balancing, and detour load shedding could be applied in a GRIN context.

Another approach to reducing hotspots by adding capacity to ToR switches is to use specialized hardware like optical circuit switches and Wi-Fi [12, 16, 22, 37]. Our work can be combined with these approaches. For example, if optical switching is used in combination with a traditional multi-level tree, then Subways can ameliorate any overload that occurs in the oversubscribed switched network. Our work also shows that it is possible to reduce the impact of hotspots with existing hardware.

10. CONCLUSION

Growing pains are often most acute at the edge of the network. In this paper, we describe Subways, a novel way to wire servers to ToR switches that enables incremental upgrades and reuse of existing hardware. Connecting servers to the ToR switches of neighboring racks decreases the traffic reaching the network backbone. When combined with advanced load balancing techniques, it also spreads the remaining backbone traffic across more ToRs. In addition, Subways improves locality and increases fault tolerance, all while remaining compatible with modern data center network practices, including multi-level switch backbones, wire bundling, rack-based servers, and inexpensive server cables.

Acknowledgments

We would like to thank our shepherd Marco Canini and the anonymous reviewers for their enormously helpful feedback. This research was partially supported by a Google Graduate Fellowship, the Hacherl Endowed Fellowship, and by the National Science Foundation (CNS-0963754).

11. REFERENCES

[1] http://www.cs.washington.edu/tr/2014/09/UW-CSE-14-09-01.pdf.

[2] (2012), TIA standard ANSI/TIA-942-A, data center cabling standard amended. Telecommunications Industry Association, 2012, http://www.tiaonline.org.

[3] A. Agache, R. Deaconescu, and C. Raiciu. Increasing datacenter network utilisation with GRIN. In *NSDI*, 2015.

[4] M. Al-Fares et al. Hedera: dynamic flow scheduling for data center networks. In *NSDI*, 2010.

[5] M. Al-Fares, A. Loukissas, and A. Vahdat. A scalable, commodity data center network architecture. In *SIGCOMM*, 2008.

[6] M. Alizadeh et al. Data center TCP (DCTCP). In *SIGCOMM*, 2010.

[7] A. Andreyev. Introducing data center fabric, the next-generation Facebook data center network. https://code.facebook.com, Nov. 2014.

[8] A. Belay et al. IX: A protected dataplane operating system for high throughput and low latency. In *OSDI*, 2014.

[9] T. Benson, A. Akella, and D. A. Maltz. Network traffic characteristics of data centers in the wild. In *IMC*, 2010.

[10] T. Benson, A. Anand, A. Akella, and M. Zhang. MicroTE: Fine grained traffic engineering for data centers. In *CoNEXT*, 2011.

[11] Y. Chen, S. Alspaugh, and R. Katz. Interactive analytical processing in big data systems: A cross-industry study of mapreduce workloads. *VLDB*, 2012.

[12] N. Farrington et al. Helios: A hybrid electrical/optical switch architecture for modular data centers. In *SIGCOMM*, 2010.

[13] P. Gill, N. Jain, and N. Nagappan. Understanding network failures in data centers: Measurement, analysis, and implications. In *SIGCOMM*, 2011.

[14] A. Greenberg et al. VL2: a scalable and flexible data center network. In *SIGCOMM*, 2009.

[15] C. Guo et al. BCube: a high performance, server-centric network architecture for modular data centers. In *SIGCOMM*, 2009.

[16] D. Halperin, S. Kandula, J. Padhye, P. Bahl, and D. Wetherall. Augmenting data center networks with multi-gigabit wireless links. In *SIGCOMM*, 2011.

[17] J. Hamilton. AWS innovation at scale. Presented at re:Invent 2014, Las Vegas, NV, 2014.

[18] Intel Corporation. *Intel 82599 10 GbE Controller Datasheet*, December 2010. Revision 2.6.

[19] R. Jones. libketama: Consistent hashing library for memcached clients. http://www.metabrew.com, 2007.

[20] S. Kandula, D. Katabi, S. Sinha, and A. Berger. Dynamic load balancing without packet reordering. *SIGCOMM CCR*, 37(2):51–62, Mar. 2007.

[21] F. T. Leighton. *Introduction to Parallel Algorithms and Architectures*. Morgan Kaufmann Publishers, Inc., 1992.

[22] H. Liu et al. Circuit switching under the radar with REACToR. In *NSDI*, 2014.

[23] V. Liu, D. Halperin, A. Krishnamurthy, and T. Anderson. F10: A fault-tolerant engineered network, 2013.

[24] N. Mysore et al. PortLand: a scalable fault-tolerant Layer 2 data center network fabric. In *SIGCOMM*, 2009.

[25] R. Nishtala et al. Scaling Memcache at Facebook. In *NSDI*, 2013.

[26] S. Peter et al. Arrakis: The operating system is the control plane. In *OSDI*, 2014.

[27] B. Pfaff et al. The design and implementation of Open vSwitch. 2015.

[28] D. R. K. Ports, J. Li, V. Liu, N. K. Sharma, and A. Krishnamurthy. Designing distributed systems using approximate synchrony in data center networks. In *NSDI*, 2015.

[29] C. Reiss, A. Tumanov, G. R. Ranger, R. H. Katz, and M. A. Kozuch. Towards understanding heterogeneous clouds at scale. ISTC-CC-TR-12-101, October 2012.

[30] R. Ricci, E. Eide, and The CloudLab Team. Introducing CloudLab: Scientific infrastructure for advancing cloud architectures and applications. *USENIX ;login:*, 39(6), Dec. 2014.

[31] A. Roy, H. Zeng, J. Bagga, G. Porter, and A. C. Snoeren. Inside the social network's (datacenter) network. In *SIGCOMM*, 2015.

[32] A. Singh et al. Jupiter Rising: A decade of Clos topologies and centralized control in Google's datacenter network. In *SIGCOMM*, 2015.

[33] D. Thaler and C. Hopps. Multipath issues in unicast and multicast next-hop selection. RFC 2991 (Informational), 2000.

[34] J. Touch and R. Perlman. Transparent Interconnection of Lots of Links (TRILL): Problem and Applicability Statement. RFC 5556 (Informational), May 2009.

[35] D. Wischik, C. Raiciu, A. Greenhalgh, and M. Handley. Design, implementation and evaluation of congestion control for multipath TCP. In *NSDI*, 2011.

[36] J. Zhou et al. WCMP: Weighted cost multipathing for improved fairness in data centers. In *EuroSys*, 2014.

[37] X. Zhou et al. Mirror mirror on the ceiling: Flexible wireless links for data centers. In *SIGCOMM*, 2012.