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

Vincent Liu Danyang Zhuo Simon Peter Arvind Krishnamurthy Thomas Anderson
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.

In this paper, 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 improves performance compared to either a single faster server link or the equivalent number of links wired to a single ToR. For example, we show that Subways offers 3× better memcache throughput while keeping request latency constant, and 1.9× better MapReduce performance, compared to an equivalent capacity network.

1. INTRODUCTION

Data center network operators face a number of challenges in keeping pace with increasing network demand. Faster and more capable servers, increasing application network-to-compute ratios [18], and kernel bypass techniques [30, 7, 20] mean that servers generate traffic at much higher rates than before. While designers can add network capacity to compensate, the cost of the network is already a large portion of the total cost of the data center [18]. Exacerbating costs, many data center applications are highly sensitive to changes in tail latency, requiring networks to be configured with low average link utilization.

For the backbone of a modern data center network, increasing capacity means increasing the number and capacity of the optical links and switches that interconnect racks. At each rack, there is a similar choice: to increase the number of links between each server and the Top of Rack (ToR) switch or to purchase higher capacity links. Either way, the switching capacity of the ToR must also be upgraded.

In this paper, we focus on the *edge* of the network, i.e., the ToR switches, their connections down to servers, and their uplinks to the aggregation switches in the network backbone. Growing pains are often most acute here—traffic is more bursty and congestion is most prevalent on the links into and out of the ToR switches. Costs at that layer, including connectivity to servers and aggregation switches, often predominate.

At the edge, one option is to wire servers in parallel to the ToR switch (trunking) instead of installing a new, faster link with equivalent aggregate speed. Trunking can be attractive because servers often have multiple ports. ToR switch capacity must be upgraded in either case; switch cost is typically driven by aggregate switching bandwidth rather than the number of downlink connectors. A second option is to use the extra server port to wire each server to the corresponding server in the neighboring rack [3], but this consumes extra processing in the common case that bandwidth or load balancing is needed.

In this paper, we study a third option, called Subways: wiring each server to ToRs in adjacent racks. Compared to trunking or an equivalent capacity faster link, the primary benefits are improved fault tolerance, a simpler upgrade path, decreased load on the core network, and improved load balancing into and out of the ToR switch layer.

There are several ways of connecting servers redundantly to adjacent ToRs. We characterize the space of Subways-style topologies and show how addressing, routing, and load balancing can be accomplished in these networks. These options, called Subways Levels, offer a range of tradeoffs in complexity and performance. In Section 6, we develop a set of wiring options to show that physical wire lengths can be kept short enough to be implemented with copper and also made relatively easy to install.

We evaluate Subways using three different artifacts: (1) a packet-level simulator, (2) a small testbed implementation, and (3) an implementation of server forwarding on top of the Arrakis high-performance server OS [30]. Compared to using an equivalent-bandwidth network with either faster single links or a trunking-based solution, we show that for our workloads Subways dramatically reduces packet loss, often by more than an order of magnitude on a fixed traffic matrix of TCP flows. We also show substantial performance benefits for application workloads due to reducing network tail latency: 1.9× speedup in MapReduce and a 3× throughput improvement in memcache for a fixed average request latency, relative to an equivalent-bandwidth network.

2. BACKGROUND AND MOTIVATION

High-performance data centers, like those of Facebook, Google, and Microsoft [22, 29, 11] house, cool, and operate 10s to 100s of thousands of servers. The servers and the switches that connect them are stacked in physical racks to simplify installation and management. Each server connects to the rest of the data center through one or more ToR switches, which serve as leaves in the multi-rooted tree that interconnects the entire data center.

Despite the inherent scalability of multi-rooted trees, to reduce the number of expensive optical links, the tree is usually thinned immediately above the ToR level. That is, the aggregate bandwidth from servers into a ToR is often 4 to 5× that of the ToR's optical uplinks. Higher levels of the tree interconnect

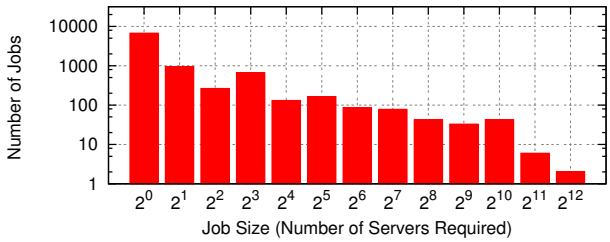


Figure 1: Distribution of job sizes in a Google data center [33]. Each bucket represents a bin from 2^k to $2^{k+1}-1$. 63% of servers are in jobs that require more than a standard 40-server rack.

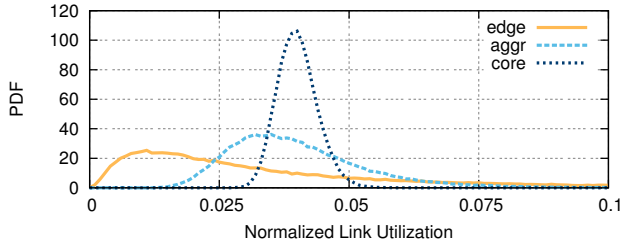


Figure 2: A PDF of normalized link utilization at different layers of the data center network derived from ToR traffic in [8]. Though all layers have the same total traffic, higher layers have successively shorter tail utilization because ECMP spreads load over multiple paths.

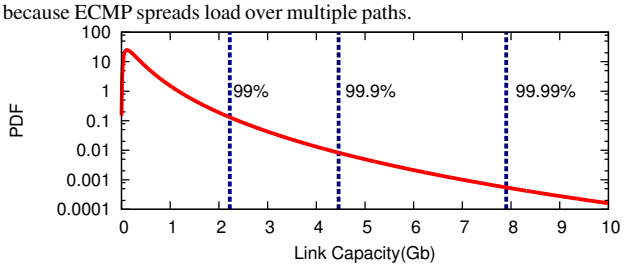


Figure 3: A PDF of the measured load on a single link of a ToR (derived from [8]) and its effect on required capacity. The vertical lines indicate the required capacity in order to handle 99%, 99.9%, and 99.99% of the offered load.

are even more oversubscribed. Due to this fact, the majority of devices and most of the network's cost lies near the edge.

At any point in time, there is a technology-dependent cost tradeoff between trunking and installing a single faster link. Subways adds an extra knob. We show that it is almost always better to wire across racks to multiple ToRs, rather than trunking within a rack or wiring directly between servers. Further, we show that Subways may be preferable to a single faster link even when there is a cost advantage of a single link compared to trunking. Not only are incremental upgrades easier, wiring to ToRs across racks reduces the uplink bandwidth required to the aggregation switches by a factor of 2x, for equivalent performance. There are several reasons for this.

2.1 Inter-rack Communication is Growing

One technique for decreasing network load is to keep communication local where possible. Intra-rack network bandwdith is generally cheaper and not as oversubscribed as communication between different racks. 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 [8].

Despite this, there are many cases when inter-rack communication is unavoidable. Sometimes, e.g., when fault tolerance or load balancing are more important, this distribution is by design. Other times, it is because the job cannot fit within a single rack. For example, consider Fig. 1, which plots the distribution of job sizes from a recent Google trace [33]. We can see from the wide range of values that *any* reasonable fixed rack size is not adequate. In fact, given a rack size of 40, 63% of servers are involved in jobs that cannot fit within a single rack. In both cases, rising job sizes and the increased prevalence of large analytics workloads imply that inter-rack communication will continue to grow at a very fast pace [18].

2.2 Edge Traffic is Bursty

Exacerbating the problems involved in handling communication growth is the fact that data center traffic is very bursty, particularly near the edge [17, 6, 8]. This is partly because many applications have traffic patterns that are inherently uneven; there are at least two additional reasons.

The first is that servers/jobs with similar purposes will often be placed in the same rack. Facebook, for example, might expand its Hadoop infrastructure by rolling out an entire rack of servers that are optimized for that particular service. Although this approach significantly reduces operational complexity, when combined with the need to preserve locality it means that the traffic patterns of servers within a rack are often highly correlated; when a shuffle occurs in a job that spans multiple racks, all of the servers in those racks get hot simultaneously.

The second reason is that, while there are many ways to load balance traffic over all the paths of the inter-rack network, no such options exist at the edge. There is a single bottleneck for the rack's traffic: the ToR switch. Fig. 2 illustrates the result of this using ToR traffic from [8]. Each layer has the same total traffic and average utilization, but the links near the edge have a traffic distribution with a much longer tail, i.e., those links are much more bursty.

Why is burstiness a problem? Fig. 3 shows traffic distribution at a ToR in a production data center [8]. (Although traffic has increased since this study, we expect it to be comparably long-tailed in today's data centers.) Because the traffic distribution is long-tailed, the ToR is underutilized most of the time, but also spends a significant amount of time at very high utilization. In that study, 99% of traffic requires less than 2 Gbps of bandwidth, but trying to handle 99.9% of the demand without congestion requires more than double the capacity (4.4 Gbps). Handling 99.99% requires another doubling (∼8 Gbps).

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 fault tolerant. 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 many modern architectures, 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 par-

	Single ToR per logical rack	Shared ToRs w/in a cluster	Shared ToRs in diff clusters
Uniform random	Level-0	Level-1	Level-2
Adaptive load balancing	N/A	Level-3	Level-4
Detours	N/A	Level-5	Level-6

Table 1: The Subways family of architectures. There are two axes in the design space: wiring along the columns and load balancing along the rows. In general, moving downward or to the right increases both performance and complexity.

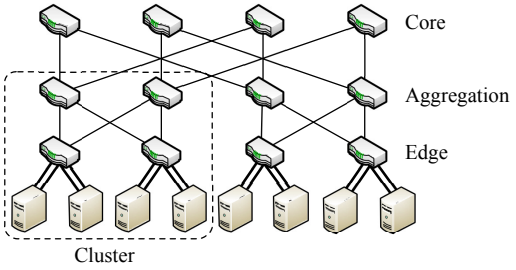


Figure 4: A FatTree with Level-0 Subways. Three layers of switches connect servers together. Each server has two ports that are both connected to the same ToR switch.

ticularly important as ToRs have relatively high failure rates compared to other network devices [13].

3. SUBWAYS

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. Clever usage of multiple server-ToR connections can mitigate many fundamental issues in today's data centers:

- *Capacity upgrades:* Adding capacity at the edge requires large amounts of up-front investment. By augmenting connectivity and allowing servers to connect to adjacent ToRs, we enable cheap, potentially incremental upgrades.
- *Inter-rack communication:* Keeping traffic within a rack decreases the load on the inter-rack network and often increases performance. An overlapping connection pattern creates short paths for more destinations.
- *ToR hot spots:* Traffic is very bursty, particularly at the ToRs. By connecting servers to multiple, differently-loaded ToRs rather than just one, we gain better load balance.
- *Fault tolerance:* Redundant server-ToR links remove one of the few remaining single-points-of-failure in data centers.

Table 1 is a visual representation of the various design points. There are two axes: wiring and load balancing. Each design provides unique tradeoffs, but in general, architectures are more complex and provide more performance as we move to the right or downward on the table. We begin with a baseline design, Level-0 Subways, before describing other variants.

Note that since Subways is an edge architecture, it is compatible with *any* data center topology. However, for simplicity we will focus on a canonical three-layer FatTree topology [4] (as in Fig. 4). ToRs are interconnected by a series of aggregation switches, which are in turn connected by core switches. ToRs and aggregation switches are grouped into *clusters* such that all connectivity within a cluster is provided by just the bottom two layers. Table 2 provides notation used in this paper.

Notation	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 (Levels-$\{2,4,6\}$)
l	# of racks in a single Subways loop (Level-1 and up)

Table 2: The variables that define a Subways architecture.

Starting Point: Level-0 Subways

We begin by describing a baseline solution, Level-0 Subways, which corresponds to the current, industry-standard approach for using a p-port server. In this approach, the p connections between each server and its ToR are trunked into a single logical connection using protocols like LACP, Cisco's EtherChannel, or Juniper's Aggregated Ethernet. The entire trunk must terminate at single, physical switch as shown in Fig. 4.

Because of the above requirement, an upgrade requires an operator to rewire many existing connections. More specifically, let us assume she wants to double capacity while keeping the number of servers (N_s) and oversubscription ratio constant. Any such upgrade requires her to install N_s new server-ToR connections and double the number of switches in the network. Level-0 also requires her to rewire $\frac{N_s}{2}$ of the existing server-ToR links and rewire/expand the existing interconnect to ensure that new ToRs have connectivity to the old ones. In addition, most trunking protocols are limited to links of the same speed. Still, this approach improves the maximum throughput of each server by virtue of having more links, but we do not gain any additional fault tolerance or load balancing benefits.

4. WIRING TYPES

Rather than isolating each rack to its own ToR switch, Subways instead wires servers to adjacent ToRs. We now introduce two wiring techniques that are based on this principle. To flesh out how an operator would design an entire data center around them, we describe them in the context of Level-1 and Level-2, which both use a simple, ECMP-like load balancing algorithm.

4.1 Level-1: Shared ToRs Within a Cluster

In Level-1 Subways, the p ports of each server connect to p different ToRs—its own and those of the p-1 closest neighboring racks. In this way, neighboring racks share at least one ToR. All of the server's ToRs reside in the same cluster so that routing in the rest of the network is not changed, and connections are randomly distributed across available ToRs.

This simple act of sharing has far-reaching benefits. For instance, if $p = 2$, two adjacent logical racks are connected to three ToRs, instead of just two. For those two racks, this

represents a 50% increase in combined peak throughput, a 50% increase in fault tolerance, and short paths to 50% more servers in addition to statistical multiplexing and other benefits. These can have a powerful effect on congestion, data center cost, and job placement.[1]

Topology. Conceptually, each 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 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 counter-clockwise from it. A loop is thus a connected component in the server-ToR topology. For simplicity, we assume a single loop length l for all loops. Loops are confined within a single cluster, and while they ideally span the entire cluster, this may not be the case due to physical wiring concerns (see Section 6).

Fig. 5 shows a single loop of 3-port servers and a length of 9. Fig. 6a shows two loops where the loops are the same size as the clusters. Note that if $p > l$, some link aggregation must occur in a manner similar to Level-0 Subways.

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. 5 shows an example of this, where an upgrade from 2 ports to 3 requires adding new ToRs, but leaves the existing ToR connections untouched. As server-ToR connections are the most numerous and length sensitive, this potentially saves a fair amount of work compared to Level-0. Note, however, that an operator may still need to rewire the upper layers of the data center and split loops to ensure that all of a server's ToRs are contained within the same cluster.

Routing. Routing is largely identical to routing in today's data centers and can be 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 [35]. 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.

In a multi-rooted tree, only nodes that are directly connected to ToRs (i.e., servers and aggregation switches) need to install 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.

[1]The exact job placement algorithm is orthogonal to our work, but we note here that it is similar to the existing problem of slab allocation.

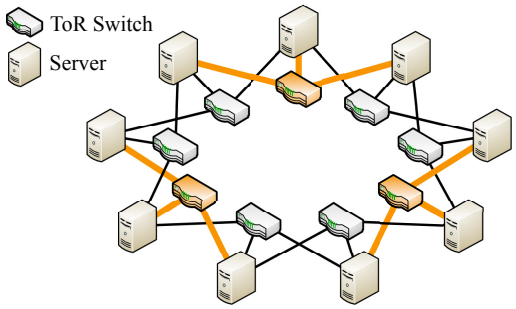


Figure 5: Diagram of the Subways-style server-ToR connection found in Level-1 and above. This example is of a single loop of 9 3-port servers. Bold orange links/switches are the hardware that must be added to upgrade from a 2-port to a 3-port configuration without rewiring.

Benefits and Tradeoffs. Allowing servers to connect to adjacent rack rather than just their own provides all of the benefits outlined in Section 3. Together, these result in up to 20× reduction in loss rates (cf. Fig. 11). It also provides:

- *Simple implementation:* Deploying this configuration of Subways involves very little change to routing protocols and can be done using existing protocols and a small number of extra routing table entries.

Level-1 Subways has the following tradeoffs:

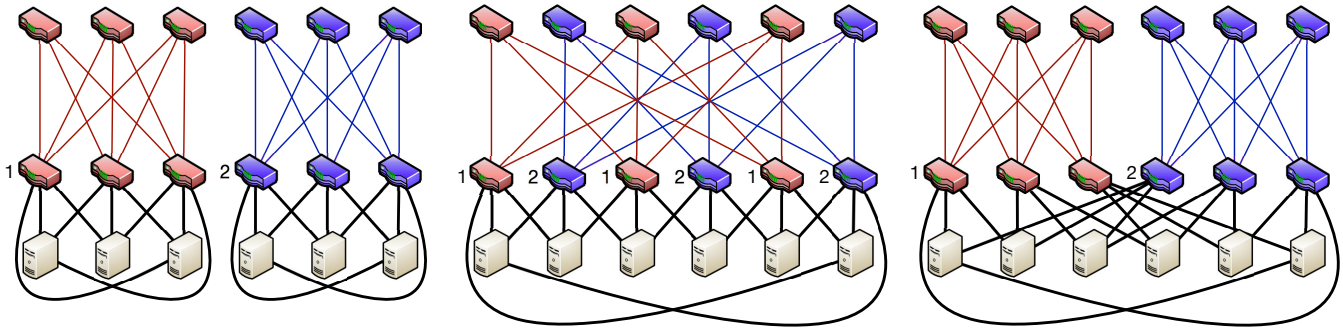
- *Increased cabling complexity/length:* Connecting to adjacent racks necessitates longer wires and more connections. However, in Section 6, we explain how to minimize length and complexity with proper bundling and server placement.
- *Upgrades are still limited:* Although Level-1 reduces rewiring and improves performance and fault tolerance, it ends up having some of the same restrictions of a Level-0 architecture. In particular, all links should be the same capacity, and some amount of rewiring is needed in an upgrade.

4.2 Level-2: Shared ToRs in Different Clusters

The server-ToR topology of Level-2 is no different than Level-1—Fig. 5 still applies. Their ToRs, however, are assigned to different clusters rather than a single cluster. This simple reassignment removes the restrictions of Level-0/Level-1 and enables true incremental augmentation. It also serves to expand the performance benefits of Level-1 to entire clusters. With this wiring methodology, we increase the number of intra-cluster shortcuts, which in turn reduces traffic in the core layer. Likewise, load can now be spread across entire clusters.

Topology. Every server in a Level-2 Subways is connected to multiple clusters as evenly as possible. The number of clusters that are connected to a single loop is configurable and depends on factors such as how much mixing is needed, how long loops should be, 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 the ToRs in Fig. 5 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



(a) Level-1 (b) Level-2 (c) Level-2 (alternate representation)

Figure 6: Logical representations of two clusters in both a Level-1 and Level-2 architecture with $p = 3$ and $l = 3$ and 9, respectively. The diagrams show how a Subways-style connection pattern like the one in Fig. 5 fits into the wider picture. We omit the core layers and color/number ToRs according to their cluster. Note that 6b and 6c are the exact same topology depicted two different ways. In both, the two clusters can have different capacities.

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. 6b and Fig. 6c show two equivalent examples where $c = 3$. Note that servers still only connect to nearby ToRs (Fig. 6b), and at the same time, the cluster topology is still amenable to cable bundling (Fig. 6c)—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 Level-2 does not actually require any rewiring. In fact, it is even possible to perform a capacity upgrade without any disruptions in service. Furthermore, the upgrades can be incremental. For 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. This leaves the existing network untouched.

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 then 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 WCMP [39]. For example, two servers with parallel 10 and 25 GbE connections would place flows on those networks with probability 2/7 and 5/7, respectively.

Benefits and Tradeoffs. Level-2 provides many benefits in addition to those of Level-1:

- *Load balancing between clusters:* Random placement of flows across clusters in addition to ToRs improves load balance at all layers of the data center.
- *Decreased load on the core layer:* Having connections to multiple clusters increases the number of intra-cluster shortcuts compared to a Level-1 architecture. Intuitively, this is because each server is connected to $N_c = \max(p,c)$ clusters (rather than just 1), and therefore shares a cluster with $N_c\times$ as many servers. Using these shorter paths significantly decreases the load on the core layer.
- *Longer loops:* This variant allows for much longer loops than a Level-1 architecture, which is limited by the size of a cluster; however, like a Level-1 architecture, wiring/routing concerns may still limit loop length.

These benefits result in up to 22× total reduction in packet loss rates (cf. Fig. 11), but come with the following tradeoffs:

- *Need for multiple addresses per server:* Adding multiple addresses to servers and using WCMP may require small modifications to the server OS and address resolution protocols.
- *Increased ToR-Agg wiring complexity:* Though wiring is still amenable to bundling and other cable management techniques, there is a possibility that the physical implementation of this design requires longer wires. Whether it does and by how much is highly dependent on the physical topology of the data center.

5. LOAD BALANCING TYPES

Changing the server-ToR wiring pattern is enough to provide significant performance and cost benefits, but it also opens up the possibility of advanced load balancing.

The previous section introduced two wiring patterns and presented designs that used them in combination with random placement. In this section, we introduce two other load balancing mechanisms that are effective for both wirings.

5.1 Levels-{3,4}: Adaptive Load Balancing

In Subways Levels-{3,4}, servers adaptively choose which ToR to use based on network conditions rather than just sending randomly over all ToRs. In this section, we assume $l > p$.

As a motivating example, consider the case where $p = 2$ and two adjacent racks are simultaneously hot—a situation that is made more likely by job placement that takes advantage of locality in Subways. With ECMP, the shared ToR would have twice the load of the two non-shared ToRs.

To address this imbalance, we introduce an adaptive, local load balancing scheme that is related to previous work on Weighted-ECMP [39, 25, 5, 9]. 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.

Though these previous systems require centralized control, 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 entirely local to each server and utilization information can be easily sharded amongst multiple, independent controllers (e.g., a separate controller for every cluster).

Initial flow placement. In the case where there is at least one 2-hop intra-ToR shortcut, the source chooses one of them with equal probability. Otherwise, it must choose to route the flow through one of its ToRs and one of the destination's ToRs. We choose each independently and base the decision 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 Level-3 and Level-4, respectively.

Setting weights for source ToRs. Recall that a set of controllers store utilization statistics for every ToR. For the source ToR in particular, we care about outbound traffic. Utilization is stored as an exponentially-weighted moving average of the fraction of capacity that is remaining.

More formally, for each server, we have $[U_1,U_2,..,U_p]$, the aggregate uplink utilization of every ToR to which it is connected. The weight for a server's ith ToR is $w_i = \frac{a_i}{\sum_{j=1}^{p} a_j}$, where a_x is a function of U_x, w'_x, the xth ToR's weight from the previous round, and B_x which is the maximum capacity of the link. This function, $f(U_x,w'_x,B_x)$, should have two properties. First, it should be inversely proportional to U_x so that higher ToR utilization leads to less weight. Second, in steady state when U_x is the same across all the uplinks, w_x should be proportional to B_x, so that a 40 GbE link will be used more than a 10 GbE link. In the remainder of this paper, we use $f(U_x,w'_x,B_x) = \frac{\sqrt{w'_x B_x}}{U_x}$.

To give a concrete example, let us consider a server with two ports. Let us assume that the aggregate uplink utilization is 20% for the left ToR and 80% for the right ToR. Both of the server links are 10 GbE. Let us also assume that $w'_{left} = 30\%$ and $w'_{right} = 70\%$. The left path has $a_1 = 8.66$ and the right path has $a_2 = 3.31$. The sender will therefore put new flows on the left path with probability $w_1 = \frac{8.66}{11.97} = 72.3\%$ and on the right path with probability $w_2 = 27.7\%$.

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 the per-ToR aggregate downlink utilization of the Aggregation-ToR links $[D_1,D_2,..,D_p]$.

The resulting weights for a server's i^{th} ToR is $v_i = \frac{b_i}{\sum_{j=1}^{p} b_j}$, where $b_x = f(D_x,v'_x,B_x)$ and v'_x is the weight on the x^{th} ToR in the previous round. In Level-4, these weights are provided to each server for each communication partner. In Level-3, they are provided to each aggregation switch for their own cluster.

Subsequent load balancing. We periodically reschedule existing, long-lasting flows to adapt to changing traffic conditions. Rebinding takes place every T_{rebind} in each end host and/or each aggregation switch. Source servers will rebind a flow exactly as it would for a new flow. For destination ToRs, either the source (Level-3) or the aggregation switch (Level-4) will rebind the destination ToR for each flow in the similar way.

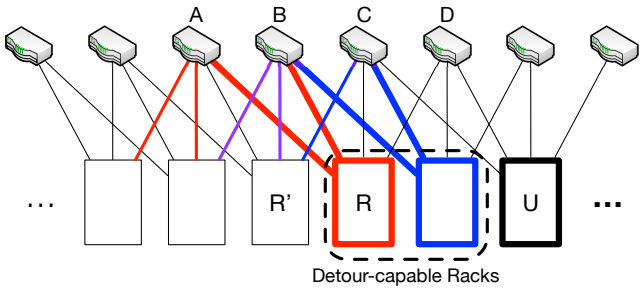


Figure 7: A rack-level diagram showing all one-hop, leftward detour paths for an example configuration. Boxes represent racks, with bold boxes indicating that the rack is hot. Bold links carry detour traffic, with colors indicating the original source.

Benefits and Tradeoffs. Adaptive load balancing provides:

- *Decreased tail latency/utilization:* Load balancing allows the network to handle more load at lower cost while also decreasing tail latency. This is especially effective when multiple adjacent racks are simultaneously loaded, an increasingly likely possibility with correlated task placement.

This results in up to 34× total reduction in packet loss rates (cf. Fig. 11) and implies the following tradeoff:

- *Server support and/or centralized controllers:* Intelligent traffic engineering requires features beyond what is included in today's commercially-available components. This may include a few extra machines.

5.2 Levels-{5,6}: Subways Detours

An interesting side effect of the Subways server-ToR connection pattern of Level-1 and up is that each loop is a connected graph even without the use of the inter-ToR network. By rerouting traffic along Subways links whenever there is congestion at a ToR, servers can avoid hotspots and spread load over more than just their own ToRs.

In the common situation where a single rack is hot, servers can detour all excess traffic to other ToRs in the loop. In an architecture where all server-ToR links have the same capacity, the result is full burst bandwidth to/from the loaded rack *regardless of network's oversubscription ratio*.

As an extreme example, consider the topology in Fig. 7. Let us assume that there are 20 servers in rack R and all of them wish to fully utilize their four 10 GbE links. Their directly-connected ToRs A, B, C, and D have only 10 Gbps of total uplink capacity—an 80:1 oversubscription ratio. Although R's neighbors can shift traffic away from those ToRs, the adaptive load balancing described in the previous section can satisfy only $\frac{1}{20}$th of R's offered load.

A server in Subways Levels-{5,6} can handle all 800 Gbps despite this oversubscription by bouncing traffic through adjacent racks. Here, some portion of the offered load would be

sent through A, B, C, and D, while the remainder is detoured away from the loaded racks, e.g., through R'. This happens recursively, with each successive ToR egressing a portion of the remaining traffic until all of it is handled.

This can be implemented using software switches that are increasingly common (e.g., Open vSwitch [31]) and with NICs that provide similar functionality. [21, 26]. While these 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.

Simplifying assumptions. Our algorithm assumes that:

- Congestion is a result of a single hot rack or contiguous group of hot racks.
- A "hot" rack is one that is very highly utilized. Formally it drives all of its connected ToRs to $> u\%$ utilization over some time period.[2]
- All other ToRs have negligible utilization.

By assuming that all hot racks are contiguous, that they are almost fully utilized, and that all other racks are very lightly utilized, we can ignore current utilization. In fact, we can precompute routes—detour paths are static for a given topology. Violations of these assumptions only result in less detoured traffic; they do not affect correctness or non-detoured flows.

Overview. For convenience, we use relative directions and positions. For example, we refer to the two edges of the group as the "left edge" and "right edge". Without loss of generality, we will focus on outbound traffic from the left edge.

The core idea behind our protocol is simple: servers and ToRs detour traffic to the left by taking packets from their right $\lfloor \frac{p}{2} \rfloor$ racks/ToRs and forwarding them to their left $\lfloor \frac{p}{2} \rfloor$ racks/ToRs. Excess traffic is thus iteratively detoured away from the loaded racks, utilizing $u\%$ of the uplink capacity of every ToR along the way. Source servers are responsible for precomputation/assignment of paths and are expected to detour a full $\lfloor \frac{p}{2} \rfloor$ links' worth of capacity. Fig. 7 shows a diagram of all one-hop, leftward detour paths for an example configuration that we use in the discussion below.

For simplicity, we focus on the general case where there is a large, contiguous group of hot racks in which the two edges of the group can be considered separately.

Which racks detour. Like most other detouring nodes, the original source detours to the left by sending traffic through their $\lfloor \frac{p}{2} \rfloor$ leftmost ToRs. Because of this, when a group of adjacent racks is simultaneously loaded, not all of them are able to detour. For instance, each of the ToRs connected to U either lack a path to an unloaded rack or, in the case of C, cannot handle more detour traffic given that it is only connected to a single unloaded rack, R'.

As a result of these restrictions, only the $\lfloor \frac{p}{2} \rfloor$ leftmost racks in a contiguous group can detour any traffic. However, each of these racks can fully utilize their $\lfloor \frac{p}{2} \rfloor$ leftmost ToRs for detours, thereby freeing their ToRs to handle traffic from the remaining loaded racks. The total amount of detourable traffic for each edge is $r\lfloor \frac{p}{2} \rfloor^2$.

When to Detour. Subways detours are triggered whenever there is persistent congestion at the ToR level. Persistent congestion is measured by a central controller that gathers ToR packet counts every T_{detect} ms.[3] Whenever a ToR is using $> u\%$ of its total ToR-aggregation 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's servers will be immediately notified of the high load. Likewise, when all of a destination rack's ToRs are hot in the incoming direction, a notification is sent to all servers that are sending to the target rack.

Notifications continue to arrive every T_{detect} ms until the congestion subsides. Included in these notifications is information on whether the $\lfloor \frac{p}{2} \rfloor + 1$ ToRs on either side of the source rack's ToRs are also congested. Note that detoured load is separated when calculating congestion in order to avoid feedback loops.

Precomputing detour paths. Our simplifying assumptions enable servers to precompute every possible detour path and the amount of traffic that should be placed on each. These values are stored as a lookup table.

When a server receives a congestion notification, it can immediately deduce the set of loaded racks from the set of loaded ToRs. From this information, it can determine whether it is detour-capable, and if so, the direction of the detour. New flows are placed onto the detour according to a weighted probability provided by the lookup table. Routing of the flows is accomplished through recursive encapsulation with headers that specify the next hop in reverse order. We refer interested readers to our anonymized tech report for a full description [1].

Benefits and Tradeoffs. Detouring provides the following:

- *Load balancing across entire loops:* Rather than only spreading load across local ToRs and clusters, servers can spread traffic among any ToR in its loop.
- *Greater burst bandwidth:* As a result of the above benefit, we can provide more capacity than is possible with just a rack's own ToRs at no extra hardware cost. If only a single rack in a loop is loaded, this increase is optimal—we can provide full burst bandwidth regardless of the amount of network oversubscription.

Detouring results in up to $50\times$ reduction in packet loss rates (cf. Fig. 11) and comes with the following tradeoffs:

- *Server support for forwarding:* To detour, servers must act as a simple forwarder. We show in Section 7.8 that the software overhead of forwarding is very small when using modern OSes and NICs. Further, some NICs are already beginning to include simple routing logic [19].
- *Potential for congestion at other racks:* Detouring adds load to other racks and can create congestion on detouring links; however, it is rare due to the bursty nature of data center traffic. Even when it occurs, detoured traffic can be placed

[2] We assume $u = 60$ in the remainder of the paper based on empirical testing using published measurement studies.

[3] One could also use existing mechanisms like ECN instead of a central controller, but such a design is out of the scope of this paper.

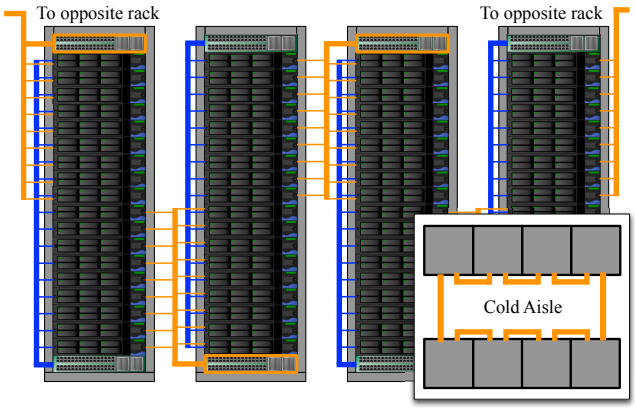


Figure 8: An example physical topology for Subways Level-1 an up. Blue links are part of the initial 1-port configuration. Orange links and outlined switches denote hardware that is added as part of the upgrade to 2 ports. The main image is the frontal view of a row of racks, while the inset is an aerial view of the entire Subways loop.

at a lower QoS level, does not effect the original traffic, and therefore has strictly higher throughput on all flows than in Level-4 or lower Subways.

6. PHYSICAL DESIGN 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. 8 shows the wiring pattern for our 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 "close"—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.[4]

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 wire 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.

[4]This example is optimized for expansion to 2 ports per server, but similar techniques can be used to handle higher port counts.

7. 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 Emulab [37] testbed to validate our simulator.
- A server detour implementation to test the feasibility of software-based detouring through servers.

7.1 Simulator Implementation

We built an event-driven, packet-level simulator consisting of 3244 lines of Java. It implements both low-level switch behavior and all of our Subways protocols. The Layer-3 switches use standard drop-tail queues and flow-level ECMP. Switch and interface processing time along with network propagation delays total to 10µs. Our workload varies by experiment. For experiments that use TCP and MPTCP, we implemented TCP New Reno and MPTCP in the end hosts using the MPTCP codebase [38] as a reference.

We simulate a standard 3-layer FatTree topology that, unless otherwise noted, has a 5:1 oversubscription ratio at the ToR layer and a 1:1 ratio at the Aggregation layer. ToR switches have 72 10 GbE ports, while the remaining switches have 24 ports. Each cluster consists of 12 racks. In the base case, there are 12 aggregation switches per cluster and 144 core switches. When varying the oversubscription ratio at a given layer, we do so by removing links and aggregation/core switches. For instance, a 2:1 aggregation-layer oversubscription ratio can be implemented by removing 6 uplinks from each aggregation switch, then consolidating the links into 72 core switches.

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 and $T_{duration}$ = 100 ms.

7.2 Validation Using Our Testbed

We validate our simulator using a small Emulab testbed [37]. The testbed includes 4 servers and 4 ToRs connected through a 2-layer FatTree-topology with an oversubscription ratio of 4:1. We implemented both Level-0 and 1 with flow-level uniform load balancing. We replicated the setup in our simulator.

For validation, we emulated a scenario where three nodes were all sending traffic to a single node. In the testbed, this was accomplished using `iperf`. In the simulator, we started three

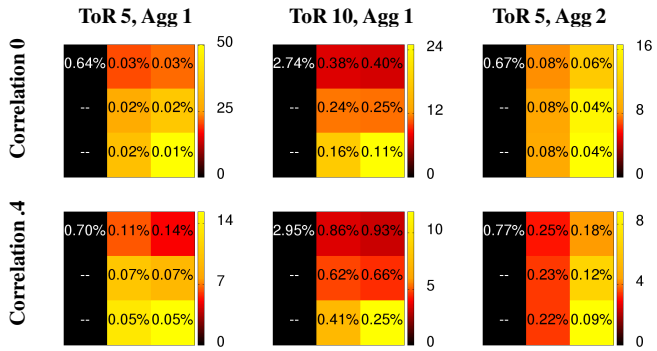


Figure 11: Comparison of Subways variants under different data center and workload conditions. Our heat metric is the ratio of Level-0 loss to Subways loss. Higher is better. We vary the ToR and Aggregation oversubscription ratios (shown at the top) and probability that adjacent racks have correlated traffic patterns. Each heatmap shows the different Subways levels in the same format as Table 1. Numbers show absolute loss percentages.

simultaneous TCP flows. Both our testbed and our simulator use TCP New Reno. In both cases, we waited until the TCP flows reached a steady-state and took the average throughput over 900 s. Table 3 shows the aggregate throughput for both topologies and setups. We found that in both cases, our simulator and testbed results matched each other closely.

	Level-0	Level-1
Testbed	94.15 Mbps	188.31 Mbps
Simulator	94.10 Mbps	183.25 Mbps

Table 3: Comparison of aggregate throughput of both the simulator and the testbed with the same topology and workload.

7.3 Mixed Workload

We first test the performance of Subways under a wide variety of conditions and a realistic, general workload.

Our mixed workload is derived from a measurement study of Microsoft data centers by Benson et al. [8, 9]. From their measurements, we derived distributions for flow size, flow length, and rack-level traffic. To account for differences in the number of servers and size of the network, we scaled the reported numbers to ensure that the distributions match the average load and ToR hotspot frequency. We also matched the drop rates reported in [28]. We then synthesized a randomized trace that matches the observed behavior. The synthetic trace assumes offered load is independent of congestion, so we model traffic as UDP flows and use packet loss rates as the performance metric. TCP will generally back off quickly, resulting in lower delivered throughput than shown here.

Experiment. Each server has four ports with 15 servers per rack. This results in a data center with 4320 servers. Levels-1, 3, and 5 have loops of length 12 while Levels 2, 4, and 6 have loops of length 24 that span two clusters.

There are three axes that we vary to stress test our design. The first is the correlation factor, which indicates the probability that a rack has the same load as its neighbor. This emulates the effect of job co-location. We also separately vary the oversubscription ratio at the ToR layer and the Aggregation layers.

Results. Fig. 11 shows the results for different types of Subways. Within a wiring type and configuration, higher levels of Subways provide successively better performance. This is true because small changes in capacity and load balancing can have a large impact on loss rate. We note three trends:

- *Detours (bottom row) become relatively more effective as we increase ToR oversubscription.* With more ToR-level capacity, simpler Subways designs are sufficient to ensure low loss for this workload; however, as oversubscription increases, detours are needed to mask the loss of bandwidth. Even for higher correlation factors, detours remain effective because they handle smaller groups of busy racks *very* well and can offload traffic from the edge of any larger groups.
- *With less available core bandwidth, Levels-$\{2,4,6\}$ significantly improve performance.* In our experiments with higher oversubscription at the aggregation layer, even Level-2 outperformed all intra-cluster designs. This is because underprovisioning the upper layers turns access to the core into the bottleneck. Levels-$\{2,4,6\}$ reduce the use of this bottleneck.
- *As the traffic correlation between adjacent racks increases, Levels-$\{3,4\}$ show their greatest relative advantage versus uniform random balancing.* With no traffic correlation, Subways performs well. In fact, it provides a greater relative benefit in these cases than in cases with high correlation. This is because it is easier to load balance completely independent traffic. However, we expect traffic will be colocated in adjacent racks because of the localization benefits that Subways offers. Despite this, Subways maintains a substantial advantage. Note that a correlation factor of .4 is fairly high, so load balancing is likely to perform well in practice.

7.4 Faster MapReduce with Less Hardware

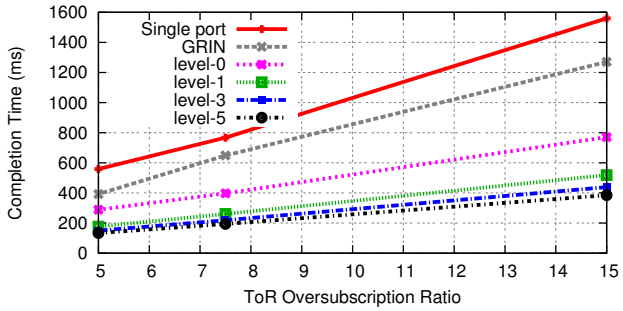
We next evaluate the effect of Subways on the performance of MapReduce. We run these tests for multiple oversubscription ratios to quantify the effect of different designs.

Experiment. We considered the performance of just the shuffle phase, where flows are initiated from mappers to reducers. The servers in several contiguous racks take on both roles. We evaluate two scenarios: a smaller case with $p = 2$ and larger case with $p = 4$. In both cases, each mapper sends the same amount of data to all reducers. Per-job shuffle size is derived from [10], and, in this experiment, is 1.5 MB between each pair. Note that we ignore the effects of cross-traffic.

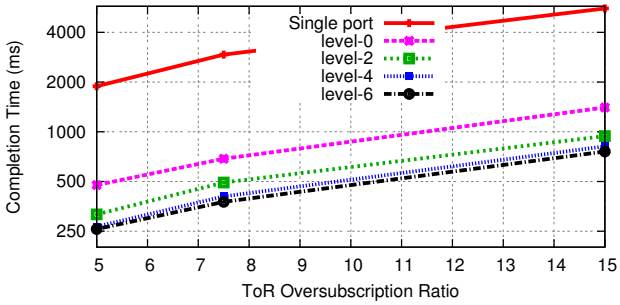
In the smaller configuration, we place 90 MapReduce servers on 6 contiguous racks. Because the smaller case is contained within a cluster, we only evaluate Levels-0, 1, 3, and 5. As before, these intra-cluster variants have $l = 12$. In the larger configuration, we place 270 servers on 18 racks across two clusters. Here we evaluate Levels-2, 4, and 6 with $l = 24$.

We also simulate GRIN[3] for the smaller case. In GRIN, each server has one port connecting to its ToR and one port connecting to a server on a neighboring rack. Packets are spread over the 4 path options (2 on the sender and 2 on the receiver side) using MPTCP [32].

Results. This experiment is an interesting case study because, in some ways, it represents a worst case for our load balancing algorithms. All MapReduce nodes are placed in adjacent



(a) 6 racks of 2-port servers.



(b) 18 racks of 4-port servers.

Figure 12: MapReduce job completion times for two differently-sized configurations.

racks. Thus, any burst traffic needs to be shunted toward the underused ToRs at the edge of the group. On the other hand, putting MapReduce tasks near each other showcases Subways' ability to use inter-neighbor shortcuts and therefore reduce the amount of traffic traversing the aggregation or core layers.

Fig. 12 shows the results for both the small and large configurations. In both cases, we can see that successive Subways variants improve MapReduce performance.[5]

Level-0 increases performance compared to the single port design by a factor of 2 in the smaller case and 4 in the larger case because of corresponding increases in the total network equipment. Between the Subways variants, the largest jump in performance occurs between Level-0 and the first Subways-style wiring pattern. One of the reasons is the increase in the number of usable ToRs (from 6 to 7 in the smaller case and 18 to 21 in the larger case). The other reason is that Subways reduces the amount of traffic that the ToRs need to send into the (oversubscribed) inter-ToR network. In the smaller case with $p=2$, most racks can shortcut an additional $\frac{2}{5}$ of flows; in the larger case with $p=4$, most racks can shortcut an additional $\frac{4}{17}$.

Adaptive load balancing and detours further decrease tail latency by more evenly spreading load across the ToRs. Both load balancing techniques do so by offloading as much traffic as possible to the ToRs at the edges of the loaded group. We note that part of the reason why the detour benefits are constant across oversubscription ratios is that the detours are limited by loop length—regardless of oversubscription ratio, detours only gain access to three additional ToRs in our larger configuration.

In sum, Subways provides up to a 2.1× speedup versus a Level-0 architecture in the smaller case. What is more, if we fix our desired completion time at, say, 300 ms, a Level-5 architecture would allow us to increase our oversubscription ratio from 5.22 to more than 11.84, a reduction of up to 56% comparing with Level-0 in the amount of required networking equipment in the ToR-interconnect. In the larger case, Level-6 can provide up to a 1.9× speedup versus Level-0. Similarly, if we fix completion time at 600 ms, Level-0 requires an oversubscription ratio of 6.39, while Level-6 requires 11.9—a reduction of 46%.

Finally, we note that GRIN, in the smaller case, reduces the tail latency by 18.5%. These gains come at a minor additional cost, but there is a limit on how much it can provide. Even at relatively low oversubscription ratios, Subways achieves a more than 2.8× speedup that increases to 3.4× for higher oversubscription ratios.

[5] Just like loss rate, MapReduce performance is primarily affected by worst case connection bandwidth; even small reductions in utilization can have a large impact on tail latency.

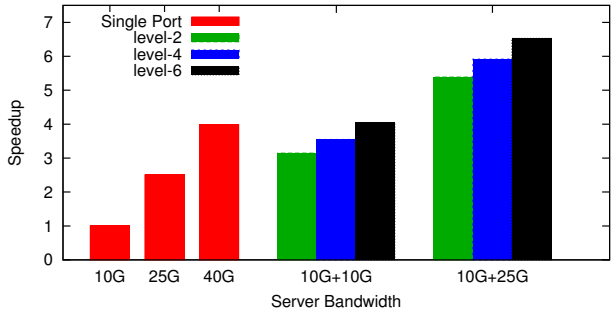


Figure 13: MapReduce job completion times for different upgrade paths.

7.5 Incremental Upgrade

In addition to the oversubscription ratios and configurations tested in the previous section, we also evaluate the performance of various upgrade paths.

Experiment. We reconsider the smaller MapReduce shuffle of the previous section. Our baseline here is a configuration where every server has a single 10 GbE link and the core 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. In each case, we add aggregation and core switches to keep the oversubscription ratio the same. For applicable configurations, we measure Levels-{2,4,6}.

Results. Fig. 13 compares these 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 Level-6 design, a 10 GbE augmentation already 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.

7.6 Improving Memcache Throughput

We also look at the effect of Subways on the throughput of a Facebook-like memcache deployment.

Experiment. We model this experiment on Facebook's memcache architecture [28]. We assume that each rack consists of either memcache or web servers, but not both. Further, memcache lookups only occur from web servers to memcache servers within a cluster. We also assume that requests are done with UDP packets of 50 bytes, while responses are 1500 bytes.

Each server has 2 ports, and where applicable, $l = 12$. The ratio of web servers to memcache servers in a cluster is 5:1. Because all communicating nodes reside in a single cluster, we do not evaluate Levels-$\{2,4,6\}$.

As in [28], gains are quantified by measuring 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. 14a shows our results. Level-0 doubles the network hardware resources and thus has 100% more capacity compared to the single port case. Level-1 and Level-3 both show a small improvement in throughput due to increases in capacity and decreases in congestion comparing with Level-0. We also note that Subways still decreases inter-ToR traffic even though racks do not communicate with themselves.

Detouring, however, allows servers to shift traffic to the other ToRs in the loop, giving an almost 3× increase in throughput over the Level-0 architecture. We can see that this result matches expectation by looking at the ToR-level capacity of both configurations. In Level-0, two ToRs can be used for memcache traffic whereas Level-5 can use 10 ToRs at 60% capacity. This accounts for the approximately 3× increase in capacity.

7.7 Improving Partition-aggregate Latency

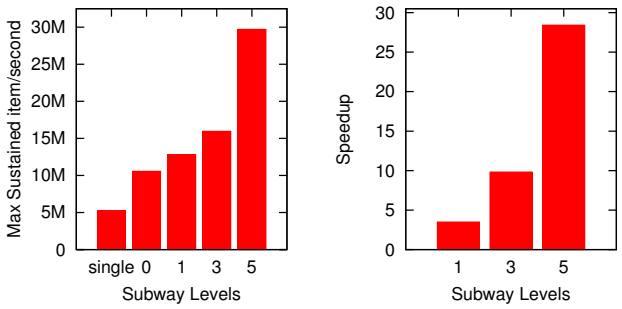
Using the same configuration, we also test Subways' effect on a memcache-based partition-aggregate workload.

Experiment. We use the same setup as the previous experiment. We choose a fixed, cluster-wide request rate of 16 M requests per second, and evaluate the speedup of a server aggregating content from 10 different sources. The server responds back to the client only when all sources reply.

Results. Fig. 14b depicts the average speedup relative to Level-0 for different intra-cluster variants of Subways. We note that the improvements to tail latency are much greater than to the maximum sustainable throughput. This is because server speed is gated on the slowest request. Because of this effect, Level-5 Subways provides a speedup of 28× over a Level-0 design with intermediate designs also providing significant gains.

7.8 Detour Forwarding Overhead

How much load does software detouring place on servers?



(a) Maximum sustained throughput. (b) Partition-aggregate

Figure 14: Memcache performance on Subways. (a) The maximum sustained throughput with average latencies of under 1 ms. (b) The speedup relative to Level-0 when aggregating memcache lookups from 10 different servers with high offered load.

We benchmark a prototype implementation of our detour protocol and measure server CPU utilization. Our prototype is implemented on the Arrakis high-performance server OS [30]. 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 10Gb Ethernet adapter. Both ports of each machine are connected to a single Dell PowerConnect 8024F 10Gb 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. 15 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.

Detouring could also be realized by software switches (e.g., Open vSwitch [31]) that are commonly deployed in data centers today. In the future, it is likely that this functionality migrates onto NICs due to the popularity of host-based switching.

8. RELATED WORK

Our work is complementary to the large body of existing work on data center topology [4, 27, 15, 14, 25, 34]. While they solely deal with the inter-ToR network, we are primarily concerned with the server-ToR links.

BCube, Hypercube, Torus, etc [16, 23] are distinct because they consider server links. In particular, they wire multi-port

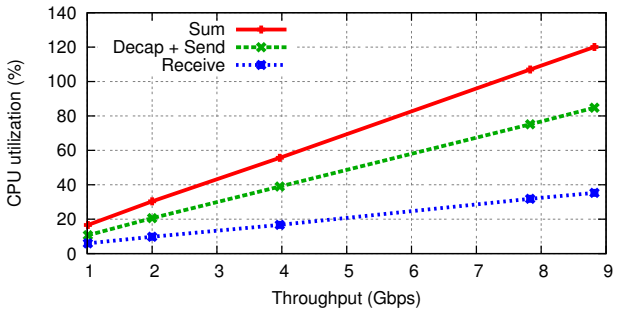


Figure 15: Detour throughput vs. server CPU utilization. Two cores are involved in detouring. We show the utilization of each individually, and the sum.

servers to build server-centric networks based on end host forwarding. Unlike Subways, however, these topologies are not designed for link capacity upgrades.

Port trunking (e.g., LACP (802.1ax), MC-LAG, and Trill [36]) is a well-known technique that also considers server links. These architectures are roughly Level-0. We show that our novel wiring and associated load balancing techniques provide benefits that are beyond what port trunking can provide.

Like Subways, GRIN [3] explores augmentation of existing data centers using additional server ports; however, their goal is to utilize existing, unused capacity rather than to add capacity. While their opportunistic approach is extremely low cost and useful in some settings, it assumes adjacent servers are rarely simultaneously loaded and discourages locality through job placement. We make the exact opposite design decisions at every turn and arrive at a different set of tradeoffs.

A separate approach to augmenting the data center network is to use specialized hardware like optical circuit switches and Wi-Fi [17, 12, 24, 40]. Our work can potentially achieve similar hotspot reduction with existing hardware. Even so, our work is complementary—less variance at the ToRs means less capacity required in the inter-ToR wireless/circuit-switched network.

9. 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. By connecting servers to the ToR switches of neighboring racks, we both decrease the traffic in the inter-ToR network and ensure that the remaining traffic is well balanced. This is in addition to improving the effects of locality, increasing fault tolerance and making upgrades easier. Our approach is compatible with any ToR interconnection network and maintains most of the benefits of existing infrastructure (e.g., cheap, short cables from the server; wire bundling; a backward-compatible, rack-based architecture; etc.).

10. 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 et al. Increasing datacenter network utilisation with GRIN. In *NSDI*, 2015.
[4] M. Al-Fares et al. A scalable, commodity data center network architecture. In *SIGCOMM*, 2008.
[5] M. Al-Fares et al. Hedera: dynamic flow scheduling for data center networks. In *NSDI*, 2010.
[6] M. Alizadeh et al. Data center TCP (DCTCP). In *SIGCOMM*, 2010.
[7] A. Belay et al. IX: A protected dataplane operating system for high throughput and low latency. In *OSDI*, 2014.
[8] T. Benson et al. Network traffic characteristics of data centers in the wild. In *IMC*, 2010.
[9] T. Benson et al. MicroTE: Fine grained traffic engineering for data centers. In *CoNEXT*, 2011.
[10] Y. Chen et al. Interactive analytical processing in big data systems: A cross-industry study of mapreduce workloads. *VLDB*, 2012.
[11] Cisco. Data center: Load balancing data center services. http://tinyurl.com/qcxhal8.
[12] N. Farrington et al. Helios: A hybrid electrical/optical switch architecture for modular data centers. In *SIGCOMM*, 2010.
[13] P. Gill et al. 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. DCell: a scalable and fault-tolerant network structure for data centers. In *SIGCOMM*, 2008.
[16] C. Guo et al. BCube: a high performance, server-centric network architecture for modular data centers. In *SIGCOMM*, 2009.
[17] D. Halperin et al. Augmenting data center networks with multi-gigabit wireless links. In *SIGCOMM*, 2011.
[18] J. Hamilton. AWS innovation at scale. Presented at re:Invent 2014, Las Vegas, NV, 2010.
[19] Intel Corporation. *Intel 82599 10 GbE Controller Datasheet*, December 2010. Revision 2.6. http://tinyurl.com/na5dhw2.
[20] Intel Corporation. *Intel Data Plane Development Kit (Intel DPDK) Programmer's Guide*, Aug. 2013. Reference Number: 326003-003.
[21] Intel Corporation. Flow APIs for hardware offloads, Nov. 2014. Open vSwitch Fall Confernce Talk. http://tinyurl.com/l2rr69q.
[22] S. Jain et al. B4: Experience with a globally-deployed software defined wan. In *SIGCOMM*, 2013.
[23] F. T. Leighton. *Introduction to Parallel Algorithms and Architectures*. Morgan Kaufmann Publishers, Inc., 1992.
[24] H. Liu et al. Circuit switching under the radar with REACToR. In *NSDI*, 2014.
[25] V. Liu et al. F10: A fault-tolerant engineered network, 2013.
[26] Netronome. NFP-6xxx flow processor. http://tinyurl.com/q5po3cv.
[27] R. Niranjan Mysore et al. PortLand: a scalable fault-tolerant Layer 2 data center network fabric. In *SIGCOMM*, 2009.
[28] R. Nishtala et al. Scaling memcache at facebook. In *NSDI*, 2013.
[29] Open Compute Project. *Server/SpecsAndDesigns*, January 2014. http://www.opencompute.org/wiki/Server/SpecsAndDesigns.
[30] S. Peter et al. Arrakis: The operating system is the control plane. In *Proceedings of the 11th USENIX Symposium on Operating Systems Design and Implementation*, Broomfield, CO, Oct. 2014.
[31] B. Pfaff et al. The design and implementation of Open vSwitch. In *NSDI*, 2015.
[32] C. Raiciu et al. Improving datacenter performance and robustness with multipath TCP. In *SIGCOMM*, 2011.
[33] C. Reiss et al. Towards understanding heterogeneous clouds at scale. ISTC-CC-TR-12-101, October 2012.
[34] A. Singla et al. Jellyfish: networking data centers randomly. In *NSDI*, 2012.
[35] D. Thaler and C. Hopps. Multipath issues in unicast and multicast next-hop selection. RFC 2991 (Informational), 2000.
[36] J. Touch and R. Perlman. Transparent Interconnection of Lots of Links (TRILL): Problem and Applicability Statement. RFC 5556 (Informational), May 2009.
[37] B. White et al. An integrated experimental environment for distributed systems and networks. In *OSDI*, 2002.
[38] D. Wischik et al. Design, implementation and evaluation of congestion control for multipath TCP. In *NSDI*, 2011.
[39] J. Zhou et al. WCMP: Weighted cost multipathing for improved fairness in data centers. In *EuroSys*, 2014.
[40] X. Zhou et al. Mirror mirror on the ceiling: Flexible wireless links for data centers. In *SIGCOMM*, 2012.