DRILL: Micro Load Balancing for Low-latency Data Center Networks

Soudeh Ghorbani[*]
University of Wisconsin-Madison

Zibin Yang
University of Illinois at Urbana-Champaign

P. Brighten Godfrey
University of Illinois at Urbana-Champaign

Yashar Ganjali
University of Toronto

Amin Firoozshahian
Intel

ABSTRACT

The trend towards simple datacenter network fabric strips most network functionality, including load balancing, out of the network core and pushes it to the edge. This slows reaction to microbursts, the main culprit of packet loss in datacenters. We investigate the opposite direction: could slightly smarter fabric significantly improve load balancing? This paper presents DRILL, a datacenter fabric for Clos networks which performs *micro load balancing* to distribute load as evenly as possible on microsecond timescales. DRILL employs per-packet decisions at each switch based on local queue occupancies and randomized algorithms to distribute load. Our design addresses the resulting key challenges of packet reordering and topological asymmetry. In simulations with a detailed switch hardware model and realistic workloads, DRILL outperforms recent edge-based load balancers, particularly under heavy load. Under 80% load, for example, it achieves 1.3-1.4× lower mean flow completion time than recent proposals, primarily due to shorter upstream queues. To test hardware feasibility, we implement DRILL in Verilog and estimate its area overhead to be less than 1%. Finally, we analyze DRILL's stability and throughput-efficiency.

CCS CONCEPTS

• **Networks** Network algorithms, Network architectures;

KEYWORDS

Microbursts, Load balancing, Traffic engineering, Clos, Datacenters

ACM Reference format:
Soudeh Ghorbani, Zibin Yang, P. Brighten Godfrey, Yashar Ganjali, and Amin Firoozshahian. 2017. DRILL: Micro Load Balancing for Low-latency Data Center Networks. In *Proceedings of SIGCOMM '17, Los Angeles, CA, USA, August 21-25, 2017,* 14 pages.
https://doi.org/http://dx.doi.org/10.1145/3098822.3098839

1 INTRODUCTION

Modern datacenter topologies such as Clos networks (Figure 1) are overwhelmingly constructed with large path diversity [4, 17, 19, 41, 42, 50, 51, 57]. A critical issue is the efficient balancing of load among available paths. While ECMP is extensively used in practice [42, 48, 64], it is known to be far from optimal [19, 27, 42]. A study of 10 datacenters, for instance, indicates a significant fraction of core links regularly experience congestion even while there is spare capacity elsewhere [23].

Many recent proposals address this need. Aligned with the recent trend of moving functionality out of the network fabric [28], many proposals strive to delegate load balancing to centralized controllers [18, 25, 32, 59], to the network edge [19], or even to end-hosts [27, 42]. These entities serve as convenient locations for collecting global or end-to-end information about congestion. A notable example is CONGA [19], in which switches at the edge (leafs in Clos networks) gather and analyze congestion feedback from remote leafs and spines to inform forwarding decisions. Planck [61], MicroTE [25], Mahout [32] and Hedera [18] collect global load information. All these approaches are based on the thesis that *non-local congestion information is necessary to evenly balance load.*

We explore a different direction: What can we achieve with decisions local to each switch? We call this approach *micro load balancing* because it makes "microscopic" decisions within each switch, sans global information; and because this in turn allows decisions on microsecond (packet-by-packet) timescales.

Micro load balancing offers hope of an advantage because load balancing systems based on global traffic information have control loops that are significantly slower than the duration of the majority of congestion incidents in datacenters, which are short-lived [23, 45]. The bulk of microbursts responsible for most packet loss, for instance, last for a few microseconds in measurements of [24]. Systems that collect and react based on global congestion information typically have orders of magnitude slower control loops [19, 61]. For example, even though CONGA adds hardware mechanisms to leaf and spine switches, its control loop still typically requires a few RTTs, by which time the congestion event is likely already over.

Our realization of micro load balancing is DRILL (Distributed Randomized In-network Localized Load-balancing). Like ECMP, DRILL presumes that a set of candidate (least-cost) next-hops for each destination have been installed in each switch's forwarding table by a routing protocol, and in the data plane acts entirely locally without any coordination among switches or any controllers. Unlike ECMP, DRILL makes forwarding decisions that are load-aware

[*] Work done while the first author was at University of Illinois at Urbana-Champaign.

SIGCOMM '17, August 21-25, 2017, Los Angeles, CA, USA

ACM ISBN ACM 978-1-4503-4653-5/17/08...$15.00
https://doi.org/http://dx.doi.org/10.1145/3098822.3098839

and independent for each packet. Even within a single switch, implementing this idea can be nontrivial. To accommodate switches with multiple forwarding engines, DRILL uses a scheduling algorithm inspired by the "power of two choices" paradigm [53]: each engine compares queue lengths of two random ports plus the previously least-loaded port, and sends the packet to the least loaded of these. We extend the classic theory to accommodate a distributed set of sources (forwarding engines), showing that the key stability result holds (§3.2.4). We show how to optimize DRILL's parameters to avoid damaging synchronization effects where many engines choose the same ports. We further investigate if DRILL's load-based scheduling algorithms within a switch could result in instability and hence low throughput [52], proving stability and a 100% throughout guarantee for all admissible independent arrival processes (§3.2.4).

This design raises a key challenge: how can we deal with packet reordering that results from load balancing at sub-flow granularities? Interestingly, we find that in Clos datacenter networks, even with failures, DRILL balances load so well that packets nearly always arrive in order despite traversing different paths. Regardless, occasional reorderings could still be undesirable for certain applications. Hence, similar to prior work [35, 42], we optionally deploy a buffer in the host GRO layer to restore correct ordering.

A second challenge is to handle asymmetric topologies: splitting a flow across asymmetric paths could cause serious problems for TCP due to reordering and differential loss rates. To handle asymmetry, DRILL partitions network paths into symmetric groups and applies micro load balancing inside each partition. We show that this results in bandwidth efficiency for admissible traffic (§3.4) and short flow completion times even under multiple failures (§4).

To evaluate DRILL, we simulate a detailed switch hardware model and a variety of topologies and workloads to compare DRILL with ECMP, CONGA, and Presto. Compared to ECMP, DRILL's fine granularity and load awareness make it substantially more effective in preserving low latency particularly under load, *e.g.,* under 80% load, it cuts the mean flow completion times (FCT) of ECMP by 1.5×. Compared to CONGA's use of "macroscopic" information, DRILL's micro load balancing enables it to instantly react to load variations as the queues start building up. DRILL results in dramatically shorter tail latencies, especially in incast scenarios (2.1× reduction in the $99.99^{\mathtt{th}}$ percentile of FCT compared to CONGA) and under heavy load (1.4× shorter $99.99^{\mathbf{th}}$ percentile of FCT compared to CONGA under 80% load). Plus, DRILL offers a simpler switch implementation than CONGA since it does not need to detect flowlets or send and analyze feedback.

Presto [42] offers an interesting comparison. It moves load balancing to the edge, but is congestion-oblivious: hosts source-route their traffic across all shortest paths, at the granularity of flow-cells. Thus it has finer granularity than ECMP, but DRILL has even finer granularity and is load-sensitive. We find that these factors yield higher performance for DRILL, but the degree depends on the nature of workload dynamics. The difference is greatest in bursty workloads, *e.g.,* , 2.6× improvement in tail FCT in an incast scenario (§4).

Finally, we implement DRILL in Verilog to test its hardware feasibility (§4). Our analysis of required logical units and their layout shows at most 1% overhead compared to a reference switch.

In summary, our results show that micro load balancing in Clos datacenter fabric is practical and achieves high performance, even outperforming schemes that use network-wide information.

2 BACKGROUND

Clos topologies enable datacenter providers to build large scale networks out of smaller, and significantly cheaper, commodity switches with fewer ports connected with links of less capacity than in traditional designs [41, 49]. Today, most datacenter and enterprise topologies are either built as one two-stage folded Clos, also called leaf-spine topologies (one example is shown in Figure 1) [19] or incorporate Clos subgraphs in various layers in their design. Clos networks or variants thereof are used in [49] and in various generations of datacenter at Google, for instance [64]. As another example, the VL2 network [41] is composed of a Clos network between its Aggregation and Intermediate switches.

A key characteristic of Clos networks is having multiple paths between any source and destination hosts. The common practice in datacenters today for balancing load among these paths is ECMP [48]. When more than one "best path", commonly selected to minimize the number of hops of each path, is available for forwarding a packet towards its destination, each switch selects one via hashing the 5-tuple packet header: source and destination IPs, protocol number, and source and destination port numbers. This path selection mechanism enables ECMP to avoid reordering packets within a TCP flow without per-flow state. All the examples of the Clos networks given above deploy ECMP [41, 49, 64].

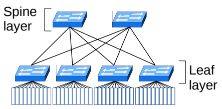

Figure 1: A leaf-spine: an example of a folded Clos.

ECMP, however, is routinely reported to perform poorly and cause congestion when flow hash collisions occur [19, 24, 42, 64]. Datacenter measurement studies, for instance, show that a significant fraction of core links regularly experience congestion despite the fact that there is enough spare capacity elsewhere to carry the load [23]. Many proposals have tried to enhance ECMP's performance by balancing finer-grained units of traffic. Aligned with the recent trend of moving functionality out of the network fabric [28], these proposals strive to delegate this task to centralized controllers [18, 25, 32, 59, 61], to the network edge [19], or even to end-hosts [27, 42]. In Presto, for instance, end-hosts split flows into *flowcells*, TSO (TCP Segment Offload) segments of size *64KB*; the network balances flowcells, instead of flows, in a load-oblivious manner [42]. Presto is built on the premise that the per-flow coarse granularity of ECMP combined with the existence of large flows in datacenters is the primary deficiency of ECMP, and in any Clos network with small flows, ECMP is close to optimal [42]. In CONGA, as another example, each edge switch balances flowlets [19] based on cross-network load information. Its central thesis is that not only the fine granularity of the load to balance, but also that global load information is essential for optimal load balancing and reacting to congestion. Presto and CONGA balance granularities coarser than packets to reduce reordering.

While improving ECMP, these proposals cannot effectively suppress short-lived congestion events that tend to persist for only sub-millisecond intervals [1, 13], sometimes called *microbursts*

[3, 13, 14, 23], as even the fastest ones have control loops with 10s of millisecond to a few second delays [19, 25, 61]. However, microbursts are responsible for a majority of packet loss in datacenters according to the measurements of [23]. In today's datacenters, despite the reportedly low average link utilizations (1% to 20-30% at various stages [62, 64]), the highly bursty nature of traffic [19, 25, 62] makes very short-lived periods of buffer overrun and consequently high loss rates the norm rather than the exception. The buffer utilization statistics at a 10-microsecond granularity from Facebook datacenters for switches connecting web servers and cache nodes, for instance, demonstrate a persistent several orders of magnitude difference between the maximum and the mean queue occupancies [62]. Plus, the maximum buffer occupancy in these Facebook web server racks is reported to approach the configured limit for approximately three quarters of the 24-hour measurement period, even though the mean link utilization for the same rack is only 1% [62]. These ephemeral high buffer occupancies are correlated with high drop rates [62]. The inherent traffic burstiness also results in high congestion drop rates in Google datacenters as utilization approaches 25%, so the utilization is typically kept below that level [64]. Given the pervasiveness of microbursts and their adverse impact on the performance, in terms of low flow completion times and high throughput, *our first goal is to provide high performance especially when microbursts emerge*.

Despite ECMP's suboptimality in handling congestion, its extreme simplicity and scalability effectively has turned it into the de facto load balancing practice in datacenters [41, 62, 64]. Notably, the fact that it is *local* to each switch in the sense that, for forwarding packets, each switch autonomously selects among available paths, irrespective of the load and choices of other switches, makes it easily deployed in conjunction with most routing protocols. Once the global topological information is gathered, each switch makes local forwarding decisions. Networks are therefore relieved of the burden of complex mechanisms for gathering global load information either via distributed algorithms (as in CONGA [19]) or in a centralized manner (as in Planck [61]). Ideally, we would want to share ECMP's scalability and simplicity. Hence, in designing DRILL, *our second goal is to make load balancing decisions local to each switch.*

3 DESIGN AND ALGORITHMS

3.1 Design overview

An ideal fluid-model approach for symmetric Clos networks. To provide intuition that we can build upon, we begin with a simplified model where traffic is a fluid. With the goal of modeling a short moment in time, we assume hosts inject fluid at fixed rates, and the network's goal is to balance the amount of fluid directed through parallel paths.[1] Furthermore, for now assume a symmetric leaf-spine data center.

In this model, consider a scheme we call **Equal Split Fluid (ESF)**: at each switch with n least-cost paths to a destination, the switch sends exactly $1/n$ of the fluid traffic to that destination along each path. ESF produces *precisely optimal load balance regardless of the traffic demands*. To see why, consider the first leaf on a path: all its traffic can be sent with equal cost to any of the n spines, so each "upward" port receives exactly $1/n$ of each flow entering that leaf. Hence, the incoming traffic to any spine (from all leafs) includes $1/n$ of every end-to-end flow. Since the spines' incoming traffic is identical, their outgoing "downward" traffic to any specific leaf is also identical. In the end, of course, not all ports have the same load; hosts and leafs will vary in how much flow they send and receive. But if we consider any two hosts, all shortest paths between them will have the same flow volume (and mix of individual flows!) at corresponding hops along the paths; we refer to these as **symmetric paths**.[2] This intuition extends to more general Clos networks (see Theorem 4 in [37]) and is essentially the fluid-model intuition behind Valiant load balancing (VLB) [68].

While optimal, ESF is merely a theoretical fluid-model ideal that the switching fabric needs to approximate in a real discrete world. We can interpret several existing load balancing schemes as attempting to approximate ESF. ECMP is similar to ESF in that across long timescales, on average, it will equally split traffic on all equal-cost outgoing ports. But it is a poor approximation at the small timescales that matter, because it makes decisions (a) in very coarse-grained chunks of whole flows, and (b) uniform-pseudorandomly without regard to load, resulting in occasional unlucky load collisions. Presto [42] shrinks the unit of discretization to the 64 KB flowcell, and uniformly spreads these flowcells using end-host source routing, partially mitigating problem (a). One could imagine going a step further to what we call *Per-packet Random* which sends each packet through a uniform-random intermediate (spine) switch. This design was avoided in [42] to reduce end-host CPU overhead and packet reordering. But regardless, Per-packet Random would help problem (a) but not (b), and we will see empirically that both problems are important.

DRILL as a near-optimal approximation of ESF. The previous discussion helps us frame the problem in a way that provides a direction forward: Can we approximate ESF even more closely? If we could, the ESF approach could achieve our goals of high performance even at microsecond timescales, and using a switch-local algorithm. But approximating the theoretical ideal is nontrivial. To achieve this, DRILL first chooses the smallest practical unit of discretization, *i.e.,* single packets. This is also a decision unit that is simple for switches to deal with statelessly (and avoids the concern mentioned in [42] of overhead of per-packet forwarding decisions at endhosts, by having switches make the decisions). Second, DRILL does not forward traffic uniform-randomly. Instead, DRILL leverages switch-local load information, sampling some or all outgoing queues when making each packet's forwarding decision and placing the packet in the shortest of these queues. Intuitively, this minimizes the "error" between the ideal fluid-based ESF and actual packet forwarding. In particular, we prove in §3.2 that DRILL is stable and can deliver 100% throughput for all admissible independent arrival processes.

[1] This model intentionally does not deal with packets or end-host control loops. If the fluid directed to some port exceeds its capacity, one can assume the excess is simply discarded in equal proportion on all flows; however, this is not relevant to the intuition we want to provide, and one can simply assume that the volume of fluid is within the capacity of each port.

[2] More precisely, let L_i be the set of hosts attached to leaf ℓ_i and $f(s, t)$ be the flow volume from host s to host t. All paths from host $h_1 \in L_1$ to host $h_2 \in L_2$ take the form $h_1 \rightarrow \ell_1 \rightarrow$ some spine $\rightarrow \ell_2 \rightarrow h_2$. Regardless of which spine is used, they have the same flow $\sum_{t \neq h_1} f(h_1, t)$ on the first link, the same flow $\frac{1}{n} \sum_{s \in L_1} \sum_{t \notin L_1} f(s, t)$ on the second link, the same flow $\frac{1}{n} \sum_{s \notin L_2} \sum_{t \in L_2} f(s, t)$ on the third link and the same flow $\sum_{s \neq h_2} f(s, h_2)$ on the last one.

Together, these mechanisms achieve a significantly better approximation of ESF than past designs. However, two key challenges remain, which we discuss next.

Packet reordering. DRILL's fine grained per-packet load balancing based on potentially rapidly changing local load information raises concern about reordering that could imperil TCP throughput. We show in §3.2 and §3.3 that under this algorithm, the load is so well balanced that even under heavy load, the probability of reordering is very small—in most cases, well below the degree that damages TCP throughput, and indeed well below the degree that can be resolved by recent proposals for handling reordering at the end hosts by modifying the Generic Receive Offload (GRO) handler [35, 42]. While DRILL can employ such techniques to avoid reordering, in many environments, it provides a substantial benefit even without them.

Topological asymmetry. In asymmetric networks, the world is no longer as simple. Naively splitting traffic equally among all paths, as in ESF, does not optimally balance load. And if TCP flows are split across paths of unequal available capacity, they will back off due to losses on the slower path, leaving some capacity unused. (These problems would carry through to Presto and per-packet Random as well.) Furthermore, if we split flows packet-by-packet among a set of paths with different load, and hence different queueing delay, we can expect a high degree of packet reordering.

To deal with asymmetric Clos networks, DRILL's control algorithms decompose the set of paths at each switch into a minimal number of groups, such that the paths within each group are symmetric. In the data plane, it hashes flows to a group (like ECMP), and then performs per-packet load-aware micro load balancing across the paths within each group (as described above for the symmetric case). Thus, as the network becomes more asymmetric, DRILL becomes more similar to ECMP. Note that this approach is not intended to handle fully arbitrary networks or alternative topologies like random graphs [65]. The primary environment we target for DRILL is symmetric leaf-spine or Clos data centers, which are likely to become asymmetric through occasional failures. Our approach essentially offers a graceful degradation from DRILL to ECMP.

Summary. In the spectrum of strictly load-oblivious schemes (ECMP, Presto) to globally load-aware schemes (CONGA [19], Planck [61]), DRILL occupies middle ground: it retains most of the simplicity and scalability of the first class, but leverages a small amount of additional local load information and negligible amount of state independent of the number of flows to achieve better performance than the state of the art in either class (§4). In the rest of this section, we present DRILL's design in more detail, beginning with the symmetric case (§3.2), and then handling reordering (§3.3) and asymmetry (§3.4).

3.2 DRILL in a symmetric Clos

DRILL in a symmetric Clos begins by using a standard control plane — OSPF with the ECMP extension in our prototype — to construct a global topology map, select routes, and install an equal-cost group in the forwarding table. DRILL's new mechanisms are in the data plane, which we focus on here.

In a symmetric Clos, our mission is to get as close to ESF as possible. Before presenting the algorithms, we provide a high level overview of the switching hardware that might affect load balancing.

3.2.1 Switching hardware. Switches have forwarding engines that make forwarding decisions for packets. While many of the simple switches deployed in datacenters have one centralized engine [8], higher-performance switches invariably have multiple forwarding engines [6, 26, 31, 67]. Very high performance switches might have multiple engines on each interface card [31]. These engines make parallel and independent forwarding decisions. Cisco 6700 Series [5], Cisco 6800 Series [5], Cisco 7500 Series [7], Cisco Catalyst 6500 backbone switch series [5], and Juniper MX Series [9] are some examples of switches that support multiple forwarding engines. In Cisco switches, for example, multiple Distributed Forwarding Card (DFC) are installed for line cards. The forwarding logic is then replicated on each DFC-enabled line card, and each card makes forwarding decisions locally and independent of other cards. Some switches have constant access to queue depth, typically as a means for micro-burst monitoring [1, 10, 12–14]. This feature allows the network provider to monitor traffic on a per-port basis to detect unexpected data bursts within a very small time window of μsec [13]. Our discussions with [8] indicate that while this information is easily accessible for packet forwarding, it is not always precise: the queue length does not include the packets that are just entering the queue until after they are being fully enqueued. Our simulator models this behavior. It is possible that in some switches, queue information would be even more imprecise or delayed; we leave a study of such switches to future work.

3.2.2 `DRILL(d,m)` scheduling policies. Here we present DRILL's algorithm for scheduling packets in each switch in a symmetric Clos. We assume that a set of candidate next-hops for each destination has been installed in the forwarding tables of each engine of the switch, using well-known mechanisms such as the shortest paths used by ECMP. DRILL is essentially a switch-local scheduling algorithm inspired by the seminal work on power of two choices [22] that, whenever more than one next hop is available for the destination of a packet, decides which hop the packet should take.

The `DRILL(d,m)` algorithm operates as follows at each switch. Upon each packet arrival, the packet's forwarding engine randomly chooses d out of N possible output ports, finds the one with the current minimum queue occupancy between these d samples and m least loaded samples from previous time slots, and routes its packet to that port. Finally, the engine updates its m memory units with the identities of the least loaded output queues among the samples.

This algorithm has time complexity $O(d + m)$. Our experiments with Clos networks with various sizes, switches with diverse number of engines, and different load show that (a) having a few choices and few units of memory is critical to the efficiency of our algorithms, *e.g.,* `DRILL(2,1)` significantly outperforms Per-packet Random and RR, and (b) increasing d beyond 2 and m beyond 1 has less of an impact on DRILL's performance, and in some cases may degrade performance due to a *synchronization effect*. We explain each point in turn.

3.2.3 The pitfalls of choice and memory. To inform our choice of d and m, we evaluate `DRILL(d,m)`'s performance and

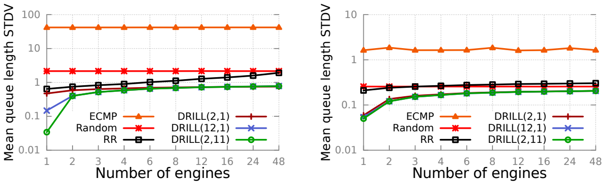

Figure 2: (a) 80% load. (b) 30% load. Adding a choice and a memory unit improves performance dramatically.

compare it with ECMP, per-packet Random, and RR using the following methodology: We build Clos datacenters of different sizes in a packet level simulator (details in §4), draw flow sizes and interarrival times from [62], and scale the interarrival times to emulate various degrees of network load. Given that the dominant source of latency in datacenters is queueing delay [20], in this section, our goal is to minimize queue lengths. (§4 will measure higher level metrics such as flow completion times and throughput.) An ideal load balancer should balance the load across uplink queues of each leaf switch as well as across the spine layer's downlink queues connected to the same leaf switch. Hence, as the performance metric, at every 10μsec during the 100 sec experiments, we measure the standard deviation (STDV) of uplink queue lengths for each leaf switch and the STDV of queue lengths of all downlinks of spine switches connected to each leaf switch. ESF would theoretically keep this metric constantly zero, and we strive to get close to zero.

Small amounts of choice and memory dramatically improve performance. Our experiments show that in networks with different sizes, deploying switches with different number of engines, and under high and low load, adding a slight amount of choice and memory, *e.g.,* `DRILL(2,1)` instead of per-packet Random or RR, significantly improves the load balancing performance especially under heavy load. In networks with 48 spines and 48 leafs each connected to 48 hosts, for instance, under 80% load, `DRILL(2,1)` reduces the *avg.* STDV of queue lengths by over 65% compared to per-packet Random, irrespective of the number of engines in each switch (Figure 2 (a)). per-packet Random, in turn, improves upon ECMP by around 94% as a result of its finer grained, per-packet operations. DRILL's improvement upon RR is more pronounced for switches with large number of engines, but even with single-engine switches under load, DRILL achieves more balanced queues as its load sensing based on queue sizes allows it to balance out the packet size variances. When the network is less loaded and switches have more engines, the improvement is less dramatic. As an example, under 30% load, `DRILL(2,1)` outperforms per-packet Random and RR by around 20% if the network is built out of 48 engine switches, and by over 75% with single-engine ones (Figure 2 (b)).

Too much memory and too many choices may degrade performance. While a few choices and units of memory improve performance dramatically, excessive amounts of them degrade the performance for switches with large number of engines (more than 6 in our experiments) under heavy load. Figures 3 shows an example for a network with 48-engine switches under 80% load. While the first extra choice, *i.e.,* `DRILL(1,2)` *vs.* `DRILL(1,1)` reduces the mean queue length STDV by 11%, having 20 choices, *i.e.,* `DRILL(1,20)`, increases this metric by 8%. The reason is that the larger number of random samples or memory units makes it more likely for a large number of engines to simultaneously select the same set of output ports which will in turn cause bursts of packets at those ports. We call this phenomenon *synchronization effect*. The resulted load imbalance may cause more queueing delays, *e.g.,* while the 99.999$^{\mathtt{th}}$ percentile of queue lengths is below 1 under `DRILL(1,2)` (*i.e.,* the queues are almost always empty), the 99$^{\mathtt{th}}$ percentile of queue lengths under `DRILL(1,20)` is slightly larger than 1, *i.e.,* under `DRILL(1,20)`, in 1% of the cases, packets experience some queueing latency because of the synchronization effect. For other cases (under light load or having fewer engines), setting $d \gg 2$ and $m \gg 1$ results in more balanced load, but the impact on queue lengths is marginal given that the queues are already almost perfectly balanced under `DRILL(2,1)`. With one engine switches under 80% load, for example, while the mean queue length STDV is lower in `DRILL(12,1)` compared to `DRILL(2,1)`, the 99.999$^{\mathtt{th}}$ percentile of queue lengths is under 1 for both, *i.e.,* packets rarely experience any queueing delays.

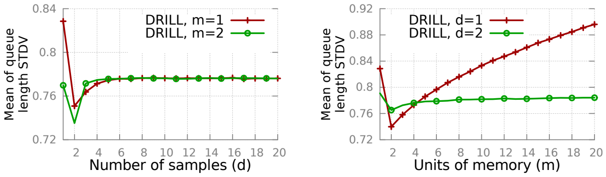

Figure 3: Excessive choices & memory cause sync effect.

3.2.4 *DRILL guarantees stability.*

A system is *stable* if the expected length of no queue grows without bound [52]. We consider an $M \times N$ combined input output queued switch with FIFO queues in which the arrivals are independent and packets could be forwarded to any of the N output ports. We assume traffic admissible, *i.e.,* $\sum_{i=1}^{M} \delta_i \leq \sum_{i=1}^{N} \mu_i$, where δ_i is the arrival rate to input port i and μ_j is the service rate of output queue j. We place *no restriction on the heterogeneity of arrival rates or service rates.* These rates can be different and could dynamically change over time. Particularly, we focus on the more interesting and more challenging case where service rates could vary over time because of various reasons such as failures and recoveries that are common in data centers [39]. We first prove that purely randomized algorithms without memory, *e.g.,* `DRILL(d,0)`, are unstable then prove the stability of `DRILL(d,m)` for $m > 0$.

Pure random sampling is unstable. First, we consider `DRILL(d,0)`, *i.e.,* the algorithm in which every forwarding engine chooses d random outputs out of possible N queues, finds the queue with minimum occupancy between them and routes its packet to it. Theorem 1 proves that such algorithm is unstable.

Theorem 1. *For admissible independent arrival processes,* `DRILL(d, 0)` *cannot guarantee stability for any arbitrary number of samples $d < N$.*

Proof. For a switch with F forwarding engines, let γ_i be the arrival rate to engine i, and μ_j be the service rate of output queue j. $\sum_{i=1}^{F} \gamma_i = \sum_{i=1}^{M} \delta_i$. Now consider output queue I. For any forwarding engine, the probability that it chooses I as a sample is $\frac{d}{N}$. I will receive the *maximum* arrival rate if any engine that samples it also selects it[3]. So its maximum arrival rate is $\frac{d}{N} \times \sum_{i}^{F} \gamma_i$. Thus, the minimum arrival rate to the remaining N-1 output queues is $\zeta =$

[3] This happens if I's length is not larger than the other samples of the engine, *e.g.,* if $\mu_I \gg \sum_{i=1}^{M} \delta_i$, I's length=0 and always equal or shorter than all queues.

$\sum_i^F \gamma_i - \frac{d}{N} \times \sum_i^F \gamma_i = (1 - \frac{d}{N}) \times \sum_i^F \gamma_i$. Clearly, if ζ is larger than the sum of the service rates of these N-1 queues, the system is unstable.

Note that the argument does not hold: (a) when there are some restrictions on arrival or service rates, *e.g.,* when the service rates are equal, or (b) when d=N. These special cases, however, are of little interest, since the former opts out some admissible traffic patterns, and the latter nullifies the benefit of randomization and may cause a synchronization effect (§3.2.3). Our experiments show that DRILL performs well with $d \ll N$.

Random sampling with memory is stable. We showed above that randomized policy cannot guarantee stability without using unit of memory. Similar to [52] and using the results of Kumar and Meyn [47], we prove that DRILL's scheduling algorithms are stable for all uniform and nonuniform independent arrival processes up to a maximum throughput of 100%. Note this result shows maximum throughput achieveable by any packet scheduling algorithm, but throughput may still be limited by routing sub-optimality.

Theorem 2. *For all admissible independent arrivals,* `DRILL(1,1)` *is stable and achieves 100% throughput.*

To prove that the algorithm is stable, we show that for a $M \times N$ switch scheduled by `DRILL(1,1)`, there is a negative expected single-step drift in a Lyapunov function, V. In other words, $E[V(n+1) - V(n)|V(n)] \leq \epsilon V(n) + k$, where $k, \epsilon > 0$ are some constants. We do so by defining $V(n)$=$V_1(n)$+$V_2(n)$, $V_1(n)$=$\sum_{i=1}^N V_{1,i}(n)$, $V_{1,i}(n)$=$(\tilde{q}_i(n) - q^*(n))^2$, $V_2(n)$=$\sum_{i=1}^N q_i^2(n)$. $q_k(n)$, $\tilde{q}_i(n)$, and $q^*(n)$ represent the lengths of the k-th output queue, the output queue chosen by the engine i, and the shortest output queue under `DRILL(1,1)` at time instance n. Details of the proof are in [37].

3.3 Packet reordering

DRILL makes forwarding decisions for each packet, independent of other packets of the same flow, based on the local and potentially volatile switch load information. One might expect this to cause excessive degrees of packet reordering that could degrade TCP performance by triggering its duplicate ACK mechanism, one of the primary means of TCP for detecting packet loss. As explained in RFC 2581 [21], when a segment arrives out of order, the receiver immediately sends a "duplicate ACK" to the sender. The sender uses the TCP *retransmission threshold*, the arrival of *three* duplicate ACKs, as an indication of packet loss and congestion. It reacts by retransmitting the packet perceived lost and reducing its transmission rate. Reordering may also increase CPU overhead as optimizations such as GRO that merge bursts of incoming packets depend on in-order delivery [35, 42]. Wary of these issues, the majority of load balancing schemes, from ECMP to CONGA [19] to Presto [42], avoid reordering by balancing coarser units of traffic.

However, it turns out DRILL causes minimal reordering. This may be somewhat surprising, but using multiple paths only causes reordering if the delays along those paths differ by more than the time between packets in a flow. Queueing delay is famously the dominant source of network delay in datacenters [20], and DRILL's well-balanced load and extremely low variances among queue lengths (Figure 2) imply that packets experience almost identical queueing delays irrespective of the paths they take. So even though flows' packets take divergent paths at very fine granularity, they should not be reordered frequently. Our experiments, using the actual TCP implementations taken from Linux 2.6, confirm this hypothesis and show that TCP performance is not significantly impacted (§4). However, for certain legacy or specialized applications it may be desirable to eliminate all reordering. This can be accomplished using recent techniques for building reordering-resilient network stacks via adding an end-host shim layer [35, 42]. In §4 we evaluate both variants of DRILL, with the shim and without.

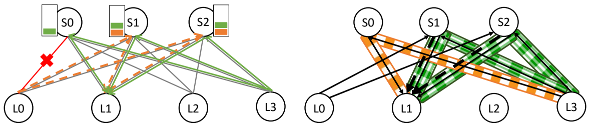

Figure 4: (a) L0-S0 link failure makes the topology asymmetric. $L_3 \rightarrow L_1$ traffic (heavy green lines) experience more congestion at S_1 and S_2, where they collide with $L_0 \rightarrow L_1$ traffic (dashed orange lines) than S_0. (b) Partial Quiver. $L_0 \rightarrow L_1$ paths add edges from S_1 and S_2 to L_1 (dashed lines), but not from S_0 to L_1. So the path via $S_0 \rightarrow L_1$ (with the orange stripes pattern) is asymmetric with those with the green plaid pattern.

3.4 Handling asymmetry

Until now, DRILL's design has assumed a symmetric network. An initially symmetric network may experience failures that cause asymmetry, for example if a link from a leaf to a spine fails. When this occurs, two key problems arise.

First, load balancers that split individual flows among available paths may waste bandwidth because of their interactions with TCP's control loop, as noted in [19]. This happens because the asymmetric paths available to a flow may have different and varying capacities for it (depending on the load of other flows that use those paths). Flow rates on each path are controlled by TCP to avoid congestion. So splitting the load of the flow equally among asymmetric paths effectively limits its rates on each path to the rate of the most congested path. This implies that the paths with more capacity will be left underutilized even if the flow has a demand for their bandwidth.

As a simple example, consider Figure 4 (a) where hosts under leaf switches L_0 and L_3 have infinite TCP traffic demands to send to those under L_1. Assume that the L_0-S_0 link fails and that all links have 40Gbps capacity. Under local schemes such as ESF, this link failure can cause collateral damage to the flows going through other links. This happens because the flows from L_0 and L_3 sent to S_1 and S_2 share the bottleneck links $S_1 \rightarrow L_1$ and $S_2 \rightarrow L_1$. Assuming that the number of these flows are equal and they are all in the steady state, TCP does not allow the rate of the flows from L_3 that take either of these two paths P_1:$L_3 \rightarrow S_1 \rightarrow L_1$ and P_2:$L_3 \rightarrow S_2 \rightarrow L_1$, to increase beyond 20Gbps to avoid congestion on $S_1 \rightarrow L_1$ and $S_2 \rightarrow L_1$. Now if the load balancer tries to keep the load on path P_0:$L_3 \rightarrow S_0 \rightarrow L_1$ equal to that on paths P_1 and P_2, it keeps the rate on P_0 also equal to 20Gbps, despite the fact that P_0 can serve traffic at 40Gbps. In other words, 50% of P_0's capacity will be left idle. Without its mechanism for handling asymmetry, DRILL will have the same deficiency — by balancing the queues, L_3 effectively limits P_0's utilization to half of its capacity. Note that some other local load balancers also suffer from this problem. Presto's failover mechanism [42], for example, prunes the spanning trees affected by the failure and uses a static weighted scheduling algorithm, similar to WCMP [73], over the remaining paths. In this example, since P_0, P_1, and P_2 have static capacity of 40Gbps each, their associated weights will be equal and Presto continues to equally spread $L_3 \rightarrow L_1$'s load across them.

Note that changing weights in a load-oblivious manner does not solve this problem since optimal weights depend on the load from other sources—a potentially rapidly evolving parameter. In the above example, for instance, optimal weights are $w(P_0) = 2$ and $w(P_1) = w(P_2) = 1$ for the given scenario, but if L_0 stops sending traffic to L_1, those weights leave P_1 and P_2 underutilized.

In addition to this *bandwidth inefficiency* in the asymmetric case, schemes such as Presto and DRILL that split flows across asymmetric paths may cause *excessive packet reordering*. In the example above, packets traversing P_1 experience higher queueing delay than those traversing P_0 since S_1 is more congested than S_0, and thus may arrive out of order with respect to packets sent along P_0.

We observe that rather than being a fundamental deficiency of local load balancers, both problems are rooted in imposing *rate dependencies* across asymmetric paths, *e.g.,* keeping the rates on $P_{i\in\{0,1,2\}}$ equal in the example above. Intuitively, to solve these problems, DRILL needs to break these dependencies. To achieve this, DRILL introduces control plane and data plane changes. We first present DRILL's control and data plane algorithms for handling asymmetry in multi-stage topologies with homogeneous links in each stage, before showing how to handle heterogeneous links.

3.4.1 Control plane operations. DRILL's control plane identifies symmetric path groups, and hands these groups over to the data plane for micro load balancing within each group. This may be performed locally at each switch utilizing OSPF's link state database, or centrally in an SDN network. In either case, the algorithm proceeds as follows.

Defining symmetry precisely: DRILL does not need the entire topology to be symmetric in every way. What functionally affects DRILL is its choice between alternate paths: downstream, the paths should be similar in terms of their queueing, regardless of the current traffic pattern. But queueing along a path depends on traffic from other source-destination pairs that collides with it. In Fig. 4(a), for instance, the path $L_3S_1L_1$ may have higher queueing than $L_3S_0L_1$ because only the former shares a link with $L_0 \rightarrow L_1$ traffic.

With this in mind, we define two links ℓ_1, ℓ_2 as symmetric, written $\ell_1 \sim \ell_2$, if they are traversed by the same set of source-destination pairs, according to the paths selected by the routing protocol. Now consider two paths P and Q consisting of links $p_1, \ldots, p_n$ and $q_1, \ldots, q_m$, respectively. We define P and Q as **symmetric**, written $P \sim Q$, if they have the same number of hops (*i.e.,* $n = m$) and the corresponding links are symmetric (*i.e.,* $\forall i \in (1, \ldots, n), p_i \sim q_i$).[4]

To understand this definition, consider two examples. First, it is easy to see that in regular Clos-based datacenters, *i.e.,* all shortest paths from a source to a destination are symmetric; and now suppose a link from a host h to its top-of-rack switch fails. Then symmetry is still satisfied because flow to and from h is removed equally from all links. In practice, the routing protocol's mechanisms will purge the failed entries from forwarding tables and DRILL continues to distribute traffic among the remaining paths using its mechanisms as presented earlier. Thus, not all failures cause asymmetry. Next, consider the earlier example of Figure 4. P_1 and P_2 are symmetric, but P_0 and P_1 are asymmetric because $(S_0, L_1)\nsim(S_1,L_1)$ due to only the latter carrying $L_0 \rightarrow L_1$ traffic.

Theorem 3 in [37] shows that, for admissible independent traffic, performing micro load balancing only among symmetric paths is sufficient to guarantee DRILL's stability and 100% throughput in Clos networks. Thus, we have a natural way for DRILL to decompose irregular Clos networks into symmetric components, and balance load locally. Next, we compute these components.

Step 1: Building a Quiver: We construct a labeled multidigraph (a directed graph with labeled edges where edges are permitted to have the same end vertices and labels) that we call the *Quiver*. For each switch in the network, we create a node in the Quiver. We then compute shortest paths between all pairs of leafs in the network. For every link from switches a to b which is on a shortest path p from leafs *src* to *dst*, we add to the Quiver a directed edge from a to b labeled *(src,dst)*. Computing all labels of the link can be done in polynomial time — with $|L|$ leafs and V vertices, it requires $O(|L|^2V^2)$ because for every $|L|^2$ pair of leafs, checking if the link appears on shortest paths between them can be checked in $O(V^2)$ with the Dijkstra algorithm. Figure 4(b) demonstrates this by showing part of the Quiver that results from the network of Figure 4(a); links that have the same labels are shown in the same color.

Step 2: Decomposing the network into symmetric components: After building the Quiver, each source switch decomposes its set of shortest paths towards a destination, *dst*, into *components:* largest subsets that contain only paths that are symmetric with each other. For faster path symmetry checks, DRILL uses a hash function which maps sets of edges from vertex a to b in the Quiver to a numerical value that we call the *score* of (a,b). The score of path p, *p.score*, is then the list of its links' scores. In the previous example, assuming that hash(L_3, S_1)=1 and hash(S_1, L_1)=2, P_1.score is <1,2>. Two paths are symmetric if their scores are equal. In addition to computing the score of a path p, DRILL also computes its capacity, *p.cap*, which is the capacity of p's slowest link. The complexity of *p.score* and *p.cap* computations is $O(d)$, where d is p's length. After these computations, DRILL iterates over *unassigned*, the set of paths that are not yet assigned to any component (it initially includes all the shortest paths towards *dst*). At each iteration, it assigns all symmetric paths in *unassigned* to a component and removes them from *unassigned*. In the example above, the set of $L_3 \rightarrow L_1$ paths is decomposed into two components: $\{P_0\}$ and $\{P_1, P_2\}$.

DRILL also assigns a weight to each component which is proportional to the aggregate capacity of its paths. In the example above, $P_{0\leq i\leq 2}.cap$=40Gbps. Hence: $\{P_0\}.w$=1 and $\{P_1, P_2\}.w$=2. This weight assignment is similar to the path weight assignments in [42, 73] and can be implemented in switches with the techniques discussed in [46].

The complexity of decomposing a set is $O(|P|d)$, where $|P|$ and d are respectively the number of paths in that set and the maximum path length. Note that the decomposition of a set does not affect other sets' decomposition, which makes the computation easy to parallelize if necessary.

An example in a multi-stage topology: Figure 5 shows a schematic example of a Clos-based network deployed in Facebook [2] where a pod consists of a number of server racks and fabric switches. A folded Clos connects ToRs and fabric switches in each

[4] Note that while sufficient for approximating similar queueing behavior across paths, this condition is not necessary; it is possible that two asymmetric paths (*e.g.,* have different number of hops) still have overall similar delays. Detecting such similarities, however, would require exchanging load information and coordinating forwarding decisions between switches. DRILL does not take this approach as it would fundamentally add to its latency and weaken its capability to react to microbursts.

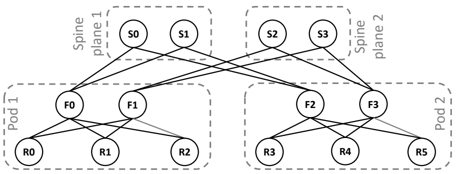

Figure 5: Handling asymmetry in multi-stage topologies.

pod. Pods are interconnected via spine planes. Each fabric switch of each pod connects to each spine switch within its local plane, *i.e.,* the first fabric switch of each pod connects to all the switches of the first spine plane. If link (R_0, F_0) fails, the $R_5 \rightarrow R_1$ paths traversing the first spine plane, $Q_0 = R_5F_2S_0F_0R_1$ and $Q_1 = R_5F_2S_1F_0R_1$, and those that traverse the second plane, $Q_2 = R_5F_3S_2F_1R_1$ and $Q_3 = R_5F_3S_3F_1R_1$, are asymmetric as some links in Q_2 and Q_3, unlike Q_0 and Q_1, can also carry traffic from Pod 2 to R_0 and have extra edges in the Quiver; $F_3 \rightarrow R_1$ paths by contrast are all symmetric.

3.4.2 Data plane operations. After detecting the symmetric components, each switch follows a two step process for forwarding each packet: **Flow classification:** By hashing the 5-tuple header of each packet like ECMP, DRILL assigns it to a component considering the weights set in the previous step. **Intra-component micro load balancing:** DRILL uses `DRILL(d,m)` to balance the load across its symmetric paths.

3.4.3 Handling heterogeneous devices. In addition to failures, *heterogeneous links* (same stage links with different capacities) and *imbalanced striping* [73] (unequal number of links from a leaf to spines) make the topology asymmetric. As an example, in Figure 4(a), if L_0 is connected to S_0 and L_1 is connected to S_0 with either one 40Gbps link or four 10Gbps links while all other pairs of leaf and spine switches are connected with one 10Gbps link, then some paths including those from $L_0 \rightarrow L_1$ are asymmetric and splitting flows across them may cause reordering and bandwidth inefficiency.

In general, with heterogeneous devices, queueing at switch a towards b is a function of not just the flows that traverse (a,b), but also the rate the flows send traffic to a and a's rate of sending that traffic to b. To capture these two additional criteria, we define the *capacity factor* $cf()$. For path p from *src* to *dst* that traverses (a,b), *cf(a,b,p)* is defined as the input rate of this path divided by its output rate: $\texttt{capacity}(src, a)/\texttt{capacity}(a, b)$, where $\texttt{capacity}(a, b)$ and $\texttt{capacity}(src, a)$ are, respectively, the capacity of the link from a to b and the capacity of p from *src* to a, *i.e.,* the capacity of the bottleneck link from *src* to a on this path. It models the rate at which the *src*→*dst* traffic builds a queue at a towards b if *src* only uses p towards *dst*. For the source vertex, we consider the capacity factor to be ∞, *i.e.,* if *a*=*src*, then *cf(a,b,p)*=∞. An edge label in the Quiver then also includes its capacity factor: when building the Quiver, for each (a,b) link on path p between *src* and *dst*, we add a directed edge from a to b to the Quiver labeled *(src,dst,cf(a,b,p))*. Other steps of the control and data plane algorithms are exactly as before. Our stability and 100% throughput results cover this case (Theorem 3 in [37]). As an example, in Fig. 4(a) if L_0-S_0, L_0-S_1, and L_1-S_0 links are 40Gbps and other links are 10Gbps, among the $L_0 \rightarrow L_1$ paths $H_0 = L_0S_0L_1$, $H_1 = L_0S_1L_1$, and $H_2 = L_0S_2L_1$, we have $H_0 \sim H_2$ but $H_0 \nsim H_1$.

4 EVALUATION

We evaluate DRILL in detailed simulations. We find that DRILL achieves high performance, for example 1.3×, 1.4×, 1.6× lower mean FCT than Presto, CONGA, and ECMP, respectively, under 80% load. Both our fine granularity and load-awareness are important factors in that performance, with the second becoming more important in highly bursty traffic patterns such as incast, and with link failures. DRILL is especially effective in handling incast as it is the most agile load balancer to react to spontaneous load bursts; it results in 2.1× and 2.6× lower 99.99**th** percentile of FCT compared to CONGA and Presto, respectively. We also show DRILL has minimal packet reordering, and explore the effect of failures, synthetic traffic patterns, scaling out, and scaling up. Finally, we implemented DRILL in Verilog to evaluate deployability. Details of these evaluations follow.

Performance evaluation methodology. To test DRILL's performance at scale, we measure flow completion times (FCT) and throughput under DRILL, and compare it with CONGA, Presto, and ECMP via simulation. We use the OMNET++ simulator [16] and the INET framework [11], with standard Ethernet switches' and hosts' networking stacks. We port real-world TCP implementations from Linux 2.6 via the Network Simulation Cradle library [15]. For DRILL, unless stated otherwise, we use single engine switches under `DRILL(2,1)`. We use 2 and 3 stage Clos networks with various sizes, without failures and with multiple link failures, under a set of realistic and synthetic workloads, and an incast application.

In a symmetric Clos, DRILL reduces mean and tail latencies. We use trace-driven workloads from real datacenter traffic for flow sizes, flow interarrival times, and traffic pattern from [62], and use a Clos with 4 spine and 16 leaf switches, each connected to 20 hosts. The links connecting spines and leafs are 40Gbps and those between hosts and leafs are 10Gbps. To emulate various degrees of the offered load, we scale flow interarrival times. Under this setting (Fig. 6), we find the load balancing granularity to be a key player in the effectiveness of the load balancer. DRILL achieves lower FCT compared to Presto which in turn has lower FCT than CONGA. (Later, we will see that load-awareness is important too even in the symmetric case.) The difference is larger under heavy load, *e.g.,* under 80% load, DRILL reduces the mean latency of Presto and CONGA by 1.3× and 1.4×, respectively (Figure 6). We also test a strawman "per-flow DRILL" which makes load-aware decisions for the first packet of a flow and then pins the flow; this marginally improves the tail latency of Presto and CONGA while being coarser grained than both.

To see where this improvement comes from, we measure queueing time at each hop (Fig. 6(c)): leaf upward to spine (Hop 1), spine downward to leaf (Hop 2), and leaf to host (Hop 3). At 10% load, queueing happens mostly at the last hop, *e.g.,* under ECMP, the last hop queuing time accounts for 97.6% of a packet's end-to-end queueing time. At Hop 3, there is no path choice and none of the load balancing schemes offer noticeable benefits — the mean FCT improvements of DRILL, CONGA, and Presto over ECMP are less than 9% — but in any case, the total queueing time is almost invisibly tiny. At 50% and 80% load, we see there is negligible queueing at hop 2 and very little difference among schemes at Hop 3. The load balancers' benefit in this scenario stems primarily from shorter

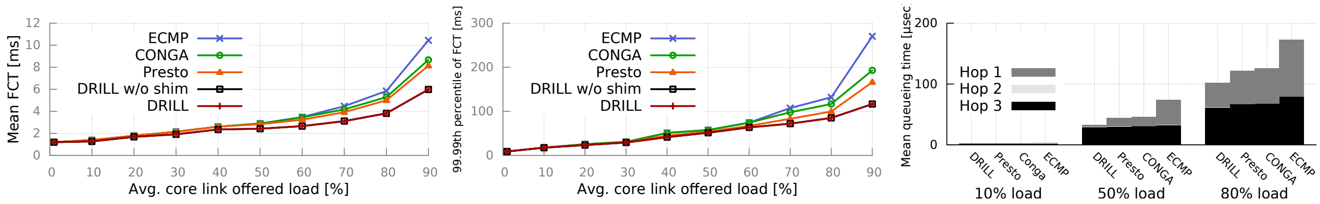

Figure 6: DRILL improves (a) mean and (b) tail latency in a symmetric Clos. (c) It shortens upstream queues under load.

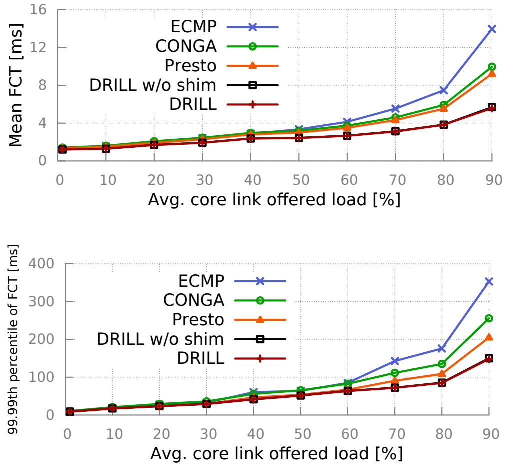

Figure 7: DRILL handles scaling-out gracefully.

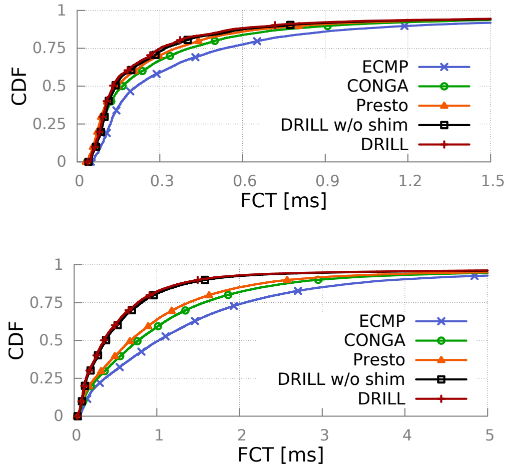

Figure 8: Scale-out topology with (a) 30% load (b) 80% load. DRILL's improvement is greater under heavy load.

upstream queues (*i.e.,* Hop 1): under ECMP the mean upstream queue delay comprises 54% of the overall queueing delay of a packet with 80% load. DRILL, Presto, and CONGA cut Hop 1 queueing time by 2.3×, 1.7×, and 1.6× respectively.

Datacenters today experience high congestion drops as utilization approaches 25% [64]. Thus, the average load is kept around 25% to control the latency [62, 64]. We note that compared to ECMP, DRILL allows the providers to use 10% more of their bandwidth capacity while keeping the 99.99**th** percentile of FCT lower than ECMP's under 25% load. That is, DRILL supports 1.4× higher load with the same tail FCT performance compared with ECMP, 1.3× higher than CONGA, and 1.2× higher than Presto.

DRILL scales out and up gracefully. We study the impact of applying the "scale-out" approach, *i.e.,* we use a larger number of less capable switches, instead of fewer but more powerful ones, to build a network with identical overall core capacity. Figure 7 shows the result for a network with 16 spines and 16 leafs, each connected to 20 hosts where all links are 10Gbps. Note that this network provides identical core capacity as the previous experiment while using switches with slower links. We apply the same load and observe that the performance of all schemes, including DRILL, degrades. This is because with slower links, queues drain more slowly; hence the negative impact of suboptimal load balancing decisions is greater. However, DRILL's gain over other schemes is more pronounced in this case. That is, DRILL handles scaling out more gracefully. The difference is greater under heavy load, *e.g.,* under 80% load, it cuts the mean latency of ECMP, CONGA, and DRILL by 2.1×, 1.6×, and 1.4× respectively. Figure 8 shows the FCT CDFs for 30% and 80% load.

We see no significant performance differences in networks with different sizes and oversubscription ratios, but with identical load and link speeds. Figure 9(a) and 9(b) show 2 examples of CDFs of FCT for 2 networks with, respectively, 20 spines, 16 leafs each connected 20 hosts (*i.e.*, over-subsection ration of 1:1), and 12 spines, 16 leafs each connected 20 hosts (*i.e.*, over-subsection ration of 5:3) where all links are 10Gbps and the load is 80% in both cases.

We also test DRILL's ability to balance load in Clos topologies with more than 2 stages such as VL2 [41] and fat-tree [49]. Figure 10 shows the result of an experiment with a VL2 network with 16 ToR switches, each connected to 20 hosts via 1Gbps links, 8 Aggregate switches, and 4 Intermediate switches. Core links are 10Gbps[5]. We put 20% and 70% load on the network. Figure 10 shows that DRILL is effective in keeping the FCT short in such networks.

We also tested the effect of scale in terms of number of forwarding engines in each switch and find its impact to be negligible on FCT for `DRILL(2,1)`, *e.g.,* we find less than 1% difference in the mean FCT between 1- and 48-engine switches under 80% load (no plot).

DRILL has minimal packet reordering. The previous figures show that FCT is low despite reordering, but next we dig deeper to see why. Figure 11 (a) shows amount of reordering measured in terms of the number of TCP duplicate ACKs under 80% load, using the same setup as our first experiment in this section. ECMP and

[5] All CONGA switches send load feedback, ToR and Aggregate ones apply CONGA's load balancing decisions; cores apply ECMP.

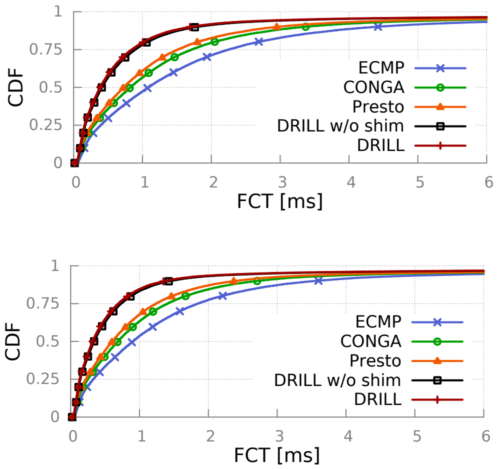

Figure 9: (a) 1:1 and (b) 5:3 oversubscription ratios.

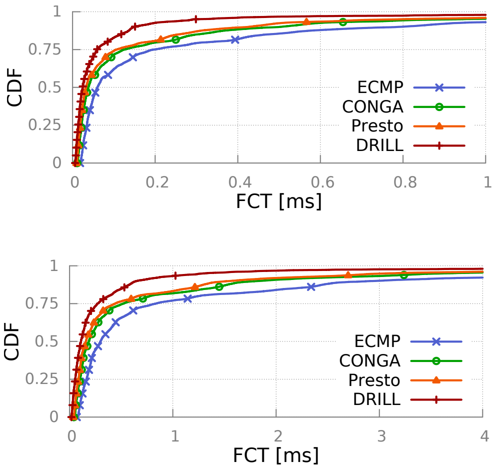

Figure 10: A VL2 network under (a) 20% and (b) 70% load.

CONGA, unlike DRILL and Presto, do not cause reordering. As a strawman comparison, we also show the amount of reordering in Per-packet Random (which forwards each packet along indepedent-random shortest paths) and Per-packet Round Robin (RR) (which fowards each packet round-robin along shortest paths), both with no load-awareness, along with Presto measured prior to its shim layer. We note two important conclusions. First, Per-packet Random, RR, and DRILL have identical granularity of load balancing but DRILL has dramatically lower packet reordering. This demonstrates how local load awareness keeps queues extremely well-balanced across paths.

Second, the degree of reordering under DRILL rarely reaches the TCP retransmission threshold even under heavy load. Under 80% load, for example, only 0.4% of flows have any duplicate ACKs. This is 8×, 7.3×, and 3.3× lower than Per-packet Random, RR, and Presto before its shim. Plus, under DRILL, only 0.02% of flows have more than the typical TCP retransmission threshold of 3. Note that compared to Per-packet Random and RR, Presto causes reordering for fewer flows, but increases the number of flows that trigger excessive (*e.g.*>3) duplicate ACKs. This is because Presto's coarser-grained flowcell forwarding shields most flows from reordering—the majority of flows fit in a single flowcell [42]. However, on the occasion that flowcells are reordered, their larger number of packets causes more duplicate ACK generations compared to the per-packets schemes. Under low load, very few flows receive any duplicate ACKs, *e.g.,* under 20% load, less than 0.8% of flows receive any duplicate ACKs under any of these schemes. This minimal degree of reordering shows why DRILL with and without the shim layer have very similar performance.

Reordering can also increase receiver host CPU overhead. Operating systems usually perform optimizations such as Generic Receive Offload (GRO) to merge bursts of incoming packets to reduces per-packet processing overhead [35, 42]. GRO performs per-flow packet batching by merging a flow's packets as long as they arrive in-order and the batch's size is below a threshold (64KB). If the batch's size exceeds that threshold or if an out of order packet is received, GRO sends the batch to the higher layer. Hence, excessive reordering increases the number of batches and consequently the CPU overhead. However, since DRILL seldom causes reordering even under load, it has negligible effect on GRO's performance. Under 80% load, for example, DRILL increases the number of batches by less than 0.5%.

DRILL gracefully handles failures. Even though high scale datacenters show high reliability [39], with the majority of links having higher than four 9's of reliability [39], there is still a high probability of at least one failure at each point in time [40, 73]. Therefore, handling failures gracefully is imperative for any load balancer. We test the performance of DRILL under 3 failure scenarios: (a) one single leaf-spine link failure, as single failures are the most common failure cases in datacenters [39], (b) 10 randomly selected leaf-spine link failures. (b) presents a rare, but still possible, case. Even in large scale datacenters, big groups of correlated link failures are rare with only 10% of failure groups (failures with downtimes either simultaneous or close together in time) containing more than four failures [39]. We load the system up to 90% of the available core capacity. We observe that DRILL and Presto are most effective at handling a single failure while DRILL and CONGA are most effective in handling multiple failures (Figures 11 (b,c) and 12). This is because CONGA shifts the load towards the parts of the topology with more capacity, and DRILL breaks the rate interdependencies between asymmetric paths, effectively allowing flows to grab the available bandwidth, increase their rates, and finish faster. Note that in all these cases, DRILL's performance with and without the shim layer that reorders the out-of-order packets (from [42]) are almost identical, since its degree of reordering is so low that it rarely reaches TCP's retransmission threshold (§3.3).

Since DRILL relies on OSPF to learn about topological changes, it is limited by the reaction and propagation delay of this protocol to react to topological changes. In our experiments, the impact of this delay is negligible. In an experiment under 70% load and with 5 link failures, *ideal-DRILL*, an idealized variant of DRILL that learns and reacts to failures instantaneously, achieves a median FCT of 3.49 ms, indicating an improvement of less than 0.6% over DRILL.

DRILL is efficient in asymmetric topologies with heterogeneous devices. As explained in §3.4, DRILL can operate in asymmetric topologies, *e.g.,* those with heterogeneous links and imbalanced striping [73]. Figure 13 shows the result of an experiment with

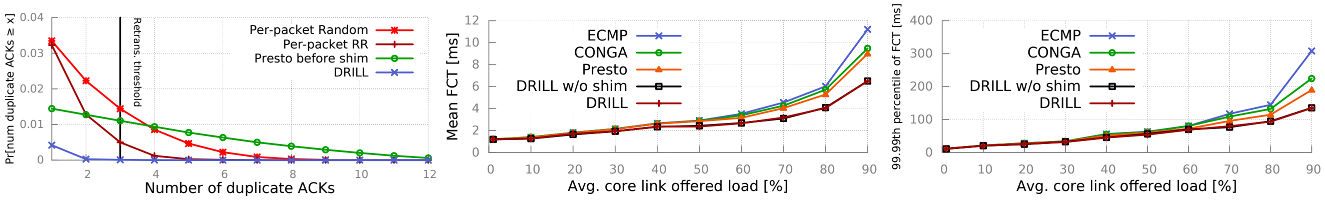

Figure 11: (a) Less than 0.1% of flows with DRILL hit TCP retrans. threshold, (b,c) DRILL handles single link failure.

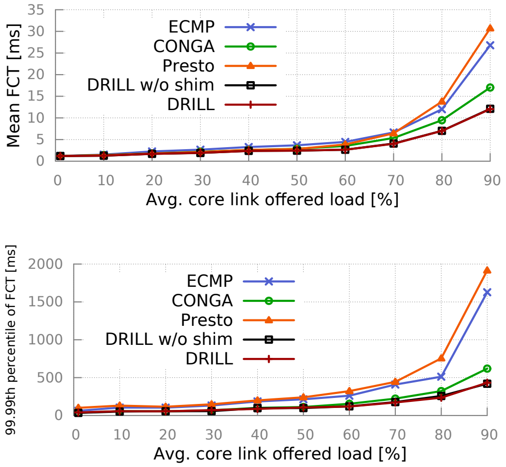

Figure 12: DRILL handles 10 link failures.

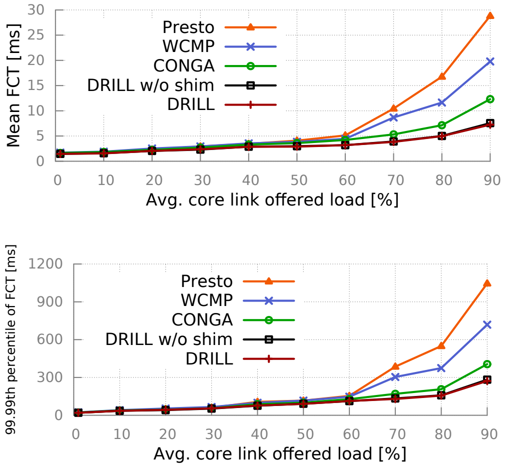

Figure 13: DRILL is efficient in heterogeneous topologies.

a topology with 16 leafs, $L_{i\in\{0,\ldots,15\}}$, each connected to 48 hosts via 10Gbps links, and 16 spines, $S_{i\in\{0,\ldots,15\}}$. Each leaf L_i is connected to spines $S_{i(\text{mod})16}$ and $S_{(i+1)(\text{mod})16}$ via two 10Gbps links and to every other spine with one 10Gbps link. In addition to CONGA and Presto, we compare DRILL with WCMP [42] which is designed to improve ECMP's performance in asymmetric Clos. We observe that DRILL and CONGA are more effective in such topologies and achieve lower FCT than Presto and WCMP.

DRILL reduces the tail latency in incast scenarios. A common and vexing traffic pattern in datacenters is incast [20, 64] which is responsible for excessive congestion and packet loss [64]. With the exception of a recent study from Google that reports incast-induced packet drops at various layers [64][6], most of the works on incast study the problem within a cluster (hosts connected via one switch or a tree topology), and naturally focus exclusively on overrun of the last hop buffer (connected to the receiver) [29, 30, 33, 43, 55, 60, 69–72]. Our experiments show that in multi-rooted topologies, the incast traffic pattern triggers buffer overruns at other layers as well. Furthermore, our results underscore the fact that this problem is interwoven with load balancing and can be mitigated by an agile load balancer capable of reacting to microbursts. Figs. 14(a, b) show an example for a network similar to our first experiment in this section under the typical load of 20% and 35%, respectively, where hosts run an incast application similar to [69], and 10% of them send simultaneous requests for 10KB flows to 10% of the other hosts (all randomly selected). The background traffic and interarrival times are drawn from [62] as before. DRILL significantly reduces the tail latency, *e.g.,* under 20% load, it has 2.1× and 2.6× lower $99.99^{\textbf{th}}$ percentile of FCT than CONGA and Presto, respectively. This happens because as this highly bursty traffic pattern causes microbursts at the first hop, DRILL can swiftly divert the load and mitigate congestion on hotspots. By better balancing the load across the spine, it also alleviates the risk of hotspots forming in spines. Fig. 14(c) shows where queueing and packet loss happen. DRILL almost eliminates the first hop queueing and drops, and significantly reduces those metrics in the second hop.

Synthetic workloads. In addition to the trace-driven workload, similar to previous works [18, 42, 49, 61], we use a set of synthetic workloads, known to either appear frequently in datacenters or to be challenging for load balancing designs [18]: *Stride(x)* in which server[i] sends flows to server[(i+x) mod number of servers], *Random* where each server communicates with a random destination not under the same leaf as itself. We use Stride(8), and *Shuffle* in

[6] [64] reports that in Google's "Saturn" fabric, 62.8% of drops occur at the last hop, *i.e.,* into the host. DRILL offers little to no benefit at the last hop, since multipath flexibility only appears at earlier hops. However: (1) There is still opportunity for significant improvement in the remaining 37% of drops. (2) [64] reports implementing various special enhancements to reduce drops, *e.g.,* enabling ECN on switches, optimizing the host stack response to ECN signals, and bounding TCP windows (§6.1 in [64]). Some of these techniques, *e.g.,* link-level pauses at ToRs, specifically target the uplink queues' congestion. DRILL is a simpler design that may present an alternative way of achieving some of the same benefits. An evaluation of DRILL applied to Saturn would be interesting, but more immediately, Saturn does not appear to be broadly representative of most datacenter fabrics.

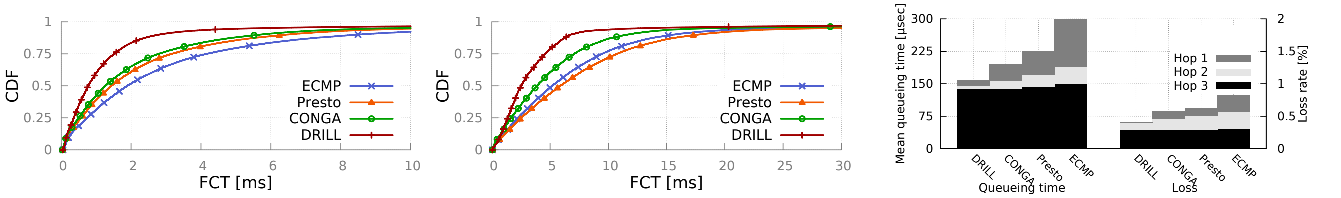

Figure 14: Incast: (a, b) DRILL cuts the tail latency under 20% and 30% load, (c) Where queueing and loss happen under 20% load, across hop 1 (first leaf upward to spine), hop 2 (spine downward to leaf), and hop 3 (leaf to host).

	Stride			Bijection			Shuffle		
	CONGA	Presto	DRILL	CONGA	Presto	DRILL	CONGA	Presto	DRILL
Elephant throughput	1.55	1.71	1.8	1.46	1.62	1.78	1	1.1	1.1
Mean FCT	0.51	0.41	0.21	0.71	0.63	0.45	0.95	0.91	0.86
99.99th percentile FCT	0.2	0.15	0.04	0.22	0.18	0.08	0.86	0.79	0.68

Table 1: Mean elephant flow throughput and mice FCT normalized to ECMP for the synthetic workloads.

which each server sends flows to all other servers in a random order. Similar to [42], we use 1GB "elephant" flows, and in addition we send 50 KB "mice flows" every 100 ms. We use a Clos with 4 leaf and 4 spine switches with each leaf connected to 8 hosts where all links have 1Gbps capacity. Table 1 reports the mean and 99.99**th** percentile of FCT for mice and mean flow throughput for elephants, all normalized by ECMP. For the Random and Stride workloads, DRILL significantly reduces mice latencies particularly in the tail and achieves higher throughout for the elephant flows. None of the tested schemes improve ECMP much for the shuffle workload since it is mainly bottlenecked at the last hop.

Hardware and deployability considerations. We implemented DRILL in Verilog in less than 400 lines of code. We estimate DRILL's area overhead using Xilinx Vivado Design Suite 2014.4 and the area estimation from [56, 58]. DRILL is estimated to require $0.04mm^2$ of chip area. Using the minimum chip area estimate of 200 mm^2 in [38], similar to [66], we estimate this to be less than 1% of the area of a typical switch chip. This demonstrates the feasibility and ease of implementing DRILL in hardware. DRILL involves two additional components. In the case of topological asymmetry, switches need to calculate the weights of traffic for each symmetric component; this can be done in control software local to the switch (if topology information is available via the routing algorithm) or through a central controller. Optionally, DRILL can employ a shim layer, deployed in a hypervisor as in [42]. As we have shown, this is not always necessary, and [42] showed its feasibility.

5 RELATED WORK

Recent works attribute the poor performance of ECMP to (a) its lack of global congestion information, or (b) hash collision when there are large flows. In the first group, Planck presents a fast network measurement architecture that enables rerouting congested flows in milliseconds [61]. Fastpass [59] posits that each sender should delegate control to a centralized arbiter to dictate when and via which path each packet should be transmitted. Hedera [18], MicroTE [25], and Mahout [32] re-route large flows to compensate for the inefficiency of ECMP hashing them onto the same path.

In the second category, Presto [42] argues that in a symmetric Clos where all flows are small, ECMP provides near optimal load balance, and therefore divides flows into "flowcells" which are source-routed so they are striped across all paths, without load-awareness. A centralized controller helps respond to failures. Other efforts in this category include balancing "flowlets" [19, 44] or per-packet spreading of traffic in a round robin fashion [27, 34]. We have compared with Presto in design and evaluation elsewhere in this paper.

CONGA [19] takes a hybrid approach by both splitting traffic into flowlets and using in-network congestion feedback mechanisms to estimate load and route flowlets. Our experiments indicate that DRILL's micro load balancing outperforms CONGA.

DRILL's queueing algorithm is inspired by "power of two choices" load balancing [53]. [54] and [63] study the impact of using memory of past choices. These models have *one* arbiter responsible for placing incoming tasks. Our multiple arbiters (forwarding engines) produce distinct behavior (Figure 3). This has led us to experimentally optimize parameter choice, but a theoretical analysis of our model may be valuable in the future.

Our earlier workshop paper introduced the micro load balancing concept [36]. We contribute new algorithms to handle failures, extensive simulations, and a Verilog switch implementation.

6 CONCLUSION

Contrary to the pervasive approach of load balancing based on macroscopic view of traffic, we explore *micro load balancing*: enabling the fabric to make decisions at *µsec* timescales based on traffic information local to each switch, solving challenges of this approach including hardware feasibility, packet reordering, and asymmetry. Our experiments show that our simple provably-stable switch scheduling algorithm, DRILL, outperforms state-of-the-art load balancers in Clos networks, particularly under high load and incast. DRILL adapts to asymmetry by decomposing the network into symmetric parts. Interesting avenues of future work include studying micro load balancing in other topologies, and the effect of delayed queue information in switches with multiple forwarding engines.

Acknowledgments: We would like to thank our shepherd, Theophilus Benson, and the reviewers for their feedback. This work was supported by a VMware Graduate Fellowship, by National Science Foundation CNS Award 1423452, and by NSERC Discovery Grant No. 346203-2012.

REFERENCES

[1] 2011. WSS Monitoring - Handling Microbursts. (2011). http://www.vssmonitoring.com/resources/feature-brief/Microburst.pdf.

[2] 2014. Introducing Data Center Fabric, the Next-generation Facebook Data Center Network. (2014). https://code.facebook.com/posts/360346274145943.

[3] 2014. Monitor Microbursts on Cisco Nexus 5600 Platform and Cisco Nexus 6000 Series Switches. (2014). http://www.cisco.com/c/en/us/products/collateral/switches/nexus-5000-series-switches/white-paper-c11-733020.pdf.

[4] 2015. ONS 2015 Keynote: A. Vahdat, Google. (2015). www.youtube.com/watch?v=FaAZAII2x0w.

[5] 2016. 6800 Series 10 Gigabit and Gigabit Ethernet Interface Modules for Cisco 6500 Series Switches Data Sheet. (2016). http://www.cisco.com/c/en/us/products/collateral/switches/catalyst-6500-series-switches/data_sheet_c78-451794.html.

[6] 2016. Cisco Catalyst 4500 Series Line Cards Data Sheet. (2016). http://www.cisco.com/c/en/us/products/collateral/interfaces-modules/catalyst-4500-series-line-cards/product_data_sheet0900aecd802109ea.html.

[7] 2016. High-density, Highly Available Aggregation and Intelligent Distributed Network Services at the Edge for Service Providers and Enterprises. (2016). http://www.cisco.com/c/en/us/products/collateral/routers/7500-series-routers/product_data_sheet0900aecd800f5542.html.

[8] 2016. Private discussions with a major switch vendor. (2016).

[9] 2016. Understanding MX Fabric. (2016). http://kb.juniper.net/InfoCenter/index?page=content&id=KB23065&actp=search.

[10] 2017. Arista Visibility. (2017). https://www.arista.com/en/products/eos/visibility.

[11] 2017. INET Framework. (2017). https://inet.omnetpp.org/.

[12] 2017. LANZ - A New Dimension in Network Visibility. (2017). https://www.arista.com/assets/data/pdf/TechBulletins/Lanz.pdf.

[13] 2017. Microburst Monitoring. (2017). http://www.cisco.com/c/en/us/td/docs/switches/datacenter/nexus6000/sw/qos/7x/b_6k_QoS_Config_7x/micro_burst___monitoring.pdf.

[14] 2017. Monitor Microbursts on Cisco Nexus 5600 Platform and Cisco Nexus 6000 Series Switches. (2017). http://www.cisco.com/c/en/us/products/collateral/switches/nexus-5000-series-switches/white-paper-c11-733020.html.

[15] 2017. Network Simulation Cradle Integration. (2017). https://www.nsnam.org/wiki/Network_Simulation_Cradle_Integration.

[16] 2017. OMNeT++ Discrete Event Simulator. (2017). https://omnetpp.org/.

[17] Mohammad Al-Fares, Alexander Loukissas, and Amin Vahdat. 2008. A Scalable, Commodity Data Center Network Architecture. In *SIGCOMM*.

[18] Mohammad Al-Fares, Sivasankar Radhakrishnan, Barath Raghavan, Nelson Huang, and Amin Vahdat. 2010. Hedera: Dynamic Flow Scheduling for Data Center Networks.. In *NSDI*.

[19] Mohammad Alizadeh, Tom Edsall, Sarang Dharmapurikar, Ramanan Vaidyanathan, Kevin Chu, Andy Fingerhut, Francis Matus, Rong Pan, Navindra Yadav, George Varghese, and others. 2014. CONGA: Distributed Congestion-aware Load Balancing for Datacenters. In *SIGCOMM*.

[20] Mohammad Alizadeh, Albert Greenberg, David A Maltz, Jitendra Padhye, Parveen Patel, Balaji Prabhakar, Sudipta Sengupta, and Murari Sridharan. 2011. Data Center TCP (DCTCP). *CCR* 41, 4 (2011).

[21] Mark Allman, Vern Paxson, and William Stevens. 1999. RFC 2581: TCP congestion control. (1999).

[22] Yossi Azar, Andrei Z Broder, Anna R Karlin, and Eli Upfal. 1999. Balanced allocations. *SIAM journal on computing* 29, 1 (1999).

[23] Theophilus Benson, Aditya Akella, and David A Maltz. 2010. Network Traffic Characteristics of Data Centers in the Wild. In *IMC*.

[24] Theophilus Benson, Ashok Anand, Aditya Akella, and Ming Zhang. 2010. Understanding Data Center Traffic Characteristics. *CCR* (2010).

[25] Theophilus Benson, Ashok Anand, Aditya Akella, and Ming Zhang. 2011. MicroTE: Fine Grained Traffic Engineering for Data Centers. In *CoNEXT*.

[26] Vijay Bollapragada, Curtis Murphy, and Russ White. 2000. *Inside Cisco IOS software architecture*. Cisco Press.

[27] Jiaxin Cao, Rui Xia, Pengkun Yang, Chuanxiong Guo, Guohan Lu, Lihua Yuan, Yixin Zheng, Haitao Wu, Yongqiang Xiong, and Dave Maltz. 2013. Per-packet Load-balanced, Low-latency Routing for Clos-based Data Center Networks. In *CoNEXT*. ACM.

[28] Martin Casado, Teemu Koponen, Scott Shenker, and Amin Tootoonchian. 2012. Fabric: A Retrospective on Evolving SDN. In *HotSDN*.

[29] Kai Chen, Hongyun Zheng, Yongxiang Zhao, and Yuchun Guo. 2012. Improved Solution to TCP Incast Problem in Data Center Networks. In *CyberC*.

[30] Yanpei Chen, Rean Griffith, Junda Liu, Randy H. Katz, and Anthony D. Joseph. 2009. Understanding TCP Incast Throughput Collapse in Datacenter Networks. In *WREN*.

[31] K. Chudgar and S. Sathe. 2014. Packet Forwarding System and Method Using Patricia Trie Configured Hardware. (2014). http://www.google.com/patents/US8767757 US Patent 8,767,757.

[32] Andrew R Curtis, Wonho Kim, and Praveen Yalagandula. 2011. Mahout: Low-overhead Datacenter Traffic Management Using End-host-based Elephant Detection. In *INFOCOM*.

[33] Prajjwal Devkota and AL Narasimha Reddy. 2010. Performance of Quantized Congestion Notification in TCP Incast Scenarios of Data Centers. In *International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems*.

[34] Abhishek Dixit, Pawan Prakash, Yu Charlie Hu, and Ramana Rao Kompella. 2013. On the Impact of Packet Spraying in Data Center Networks. In *INFOCOM*.

[35] Yilong Geng, Vimalkumar Jeyakumar, Abdul Kabbani, and Mohammad Alizadeh. 2016. Juggler: A Practical Reordering Resilient Network Stack for Datacenters. In *EuroSys*.

[36] Soudeh Ghorbani, Brighten Godfrey, Yashar Ganjali, and Amin Firoozshahian. 2015. Micro Load Balancing in Data Centers with DRILL. In *HotNets*.

[37] Soudeh Ghorbani, Zibin Yang, Brighten Godfrey, Yashar Ganjali, and Amin Firoozshahian. 2017. *DRILL: Micro Load Balancing for Clos Data Center Networks*. Technical Report. University of Illinois at Urbana-Champaign. http://ghorban2.web.engr.illinois.edu/papers/drill.

[38] Glen Gibb, George Varghese, Mark Horowitz, and Nick McKeown. 2013. Design Principles for Packet Parsers. In *ANCS*.

[39] Phillipa Gill, Navendu Jain, and Nachiappan Nagappan. 2011. Understanding Network Failures in Data Centers: Measurement, Analysis, and Implications. In *CCR*.

[40] Ramesh Govindan, Ina Minei, Mahesh Kallahalla, Bikash Koley, and Amin Vahdat. 2016. Evolve or Die: High-Availability Design Principles Drawn from Googles Network Infrastructure. In *SIGCOMM*.

[41] Albert Greenberg, James R. Hamilton, Navendu Jain, Srikanth Kandula, Changhoon Kim, Parantap Lahiri, David A. Maltz, Parveen Patel, and Sudipta Sengupta. 2011. VL2: A Scalable and Flexible Data Center Network. *SIGCOMM* (2011).

[42] Keqiang He, Eric Rozner, Kanak Agarwal, Wes Felter, John Carter, and Aditya Akella. 2015. Presto: Edge-based Load Balancing for Fast Datacenter Networks. In *SIGCOMM*.

[43] Jaehyun Hwang, Joon Yoo, and Nakjung Choi. 2012. IA-TCP: A Rate Based Incast-avoidance Algorithm for TCP in Data Center Networks. In *ICC*.

[44] Srikanth Kandula, Dina Katabi, Shantanu Sinha, and Arthur Berger. 2007. Dynamic Load Balancing without Packet Reordering. *CCR* (2007).

[45] Srikanth Kandula, Sudipta Sengupta, Albert Greenberg, Parveen Patel, and Ronnie Chaiken. 2009. The Nature of Data Center Traffic: Measurements and Analysis. In *IMC*.

[46] Nanxi Kang, Monia Ghobadi, John Reumann, Alexander Shraer, and Jennifer Rexford. 2015. Efficient Traffic Splitting on Commodity Switches. In *CoNEXT*.

[47] P.R. Kumar and S.P. Meyn. 1993. Stability of queueing networks and scheduling policies. In *Decision and Control*.

[48] Petr Lapukhov and Ariff Premji. 2016. RFC 7938: Use of BGP for Routing in Large-Scale Data Centers. (2016).

[49] C. Leiserson. 1985. Fat-trees: Universal Networks for Hardware Efficient Supercomputing. *IEEE Transactions on Computer* C-34, 10 (1985).

[50] Xiaozhou Li and Michael J. Freedman. 2013. Scaling IP Multicast on Datacenter Topologies. *CoNEXT* (2013).

[51] Vincent Liu, Daniel Halperin, Arvind Krishnamurthy, and Thomas Anderson. 2013. F10: A Fault-tolerant Engineered Network. In *NSDI*.

[52] Adisak Mekkittikul and Nick McKeown. 1998. A Practical Scheduling Algorithm to Achieve 100% Throughput in Input-Queued Switches. In *INFOCOM*.

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

[54] Michael Mitzenmacher, Balaji Prabhakar, and Devavrat Shah. 2002. Load Balancing with Memory. In *FOCS*.

[55] David Nagle, Denis Serenyi, and Abbie Matthews. 2004. The Panasas Activescale Storage Cluster: Delivering Scalable High Bandwidth Storage. In *SC*.

[56] Jad Naous, David Erickson, G Adam Covington, Guido Appenzeller, and Nick McKeown. 2008. Implementing an OpenFlow Switch on the NetFPGA Platform. In *ANCS*.

[57] Radhika Niranjan Mysore, Andreas Pamboris, Nathan Farrington, Nelson Huang, Pardis Miri, Sivasankar Radhakrishnan, Vikram Subramanya, and Amin Vahdat. 2009. PortLand: A Scalable Fault-tolerant Layer 2 Data Center Network Fabric. *CCR* (2009).

[58] Ketan Padalia, Ryan Fung, Mark Bourgeault, Aaron Egier, and Jonathan Rose. 2003. Automatic Transistor and Physical Design of FPGA Tiles from an Architectural Specification. In *ACM/SIGDA eleventh international symposium on Field Programmable Gate Arrays*.

[59] Jonathan Perry, Amy Ousterhout, Hari Balakrishnan, Devavrat Shah, and Hans Fugal. 2014. Fastpass: A Centralized Zero-queue Datacenter Network. In *SIGCOMM*.

[60] Amar Phanishayee, Elie Krevat, Vijay Vasudevan, David G Andersen, Gregory R Ganger, Garth A Gibson, and Srinivasan Seshan. 2008. Measurement and Analysis of TCP Throughput Collapse in Cluster-based Storage Systems. In *FAST*.

[61] Jeff Rasley, Brent Stephens, Colin Dixon, Eric Rozner, Wes Felter, Kanak Agarwal, John Carter, and Rodrigo Fonseca. 2014. Planck: Millisecond-scale Monitoring and Control for Commodity Networks. In *SIGCOMM*.

[62] Arjun Roy, Hongyi Zeng, Jasmeet Bagga, George Porter, and Alex C Snoeren. 2015. Inside the Social Network's (Datacenter) Network. In *SIGCOMM*.

[63] Devavrat Shah and Balaji Prabhakar. 2002. The use of memory in randomized load balancing. In *ISIT*.

[64] Arjun Singh, Joon Ong, Amit Agarwal, Glen Anderson, Ashby Armistead, Roy Bannon, Seb Boving, Gaurav Desai, Bob Felderman, Paulie Germano, and others. 2015. Jupiter Rising: A Decade of Clos Topologies and Centralized Control in Google's Datacenter Network. In *SIGCOMM*.

[65] Ankit Singla, Chi-Yao Hong, Lucian Popa, and P. Brighten Godfrey. 2012. Jellyfish: Networking Data Centers Randomly. In *NSDI*.

[66] Anirudh Sivaraman, Suvinay Subramanian, Sharad Alizadeh, Mohammad Alizadehand Chole, Shang-Tse Chuang, Anurag Agrawal, Hari Balakrishnan, Tom Edsall, Sachin Katti Katti, and McKeown Nick. 2016. Programmable Packet Scheduling at Line Rate. In *SIGCOMM*.

[67] Thomas M Thomas and Doris E Pavlichek. 2003. *Juniper Networks reference guide: JUNOS routing, configuration, and architecture*.

[68] Leslie G. Valiant. 1982. A Scheme for Fast Parallel Communication. *SIAM journal on computing* 11, 2 (1982).

[69] Vijay Vasudevan, Amar Phanishayee, Hiral Shah, Elie Krevat, David G. Andersen, Gregory R. Ganger, Garth A. Gibson, and Brian Mueller. 2009. Safe and Effective Fine-grained TCP Retransmissions for Datacenter Communication. In *SIGCOMM*.

[70] Haitao Wu, Zhenqian Feng, Chuanxiong Guo, and Yongguang Zhang. 2013. ICTCP: Incast Congestion Control for TCP in Data-center Networks. *IEEE/ACM Transactions on Networking (TON)* 21, 2 (2013).

[71] Jiao Zhang, Fengyuan Ren, and Chuang Lin. 2011. Modeling and Understanding TCP Incast in Data Center Networks. In *INFOCOM,*.

[72] Yan Zhang and Nirwan Ansari. 2011. On Mitigating TCP Incast in Data Center Networks. In *INFOCOM,*.

[73] Junlan Zhou, Malveeka Tewari, Min Zhu, Abdul Kabbani, Leon Poutievski, Arjun Singh, and Amin Vahdat. 2014. WCMP: Weighted Cost Multipathing for Improved Fairness in Data Centers. In *EuroSys*.