MICRO LOAD BALANCING WITH DELAYED QUEUE LENGTHS

BY

FATIMA TARIQ

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Computer Science
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2019

Urbana, Illinois

Adviser:

Associate Professor Philip Brighten Godfrey

ABSTRACT

DRILL is a micro load balancing algorithm designed to efficiently utilize the path redundancy in modern data centers. It uses egress port queue lengths to make fast packet routing decisions to reduce upstream congestion and queueing delays. However, high performance switches with multiple forwarding engines making routing decisions in parallel, do not have direct access to these queue lengths. We explore and evaluate different ways of obtaining this information in data center settings, specifically using incoming traffic and specially generated update packets to piggyback this information. We find that staleness of this data does not have a huge impact on flow completion times compared to DRILL (6% increase) and still achieves a considerable advantage over ECMP (28% decrease).

To my parents, for their love and support.

TABLE OF CONTENTS

CHAPTER 1: INTRODUCTION

Modern data centers have multi-tiered Clos topologies [1–8]. This results in multiple paths between a pair of source and destination servers. ECMP, a widely used protocol in data centers [7, 8], assigns pseudo-random paths to flows. However, data center traffic is composed of variably sized flows [3] so a random distribution leads to hash collisions giving rise to congestion along some paths and under utilization of the others [8–11]. Furthermore, data centers are also prone to link failures which result in an asymmetric topology [12] that aggravates the load imbalance of ECMP [13]. Load balancing schemes should, thus, be aware of network state and resilient to congestion and link failures.

Several load balancing schemes have been proposed in the past, each with their own set of shortcomings. Centralized schemes (e.g. Hedera [11] and FastPass [14]) incur the overhead of communication with controller while end hosts based protocols (e.g. FlowBender [8] and Presto [9]) have a limited view of the network. CONGA [13] is based at switches and needs a round trip time to estimate congestion. Most of these schemes balance load at a high granularity (flows [11], flowcells [9] or flowlets [13]) and have slow control loops and so, are unable to tackle microbursts. This motivates the need for a protocol placed at the switch level making decisions using local information in order to detect and react to microbursts in a timely manner.

DRILL [7] is a load balancing protocol that leverages "power of two choices" [15] to make a local decision at switch level. The forwarding engine checks the queue lengths of two randomly selected links and the most recently used link from the set of possible next hops and forwards the packet on the link with the least loaded queue. DRILL is proven to outperform the state of the art schemes such as CONGA [13] and Presto [9].

However, in high performance switches running several forwarding engines in parallel, forwarding engines do not have direct memory access to outgoing queue lengths [7]. We study different approaches of obtaining this information and their effect on mean flow completion times.

Switches can generate small query packets that travel to the egress ports and bring the queue lengths to the switch. We can increase the interval between these queries to reduce the overhead. Our evaluations explore the effect of increasing this interval on the mean flow completion times and find that at an interval of 10 s, the mean flow completion times are within 22% of that of DRILL even at high loads.

We also explore the use of incoming data packets to piggyback the outgoing port queue lengths. The rate of arrival of this information increases as congestion increases so DRILL algorithm is able to make more accurate decisions to deal with microbursts. We find that the mean flow completion time using this method is within 11% of that of DRILL even at high loads.

Finally, we combine the above two approaches of obtaining queue lengths and find that when using incoming packets as well as update packets at an interval of 10 s to obtain queue length information, there is only a 6% increase in mean flow completion time compared to DRILL even at high loads. So, we can conclude that the adaptations required by DRILL for hardware implementation do not have a huge impact on its performance and still offer a significant advantage over ECMP (28% reduction in mean flow completion times).

CHAPTER 2: BACKGROUND

2.1 DATA CENTER TOPOLOGIES

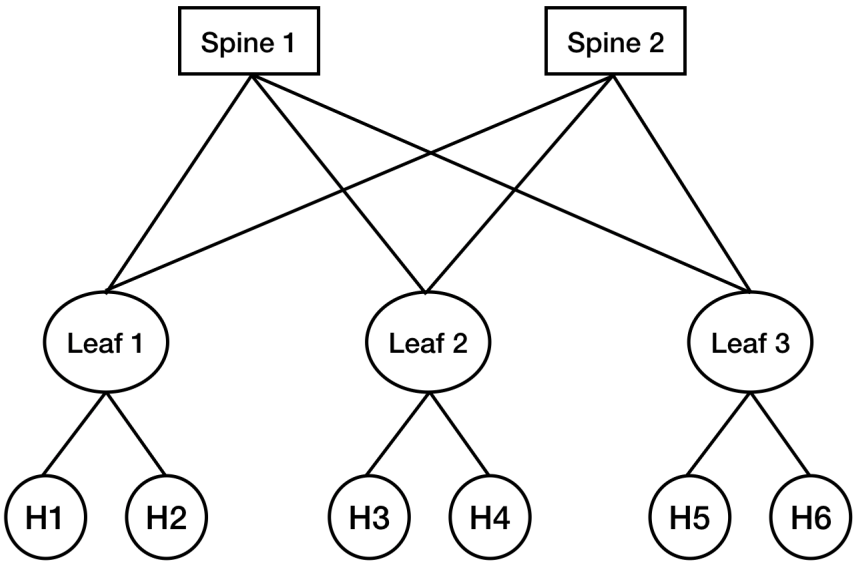

Figure 2.1: A small Leaf Spine topology

Data centers networks are made up of multi-tiered topologies [1–6] with servers being the lowest layer of the network and connected to each other by two or more layers of switches higher up in the hierarchy. A basic 2-tiered Clos topology as illustrated in Figure 2.1 is also known as the Leaf Spine topology. Modern data centers [2, 3] and recent research works [5, 6] feature data center topologies that are a superposition of different variations of Leaf Spine. These topologies are scalable in nature so more servers and switches can be added to the network as required. This allows data centers to use cheap commodity switches and lower bandwidth links in bulk to connect a large number of servers [5, 6]. Furthermore, the redundancy in the network also makes it more resilient to failures compared to a network made up of a few high performance switches and links [6]. However, this also leads to more challenging routing and load balancing problems.

2.2 DATA CENTER WORKLOADS

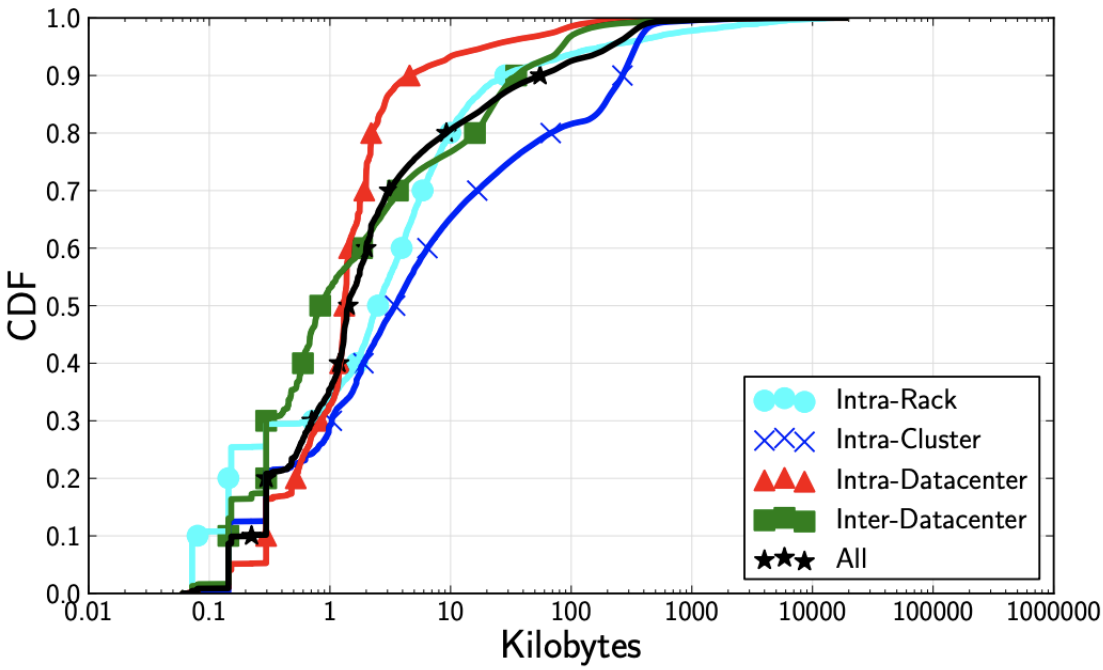

Figure 2.2: Distribution of a Web Search Workload [3]

Data center workloads are heavily skewed in a way that there is an extremely high number of small flows called mice flows and a very few large sized flows called elephant flows. The above graph (Figure 2.2) shows the CDF of a typical web search workload in a Facebook data center [3]. In the intra-data center traffic, more than 90% of the flows are less than 10KB in size. Less than 1% of the flows are longer than 1MB in size [3].

2.3 CONGESTION AND FAILURES

With such a variation in flow sizes, inter arrival times and path diversity [3], data centers are prone to congestion. These congestion events cause an unusual rise of traffic on particular links, resulting in queue buildup and eventual packet loss that is very detrimental to flow completion times. Data center traffic is highly bursty in nature [3,7,13] which gives rise to short lived congestion events, called microbursts, that last for less than a millisecond [7].

This requires data center transport protocols to be able to detect and react to congestion very quickly.

Data centers are also prone to frequent link failures [12, 13] that give rise to asymmetric topologies [7, 8, 13]. When a link fails, the packets at the last hop before failed link are dropped until a routing protocol reruns and eliminates the path with the failed link from the routing tables. A link failure can result in a median of 59,000 packet drops [12]. When the routing tables converge, the traffic is rerouted on to the remaining links [16].

2.4 LOAD BALANCING SCHEMES

In data center multipath topologies, for every pair of servers connected to different leafs, there are multiple equal cost paths a packet can take. For example, in Figure 2.1, a packet from H1 to H6 can travel through Spine 1 or Spine 2 with four hops each. From a spine switch, there is only one downlink path towards the destination server but the leaf switches have multiple uplink paths on the shortest path. Multipath routing algorithms add multiple next hops next to a destination prefix at leaf switches, so the load balancing decision needs to be made at the leaf switches.

Given the variation in flow sizes in a data center, the ideal way to distribute traffic is to iterate over outgoing links on a shortest path to the destination and send one incoming packet on each path in a Round Robin (RR) fashion. However, this approach requires switches to store state. A stateless load balancing method is to route each packet on a random outgoing link in a uniform distribution, also known as Random Packet Spraying (RPS) [17]. However, both these methods operate at packet level granularity and result in significant packet reordering.

To reduce packet reordering, load balancing schemes often route complete flows on one path. The most widely used multipath protocol [7, 8], ECMP (Equal Cost Multi Path) assigns a random path to a flow based on a hash computed using IP and port addresses of the source and destination. However, hash collisions of two elephant flows leads to an asymmetric distribution of traffic resulting in congestion on a few paths and underutilization of the others. This results in only 40-80% of the network capacity being utilized [10]. ECMP performs quite poorly in data centers as observed by many pieces of work [8–11].

2.5 INTERACTION WITH TRANSPORT PROTOCOLS

TCP uses round trip time estimation to adjust its congestion window. When the packets of a flow take a fixed path, the RTT estimate is accurate. However if we route packets independently in multipath topologies, packets of a flow can take multiple paths, often with an uneven level of congestion. So, packets of the same flow reach the destination in different order. Apart from the significant packet reordering, the more congested links also result in an increase in the round trip time estimate, resulting in a reduced sending rate at the source and consequently, an underutilization of less congested paths [7]. Load balancing protocols that route complete flows on one path e.g. ECMP avoid the reordering and overestimation of round trip time problem but do not perform well because link utilization is skewed. Load balancing schemes should keep variance in path utilization small so there is minimal reordering and similar round trip times across all paths.

CHAPTER 3: RELATED WORK

Several load balancing schemes have been proposed in the past that have tried to incorporate congestion awareness into load balancing with fundamental differences in their approach. Some recent works propose a centralized approach (e.g. Hedera [11] and FastPass [14]), while others shift the functionality to end-hosts (e.g. FlowBender [8] and Presto [9]) or switches (e.g. CONGA [13] and pFabric [18]).

Hedera [11], for instance, proposes a centralized controller that computes non-conflicting paths for flows in the network and provides routing information to switches. While this scheme is able to calculate the most optimal paths to distribute traffic, the overhead of communication with the controller makes it slow to react to congestion [9, 13]. This delay is very detrimental to the time sensitive short flows.

FastPass [14] is another centralized load balancing schemes that balances load at a packet level granularity as compared to the flow level granularity of Hedera [11]. This centralized arbiter determines the time to schedule and the path to take for each packet in the network. Although, their scheme consists of fast algorithms, the overhead is high along with significant chances for reordering.

FlowBender [8] adds congestion awareness to ECMP by including TTL field in the hash function. The sender increments the TTL of the subsequent packets of a flow if a received packet has the ECN bit set. Since, a set ECN bit reflects congestion in the path assigned to the flow by ECMP, changing TTL maps the flow to a different path. However since ECN information is received in a round trip time, this protocol is too slow to react to microbursts and failures.

Presto [9] leverages the property of ECMP that its load balance is closer to optimal when traffic is only composed of short sized flows. Since congestion usually results from collision of large sized flows, Presto makes elephant flows mimic mice flows by dividing flows into flow cells of 64KB and iterating these flow cells over paths in a round robin fashion. However, the possibility of synchronization of two end hosts results in periodic high load on the same links, similar to the behavior of microbursts. Secondly, as the end hosts spray flowcells unaware of congestion in network, the protocol relies on vSwitches to detect failures and prune failed paths from routing tables. Thus it reacts slowly to asymmetry in topology and cannot deal with microbursts in a timely manner thus affecting flow completion times of short flows.

CONGA [13] maintains a global view of the network. It stores a Congestion Table at each leaf switch for the capacity of each uplink to every other leaf switch. The congestion is estimated by taking the maximum along the path and using the response packets to piggyback this information to the leaf switch. The theoretical analysis of CONGA is provided using the Price of Anarchy which is small for 2-tier Leaf Spine topologies. However, data centers (including those of Google [2] and Facebook [3]) frequently use 3-tier topologies and the Price of Anarchy is higher for these topologies.

pFabric [18] rethinks the data center fabric and uses the principle of Shortest Remaining Processing Time First to minimize flow completion times. If an egress port is free, the protocol schedules the earliest packet of the flow with the least number of packets remaining. While this algorithm results in very small FCTs for mice flows, elephant flows are aggressively choked as any new mice flow is prioritized over them. The protocol also requires significant changes to the rate control mechanisms at end hosts and scheduling mechanisms at switches, especially as the packets are enqueued within the switch and not at the egress port queues.

MPTCP [16] takes a different approach as it focuses on making the transport protocol adapt to multipath topologies. It creates dynamic subflows that control their own congestion windows to deal with the problems described in the above Section 2.5. However, as other works [8, 9, 13] point out MPTCP adds a significant complexity at end host level and deploying it in a data center environment is made even more difficult considering the fact that data centers allow vendors to run their own Virtual Machines at servers. These machines run their own protocols often bypassing the kernel and so transport protocols cannot be enforced.

CHAPTER 4: DESIGN

4.1 DESIGN REQUIREMENTS

1. **Congestion Awareness and Responsiveness:** A load balancing scheme should be aware of congestion in the network. Proactive load balancing schemes such as ECMP and Presto [9] inherently try to balance load distribution to avoid congestion buildup. However, this congestion unawareness renders them unable to respond to congestion. There are several cues that can indicate the buildup of congestion such as high queue lengths, packet drops, set ECN bits, etc. and are utilized by recent work for congestion reactive protocols [7–9, 13].

2. **Failure Resilience:** A load balancing scheme should be able to detect and react quickly to both partial and complete link failures.

3. **Minimal Changes to Existing Infrastructure:** While centralized load balancing schemes [11, 14] understandably require significant changes to the network, even the load balancing schemes for which most of the functionality resides at the switches [8, 13, 18] need changes to the end hosts. Ideally, the protocol should require minimal changes to the network hardware, as well as the transport protocols. Otherwise, the huge overhead of implementation makes it unfavorable.

4. **Adaptable to a variety of Data Center Topologies and Workloads:** A load balancing scheme should be able to adapt to a variety of topologies, including multi-tier and asymmetric. Schemes specialized for simple topologies such as CONGA [13] are difficult to adapt to multi-tier topologies [1–6] while round robin or uniform random schemes including ECMP assume symmetric nature of topologies and perform poorly on asymmetries resulting from link failures or inherent nature of topology design (such as Jellyfish [19] and BCube [20]).

4.2 DRILL

DRILL [7] is a micro load balancing protocol that makes per packet routing decisions at switch level using egress port queue lengths. Load balancing at this fine granularity ensures a more balanced distribution of load across available paths and additionally, the readily available nature of congestion information (estimated by the magnitude of outgoing queue lengths) leads to better handling of microbursts. DRILL leverages "power of two choices" [15] to make routing decisions. The forwarding engine at leaf switch checks the outgoing queue lengths of two upstream links picked at random and the link used to forward the last upstream packet and forwards the packet on the least loaded of these three queues.

DRILL outperforms state-of-the-art load balancing schemes, such as CONGA [13] and Presto [9], and the improvement becomes increasingly pronounced under high loads. DRILL is also failure resilient, scalable and adaptive to asymmetric topologies and various workloads.

4.3 SWITCH HARDWARE

Switches have forwarding engines that process packets including routing table lookups and path selection. Data centers use high performance switches which usually have multiple forwarding engines processing packets in parallel [7, 21]. These switches make local, independent forwarding decisions and are not aware of queue lengths at egress ports. DRILL requires queue length information of (two random and one memory) egress ports. Forwarding engines can exchange egress port queue lengths by generating packets. While high performance switches have a very low latency, as low as 550ns - 2usec for Arista 7300 series [22] for example, waiting for updated queue length for every packet can increase processing time substantially.

Some switches also have access to queue length information to detect microbursts [7,23,24]

but this information is subject to random delays and imprecisions [7, 24]. As these switches can face random delays in arrival of queue length information, by establishing a range of the maximum and minimum delays, we can estimate the expected performance by looking at the evaluation in Section 5.2 that studies the effect of delayed updates in queue length on the flow completion times in a typical data center environment.

4.4 GETTING QUEUE LENGTH INFORMATION

We can spread egress port queue lengths by piggybacking this information onto packets. A potential way to avoid waiting for all queue lengths for every packet decision is for the forwarding engine to maintain a local state of the outgoing queue lengths and update it for every information arrival. We explore two ways to obtain this information in this study, by generating update packets or by utilizing incoming packets of the data center traffic.

4.4.1 Getting information at periodic intervals

As forwarding engines do not have access to all egress port queue lengths, they can generate packets to distribute the easily accessible queue lengths to other forwarding engines within the switch. It is not possible to wait for all egress port queue length updates for every packet decision at the forwarding engine because this incurs a large overhead in packet processing time, so we use the locally maintained state for runtime packet decisions. We can establish an interval at the switch with which we periodically exchange these queue lengths. Each forwarding engine maintains a timer and after a specific interval, generates packets to distribute the available queue lengths and on every such packet arrival, update its locally maintained state. A large interval reduces the overhead of update packet generation but effectively increases the load balancing granularity; as multiple packets are routed using the same information, if the interval is large enough, we approach flowlet level granularity. A large interval also gives rise to a very bursty traffic behavior. There is a high probability

of DRILL scheduling flows on the path with the shortest egress port queue. If this queue length is not updated for the next several packet decisions, we experience traffic bursts on a selective link per interval. In our evaluations (Section 5.2), we study the delay that the we can tolerate without experiencing a significant degradation in performance compared to DRILL.

4.4.2 Getting information from incoming packets

We can update the outgoing queue length of the incoming link the packet last used because the forwarding engine processing this packet has easy access to it. This only results in queue length update for one outgoing queue per incoming packet. Under smaller loads, we have less traffic in the network and thus, we get slower updates to the maintained queue state but as load increases, the inter-arrival time of packets decreases and so the rate of updates increases. Therefore, this system is able to deal with congestion scenarios as the queue length information is more updated and the forwarding engines can make more accurate and informed decisions under high load. In our evaluation (Section 5.3), we compare the performance of this system with DRILL, ECMP and the approach describes in the previous subsection.

However, this might give rise to a scenario where uplink utilization is skewed. Consider the arbitrary network state snapshot shown in Figure 4.1. Assume the network has a moderate load and the queue lengths are a result of a microburst. The last packets that traverse through Spine switch 1 in both ways might update the queue lengths of the uplink to Spine switch 1 at both Leaf switch A and B with higher values compared to other uplink queues as shown. Both Leaf switch A and B route new packets to Spine switches 2, 3 and 4 and these queue lengths keep getting updated by the bidirectional flow of traffic. However, both Leaf switches avoid sending packets to Spine switch 1 because even as these queues get drained with time, according to their stale information the queue length is high and there is no

packet flow to update this information. Thus this path becomes extremely starved. This situation is reversed only when congestion rises again to a point where any of the queue lengths for the other three links exceed this stale queue length. For example, if the queue lengths at Leaf switch A of uplinks to Spine switches 3 and 4 become 13 each, Leaf switch A can possibly route packets to Spine switch 1 again thus updating the new drained queue length at Leaf switch B. Leaf switch B is now very likely to send a packet to this underutilized link, updating queue lengths at Leaf Switch A and the deadlock is broken. As the number of Leaf switches increase, the possibility of getting path starvation decreases. However, latency sensitive small flows in data centers are affected by even small imprecisions in load balancing resulting that might result in longer queueing delays.

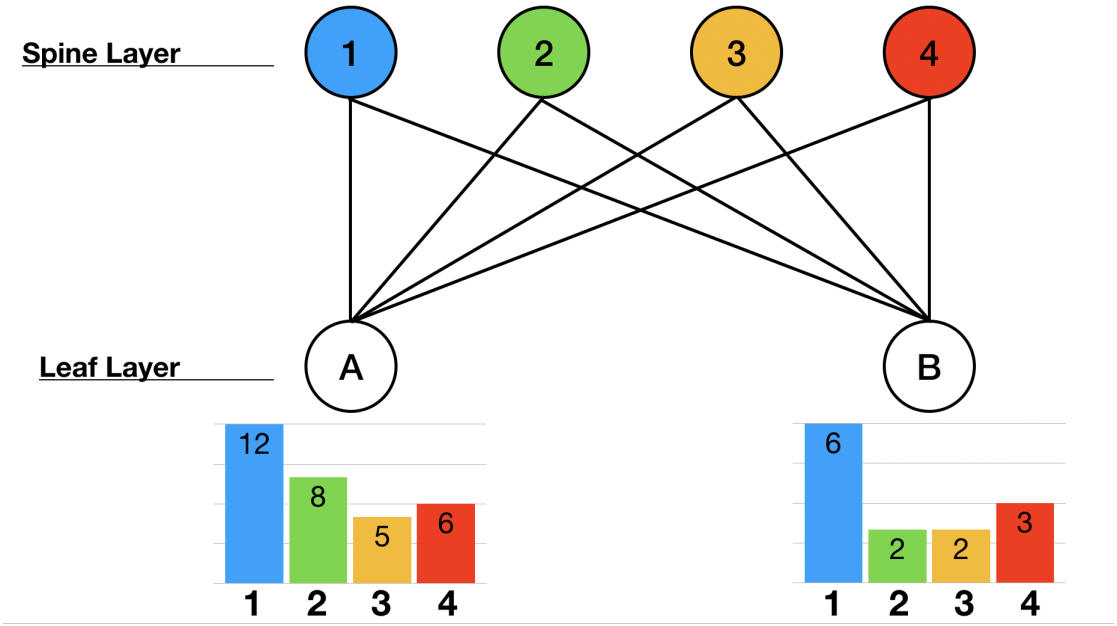

Figure 4.1: An arbitrary network snapshot of a small data center topology displaying the queue lengths of the outgoing links to all the spine switches at the leaf switches

4.4.3 Getting information from a combination of incoming packets and periodically generated packets

To proactively avoid the situation described above, we construct a hybrid approach combining the two methods. We piggyback the egress port queue length on every incoming packet to get maximum possible updates from the network. Additionally, we add a timer that sends periodic updates for all queue lengths to ensure that we do not work with stale data beyond a threshold. This also helps break any starvation situations early on. In Section 5.4, we evaluate the performance of this system using DRILL and ECMP as our benchmarks. We also compare our hybrid approach with the two individual approaches described above to determine the improvement achieved by combining them.

CHAPTER 5: EVALUATION

5.1 METHODOLOGY

For our evaluations, we use a leaf spine topology made of 4 spine and 8 leaf switches with 10 hosts attached to each leaf. The links connecting spine and leaf switches are 40 Gbps and the links connecting hosts to leaf switches are 10 Gbps, resulting in an oversubscribed topology of factor 1.25x. We use real data center trace-driven web search workloads [3] to generate traffic and run our simulations for 30 seconds. Since flow completion time (FCT) is an extremely important metric to gauge performance, we use mean FCT for comparison. We use DRILL and ECMP as benchmarks for our evaluation.

Since the codebase used to evaluate DRILL in [7] is not publicly available, we implement our own version of the algorithm as described in the paper [7] in a network simulator, ns2 [25]. We also implement our adaptations to DRILL in ns2 and release this as open source [26]. The multipath classifier at every switch maintains queue lengths for outgoing links in a data structure and the index of the last used link (memory link). When making a routing decision of every packet, we pick two unique random uplinks (excluding the memory link) and get the corresponding queue lengths from this data structure. For our evaluation, we update this data structure either periodically at controlled intervals (Section 5.2) or for every packet arrival (only for the queue length of the link the incoming packet used) (Section 5.3) or a combination of both (Section 5.4). However, the mean FCT's we obtain for DRILL are different from those mentioned in the paper, possibly because of different TCP parameters (e.g. initial congestion window, maximum congestion window, minimum RTO, etc.). We obtain a mean FCT of 3ms at a 10% load compared to 1ms as mentioned in the paper, for both DRILL and ECMP. At a 90% load, our mean FCT is 39ms for DRILL and 58ms for ECMP (1.5x ratio), compared to 5.5ms for DRILL and 14ms for ECMP (2.5x ratio) as

mentioned in the paper.

5.2 EFFECT OF UPDATES AT FIXED INTERVALS ON PERFORMANCE

In this section, we evaluate the performance of DRILL when generating packets periodically to spread queue length information. We keep the interval between these updates constant for simplicity. We study the effect of increasing this interval (Per Interval Updates) on performance. Ideally, we want a large interval in queries while keeping the difference in performance minimal so the overall overhead of updates is reduced. Our hypothesis is that as this interval increases, the load balance worsens because the decisions are made using stale information. A switch could route multiple packets onto a single link between updates if this link had a small queue length from the previous update thus temporarily increasing congestion on that link. To get a quantitative idea of the change in performance under delayed queue length information, we run two different sets of experiments; one varies delays under fixed loads and the other varies load under fixed delays.

5.2.1 Varying Delays at Fixed Loads

We run simulations under a moderate load (40%) and high load (80%) varying the delay from 100 nanoseconds to 100 microseconds with small intervals. We do not go below 100 nanoseconds because this is less than the transmission delay of a packet even for high performance links (150 Gbps) so it effectively reflects per packet queue length updates and also, internal latencies of high performance links do not go any lower (350 nanoseconds for Arista 7150 switches [27]).

At a load of 40% (Figure 5.1), as we increase update interval to 1 microsecond, there is not a huge difference in performance as compared to DRILL i.e. there is less than 1.5% increase in mean FCT. As we increase the delay from 1 s to 100 s, there is a drastic increase in

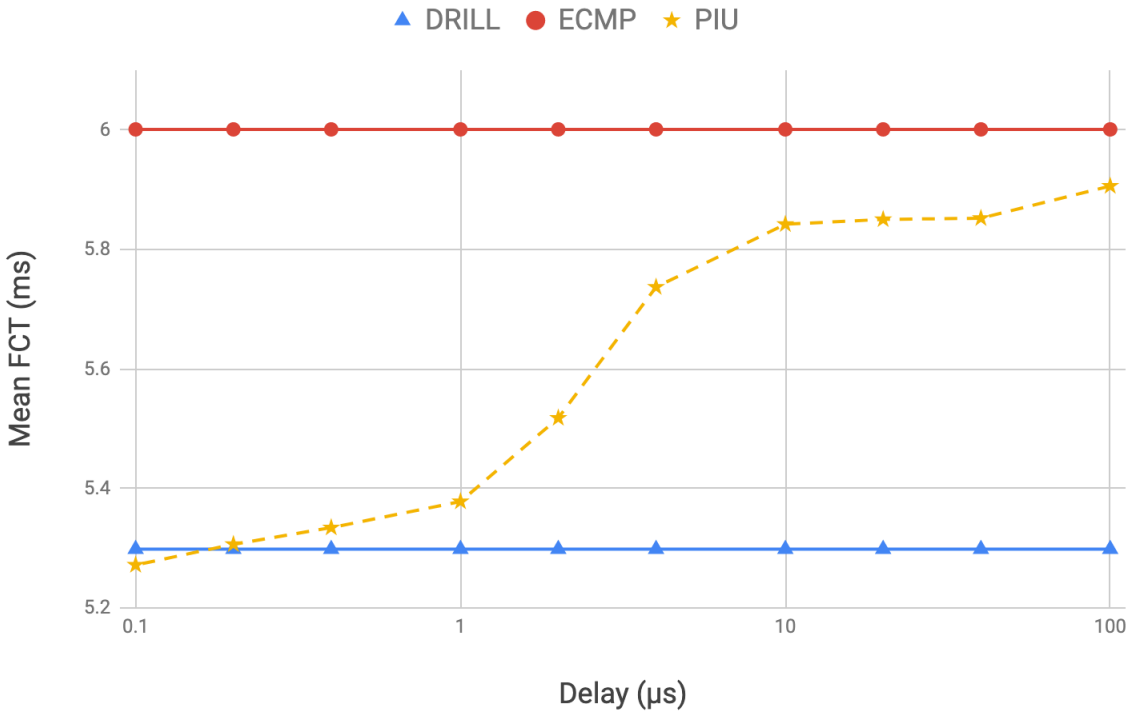

Figure 5.1: Effect of delay in queue length information on mean Flow Completion Times at a load of 40% (Note: logarithmic horizontal axis)

mean flow completion times. When the delay is increased beyond that, the rate of increase in mean FCT decreases and mean FCT slowly approaches that of ECMP.

At a load of 80% (Figure 5.2), the change in mean flow completion times is much more drastic. This is expected because at a high load, balancing traffic evenly is much more crucial and between update intervals, routing decisions for many incoming packets are made with stale information. At a delay of up to 1 microsecond, the performance is close to DRILL i.e. mean flow completion times increase by around 3%. As the delay increases to 30 microseconds, the mean flow completion time approaches that obtained by using ECMP and increases at a faster rate beyond that. When the update interval is 100 s, we are getting queue length updates after a high number of packet decisions (roughly more than 300 packet decisions for a 40 Gbps link). This makes load balancing granularity very coarse, as many

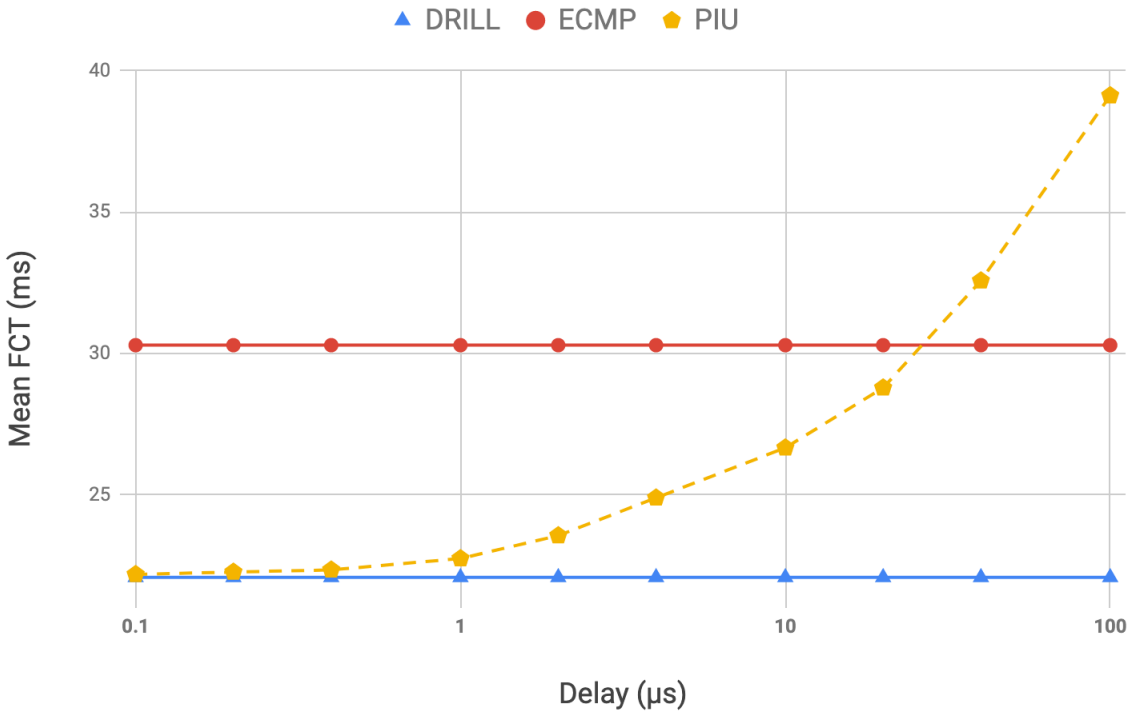

Figure 5.2: Effect of delay in queue length information on mean Flow Completion Times at a load of 80% (Note: logarithmic horizontal axis)

packets are routed based on the same state and formula. The flow completion time with a 100 s interval is around 1.8x higher than DRILL and 1.3x higher than ECMP.

We find that generally the mean flow completion time increases as the delay between updates increases. The effect is increasingly detrimental to mean FCT at higher loads.

These experiments can also be used to estimate performance of switches that have queue lengths available for monitoring purposes with random delays as mentioned in Section 4.3. So, for instance, if a switch gets queue lengths with a delay of between 2 s to 4 s, it can face an increase of 0.86-1.21% in mean FCT compared to DRILL at a load of 80%.

5.2.2 Varying Loads at Fixed Delays

To get an overall idea of how particular update intervals would perform in a datacenter environment with varying loads, we repeat the experiments with delays of 0.1 s, 1 s, 10 s and 100 s and increase the load from 10% to 90%. We use ECMP and DRILL as performance benchmarks.

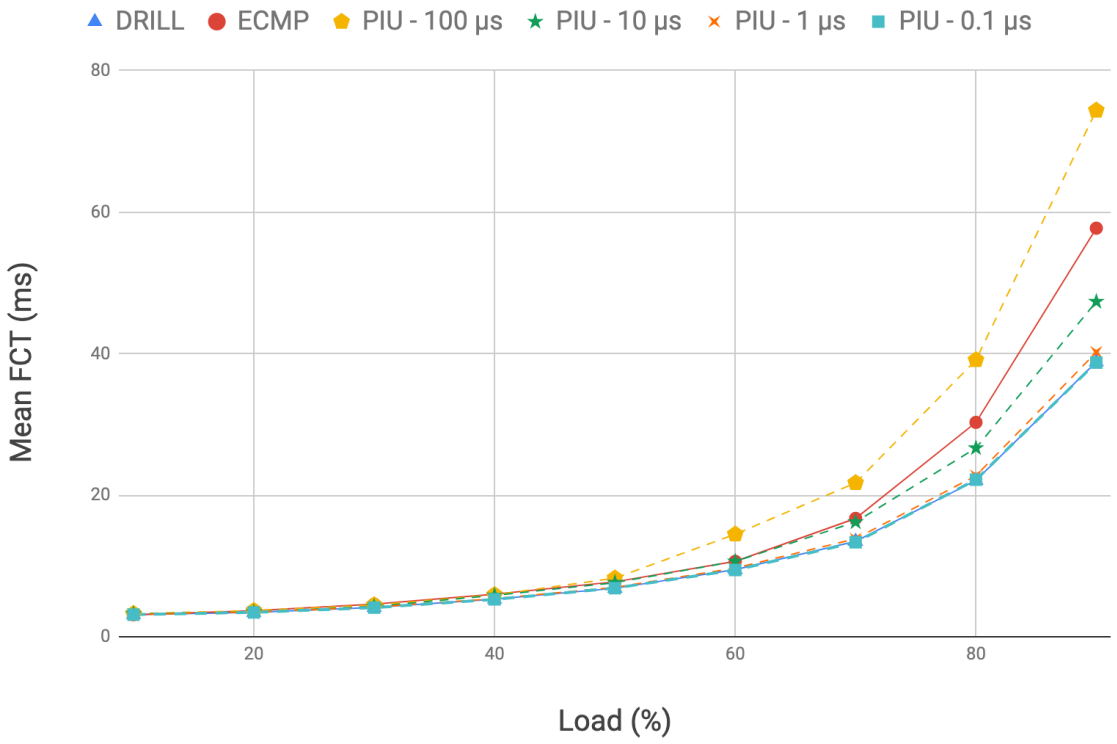

Figure 5.3: Mean Flow Completion Times of Per Interval Updates at intervals of 0.1 s to 100 s with increasing load

As shown in Figure 5.3, at a delay of 0.1 s, the mean FCTs are approximately equal to those of DRILL. This is expected because this simulates fresh updates for every packet decision (interval is less than packet transmission time). Increasing delays does not have a high effect on smaller loads. At intervals as high as 100 s, there is only a about a 10% increase in mean FCTs up to a load of 40%. As load increases, performance gets worse with

increase in update intervals. At a load of 90%, there is a 3.5%, 22% and 92% increase in mean FCT as compared to DRILL with update intervals of 1 s, 10 s and 100 s respectively. This shows that there is an substantial decrease in performance with an increase in delay.

5.3 EFFECT OF PER INCOMING PACKET UPDATES ON PERFORMANCE

We evaluate the approach discussed in Section 4.4.2 to update the egress queue length of the incoming link every packet uses. The mean flow completion times are close to those of DRILL. There is an increase in mean FCT of 6.5% at a 10% workload and 11% at a 90% workload compared to DRILL, as shown in Figure 5.4. So, the approach works well, in practice, even at higher loads.

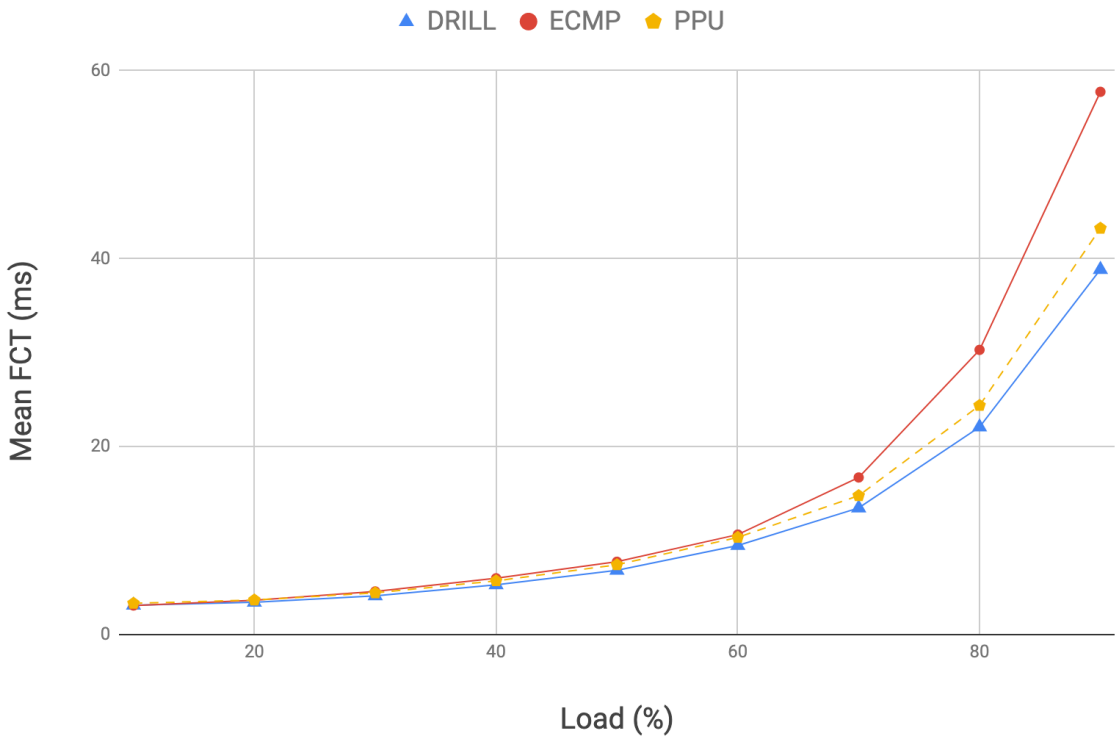

Figure 5.4: Mean Flow Completion Times (FCT) of Per Packet Updates in queue lengths with increasing load

Comparing this approach with the queue length update interval of 10 s or higher, we see a significant improvement at higher loads. At a load of 90%, there is a 42% decrease in mean FCT with Per Packet Update in queue length (43.2ms) compared to Per Interval Update of 10 s (47.4ms) However, at smaller loads between 10% to 30% , the performance is worse with an increase in mean FCT between 2.5% to 5% compared to Per Interval Update of 10 s.

We can assume that every leaf switch receives an equal amount of uplink and downlink traffic. For every packet that the leaf switch receives from an uplink queue, it updates the queue length of an outgoing link to a spine switch. Assuming an equal distribution of downlink traffic from all spine switches, we update the individual queue lengths once every $2n$ packets where n is the number of spine switches (4 for our evaluation). If we compare this to our implementation (Section 4.4.1) of getting updates at fixed intervals, this protocol is comparable to a delay of 2.4 s considering the uplink capacities of 40Gbps each and 100% utilization of links. This comparison imposes an upper limit as links are not always constantly utilized. At a smaller load, the inter-arrival time of packets is smaller so the performance is likely to correspond to a higher delay. Our evaluations support this. We find that at a load of 40%, the mean FCT for Per Packet Update (5.52 ms) is 3.8% higher than that of a Per Interval Update at 2 s (5.726 ms), while it is comparable to that of Per Interval Update at 4 s (5.738 ms). At an increased load of 80%, the mean FCT for Per Packet Update (24.37 ms) is 3.4% higher than that of a Per Interval Update at 2 s (23.56 ms) and crosses the graph at a Per Interval Update of approximately 3 s.

5.4 EFFECT OF COMBINED UPDATES ON PERFORMANCE

Finally, we implement a combination of the above two approaches and evaluate its performance. We piggyback queue lengths on incoming packets and after a fixed interval, query all queues to prevent uneven staleness and temporary starvation of links as explained in

Section 4.4.3. We run two different sets of experiments; varying delays under fixed loads and varying load under fixed delays.

We also include the results of the above two approaches in our graphs for better comparison. The Per Packet Updates serve as an upper bound on the mean FCT of Combined Updates, because theoretically, adding more periodic updates to this approach should not result in degradation of performance. So as the delay between update intervals increases, the performance should approach that of Per Packet Updates.

5.4.1 Varying Delays on Fixed Loads

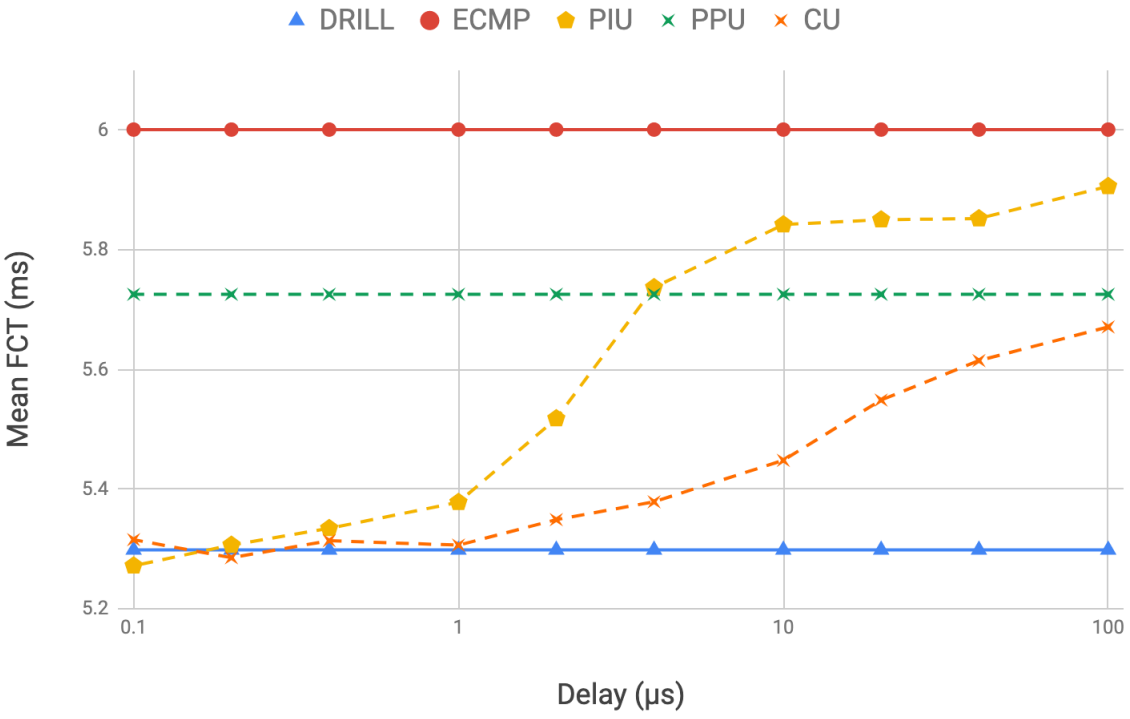

Figure 5.5: Effect of delay in queue length information on mean Flow Completion Times at a load of 40% (Note: logarithmic horizontal axis)

We run simulations under a moderate load (40%) and high load (80%) varying the delay from 100 nanoseconds to 100 microseconds with small intervals.

At a moderate delay of 40%, as shown in Figure 5.5, Combined Updates perform close to DRILL at smaller delays. At a delay of 10 s, the mean FCT (5.45ms) is a 2.8% increase to DRILL (5.30ms), compared to a 10.2% increase at 5.84ms for Per Interval Update at 10 s and a 8.1% increase at 5.73ms for Per Packet Updates. The performance approaches that of Per Packet Updates when increasing delay beyond 10 s.

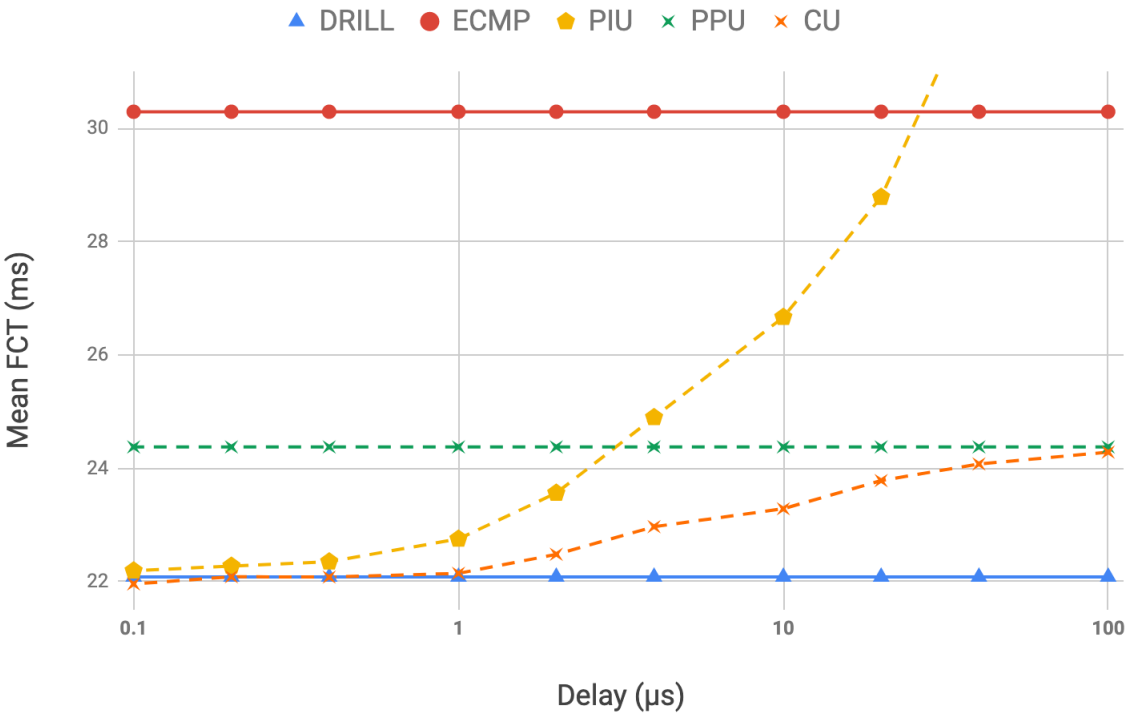

Figure 5.6: Effect of delay in queue length information on Mean Flow Completion Times at a load of 80% (Note: logarithmic horizontal axis)

At a high delay of 80%, as shown in Figure 5.6, mean FCTs of Per Interval Updates show an increase with delay and Per Packet Updates perform relatively better. Combined Updates performs close to DRILL at low delays and close to Per Packet Updates at high delays. At a delay of 10 s, the mean FCT (23.3ms) is a 5.4% increase to DRILL (22.1ms), compared to a 20.8% increase at 26.7ms for Per Interval Update at 10 s and a 10.4% increase at 24.4ms

for Per Packet Updates.

5.4.2 Varying Loads on Fixed Delays

We try different update intervals for the Combined Updates method and find out how they perform with increasing load.

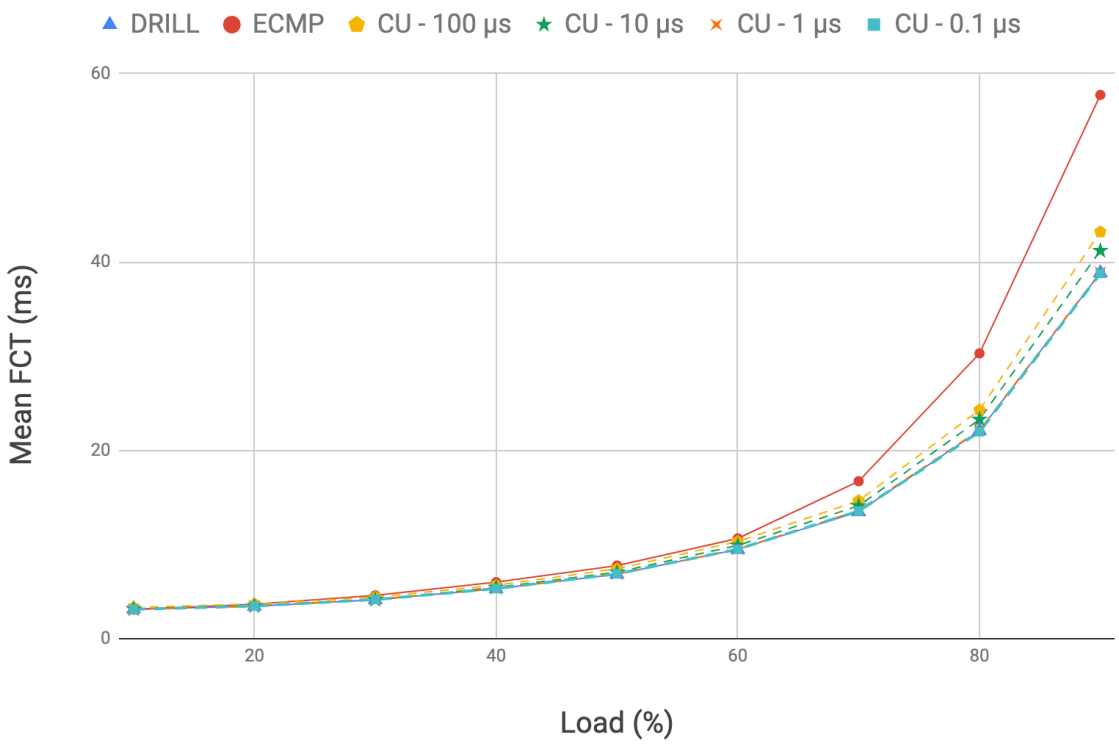

Figure 5.7: Performance of Combined Updates at various intervals with increasing load

At delays of 0.1 s and 1 s, the performance is almost exactly the same as that of DRILL (Figure 5.7). This is expected because in addition to single queue length updates from every incoming packet, we are getting all queue length updates at small intervals equivalent to up to 3 packets at a 100% utilization. At a load of 90%, compared to DRILL (38.8ms), there is a $\approx 0\%$ increase (38.8ms) at a delay of 0.1 s, a 0.2% increase (38.9ms) at a delay of 1 s, a 6.1% increase (41.2ms) at a delay of 10 s and a 11.2% increase (43.2ms) at a delay of 100 s.

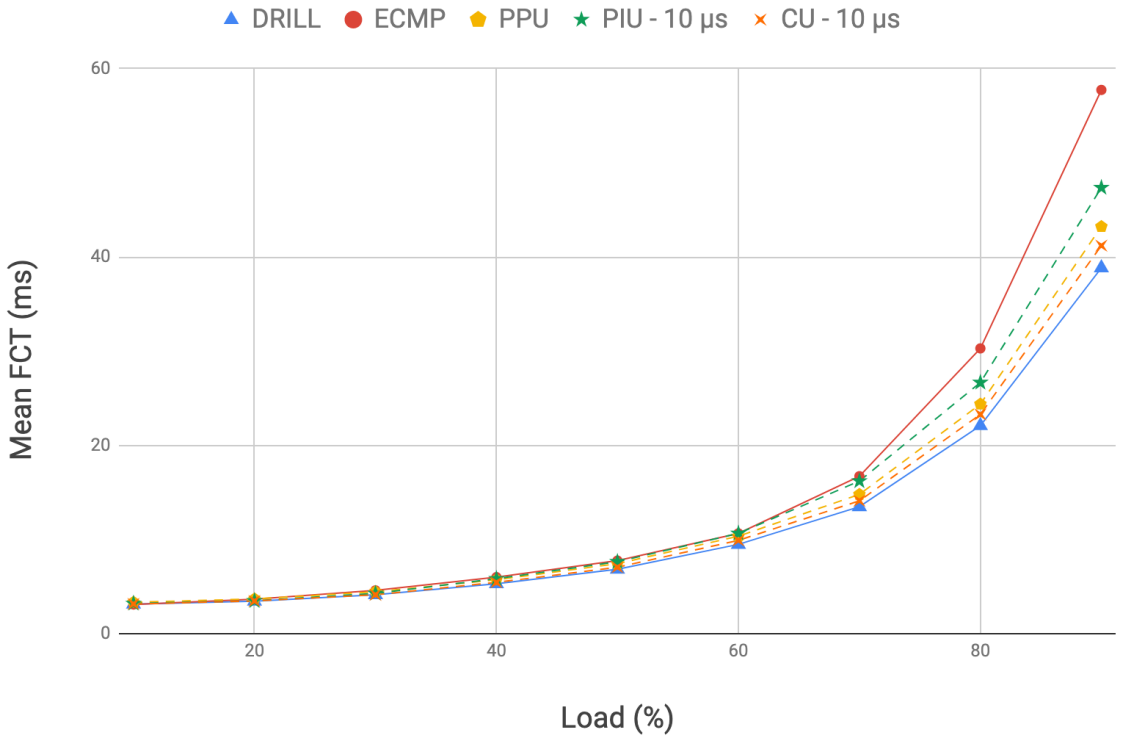

Figure 5.8: Comparison of DRILL, ECMP, Per Interval Updates at 10 s, Per Packet Updates, and Combined Updates at 10 s with increasing load

Figure 5.8 shows a comparison of all 3 approaches with intervals of 10 s for Per Interval Updates and Combined Updates. When comparing the increase in mean FCT with DRILL of Per Interval Updates at 10 s (Section 4.4.1), Per Packet Updates (Section 4.4.2) and Combined Updates with interval of 10 s (Section 4.4.3) respectively:

- At a small load of 10%, there is a 2.47%, 6.55% and 0.67% increase
- At a medium load of 50%, there is as 11.5%, 8.59% and 3.02% increase
- At a high load of 90%, there is a 21.9%, 11.3% and 6.08% increase

5.5 MEAN STANDARD DEVIATION OF QUEUES

We also study the standard deviation of outgoing queue lengths. Having a lower standard deviation between queue lengths represents an evenly distributed traffic load.

We sample queue lengths of all the uplink queues at leaf switches and downlink queues at spine switches, at an interval of 1 s for a duration of 5 seconds. We obtain the standard deviation of the 4 outgoing queue lengths at each switch and get the mean of queue length standard deviations for all leaf and spine switches.

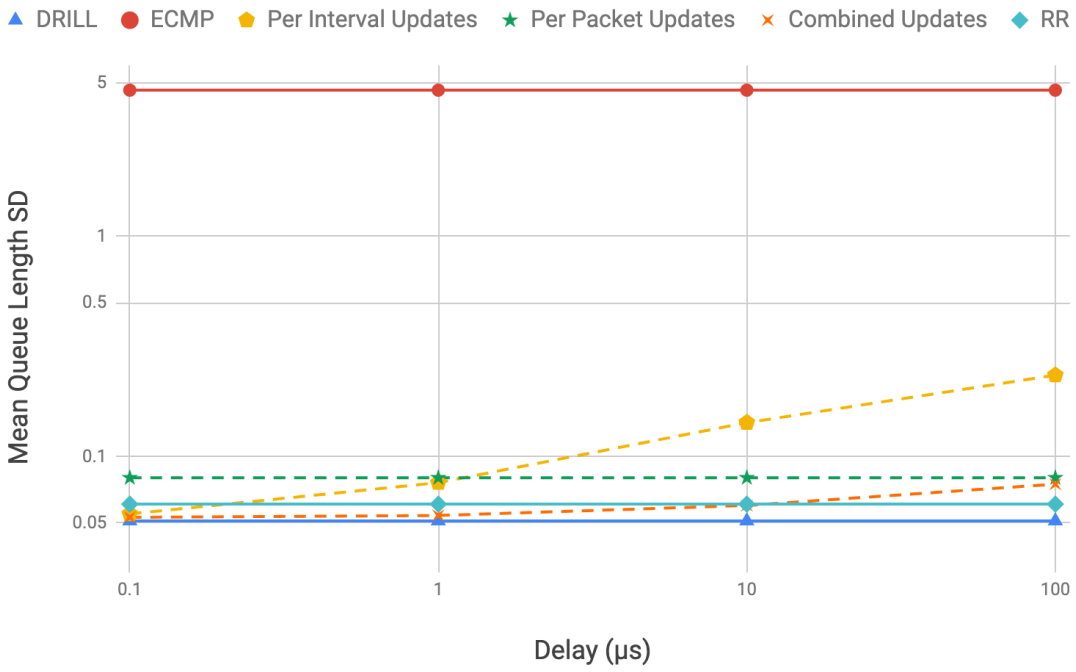

Figure 5.9: Mean queue length standard deviation against delay at a load of 40% (Note: logarithmic vertical and horizontal axes)

Our experiments show that there is a significant difference between the mean queue length standard deviations of DRILL and ECMP but that of Round Robin is only slightly higher than DRILL. At a load of 40% (Figure 5.9), DRILL ($\approx$ 0:05) has around 100x lower mean

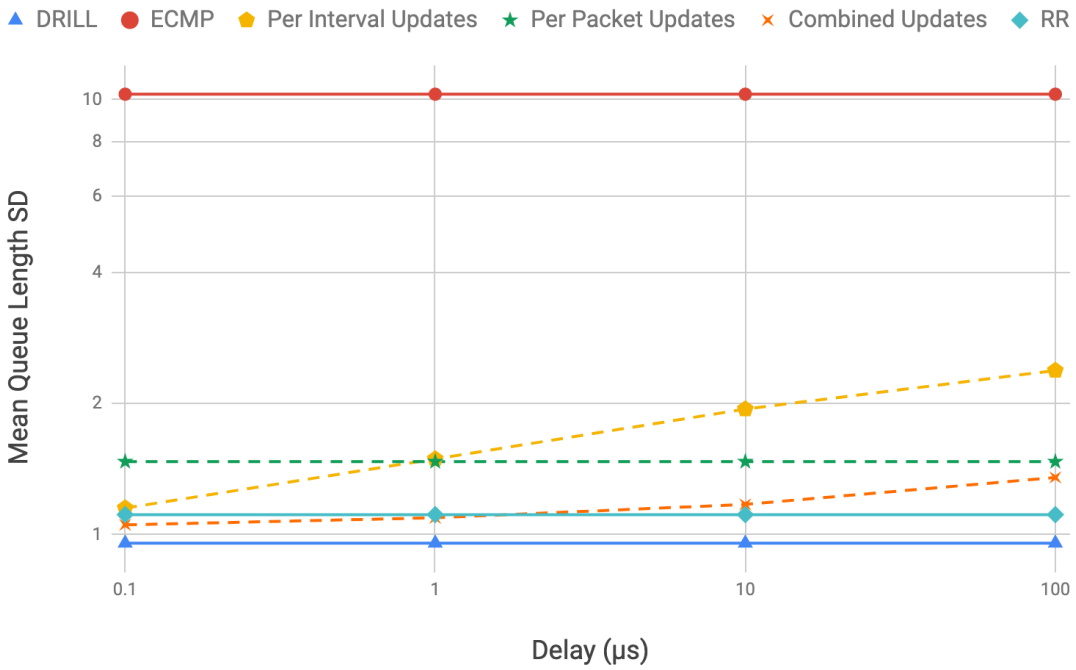

Figure 5.10: Mean queue length standard deviation against delay at a load of 80% (Note: logarithmic vertical and horizontal axes)

standard deviation than ECMP ($\approx$ 4:6) and at a load of 80% (Figure 5.10), DRILL ($\approx$ 0:95) has around 10x lower mean standard deviation than ECMP ($\approx$ 10).

We vary the delay in periodic updates from Section 4.4.1 and find an increase in mean standard deviation of queues. At a moderate load of 40% (Figure 5.9), when the delay is low, the mean standard deviation is close to DRILL but as the delay increases to 10 s, it increases to 0.14 (2.8x higher than DRILL) and as the delay increases to 100 s, it increases to 0.23 (4.5x higher than DRILL). At increased loads, the rate of increase in mean standard deviation is smaller but the values are more significant. Considering a load of 80% as shown in Figure 5.10, at a delay of 10 s, there is a 2.0x increase while at a delay of 100 s, there is a 2.5x increase.

Getting egress port queue length using incoming packets, as explained in Section 4.4.2,

raises the mean standard deviation to around 0.08 (60% increase compared to DRILL) at a load of 40% (Figure 5.9) and to around 1.45 (50% increase compared to DRILL) at a load of 80% (Figure 5.10). The mean standard deviation is comparable to the periodic updates approach (Section 4.4.1) at a delay of around 1.1 s at a 40% load and at a delay of around 850 ns at an 80% load.

However, when using the combination of the above two methods, as explained in Section 4.4.3, the difference in standard deviation decreases compared to the individual performance of these approaches. At a load of 40% (Figure 5.9), at a small delay of 1 s, the mean standard deviation is comparable to DRILL. At a delay of 100 s, the mean standard deviation is $\approx 1.4x$ higher than DRILL (compared to $\approx 4.5x$ for the periodic updates and $\approx 1.6x$ for the per incoming packet updates). At a higher load of 80%(Figure 5.10), at a small delay of 1 s, the mean standard deviation is $\approx 1.1x$ higher than DRILL (compared to $\approx 1.6x$ for the periodic updates and $\approx 1.5x$ for the per incoming packet updates) and at a higher delay of 100 s, it is $\approx 1.4x$ higher than DRILL (compared to $\approx 2.5x$ for the periodic updates and $\approx 1.5x$ for the per incoming packet updates).

As a side note, we observe that the standard deviation of spine switch queues is significantly larger than that of leaf switches under DRILL (0.044 at leaf switches and 1.87 at spine switches at a load of 80%) and its adaptations. This is because the leaf switches independently work to reduce variation in their queue sizes. They send an almost uniform distribution of load to spine switches so the sum of outgoing queue lengths at spine switches has a small variation. However, the lack of coordination among leaf switches might result in uneven distribution of packet destinations such that a downlink path from the spine switch might get congested. For instance, looking back at the topology in Figure 2.1, if we have a flow, F1, from H1 to H3 and a flow, F2, from H2 to H5, Leaf 1 might send a higher proportion of F1 packets on Spine 1 and a higher proportion of packets of F2 on Spine 2. This

will result in queue buildup on link 2 of Spine 1 and link 3 of Spine 2. Other switches might act in a similar manner aggravating the standard deviation of spine switch outgoing queues. While we could have achieved a more even distribution by looking at packet destinations when making routing decisions at leaf switches (for our example, dividing F2 and F3 packets more evenly among Spine 1 and Spine 2), this would have made the algorithm much more complicated and thus increased the packet processing time.

CHAPTER 6: FUTURE WORK

Piggyback More Queue Lengths: In this study, to get an egress port queue length, we take help of packets that use the ingress queues of the same link as egress port queue lengths would be accessible by that packet. In some switches, the incoming packets might have a greater visibility. Subsequently, they might be able to piggyback multiple egress queue lengths. This will result in more updated local state in switches and hence more accurate decisions. Ingress and egress ports in a switch might be organized in groups, in which case, the packet will only have access to egress port queue lengths within this group. So the packet can bring in all of these queue lengths or a subset. We can study how multiple queue length updates per packet affects the performance and how selection of a subset can also give us close to optimal results.

Include Updated Queue Length in Packet Decision: As discussed in Section 4.4.2, we can piggyback an egress port queue length on an incoming packet. This information is the most recent of the maintained queue length state, so the decisions made using this information are more accurate. DRILL picks the last used and two random links for routing decision. We can modify the algorithm to always include the most recently obtained queue length for the next packet routing decision and study the effect on performance.

Adapt algorithm to reduce downlink congestion at spine switches: As pointed out in Section 5.5, DRILL has a significantly higher mean standard deviation at spine switches compared to the leaf switches. This is because the algorithm does not look at the packet destination before making routing decisions. As a result, spine switches get an uneven distribution of downlink traffic causing congestion at these paths. We can explore possibilities of modifying the algorithm to divide traffic based on their destinations without making the algorithm too complicated.

Multiple Egress Port Queues: Switches often have multiple queues at each ingress and egress port to handle traffic with different priorities [21, 28] that complicates obtaining these queue lengths. DRILL uses absolute queue length per link to make routing decisions. We can study how the protocol changes if we only consider the queue length for the class the packet belongs to. Also, some switches monitor the mean queue depth per port which is conveniently available to the switch [29]. We can look at how DRILL performs if we select the link to forward the packet using mean queue of the physical link while oblivious to the variance within these queues and the packet is enqueued in the egress port queue corresponding to its priority.

Incremental Deployability: Since every switch running DRILL (original and adapted) makes packet routing decisions using locally available information and does not depend on any other component of the network to provide any other information, it is able to function on its own. So, we can replace a few switches in the network running the default protocols with ones running DRILL without any complications. While DRILL is incrementally deployable in theory, we do not know how it will perform. It would be interesting to study how a network would perform under a subset of nodes running DRILL and the difference in performance as we grow this subset.

Other Data Center Topologies and Workloads: DRILL was evaluated on several topologies (leaf spine, scalable leaf spine and asymmetric) and data center trace driven and synthetic workloads [7]. The approaches suggested in this study can also be more thoroughly evaluated on other symmetric (e.g. Fat-Tree [1], VL2 [6]) and asymmetric (e.g. BCube [20], Jellyfish [19]) topologies and synthetic and trace driven (e.g. Data Mining and Hadoop [3]) workloads to see how the delay threshold estimates change.

CHAPTER 7: CONCLUSION

DRILL [7] is a micro load balancing algorithm that makes fast routing decisions per packet using local information and outperforms state of the art protocols [9, 13]. However, the hardware deployability of the algorithm is complicated by the unavailability of queue length information at forwarding engines. We suggest two methods to obtain this information and evaluate their effect on flow completion times. Forwarding engines can maintain a local state of queue length information and update it at fixed intervals by exchanging update packets, however there is a tradeoff between freshness of data and overhead of generating packets. Using incoming packets to piggyback the egress port queue of the link they used reduces this overhead and works well at high loads but can lead to short term link starvation scenarios. We find that using a combination of these two techniques has at most a 6% increase in mean flow completion times even at a high loads.

REFERENCES

[1] C. E. Leiserson, "Fat-trees: Universal Networks for Hardware-efficient Supercomputing," *IEEE Trans. Comput.*, vol. 34, no. 10, pp. 892–901, Oct. 1985. [Online]. Available: http://dl.acm.org/citation.cfm?id=4492.4495

[2] A. Singh, J. Ong, A. Agarwal, G. Anderson, A. Armistead, R. Bannon, S. Boving, G. Desai, B. Felderman, P. Germano, A. Kanagala, J. Provost, J. Simmons, E. Tanda, J. Wanderer, U. Hölzle, S. Stuart, and A. Vahdat, "Jupiter Rising: A Decade of Clos Topologies and Centralized Control in Google's Datacenter Network," in *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication (SIGCOMM)*. ACM, 2015. [Online]. Available: http://doi.acm.org/10.1145/2785956.2787508 pp. 183–197.

[3] A. Roy, H. Zeng, J. Bagga, G. Porter, and A. C. Snoeren, "Inside the Social Network's (Datacenter) Network," in *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication (SIGCOMM)*. ACM, 2015. [Online]. Available: http://doi.acm.org/10.1145/2785956.2787472 pp. 123–137.

[4] B. Lebiednik, A. Mangal, and N. Tiwari, "A Survey and Evaluation of Data Center Network Topologies," *CoRR*, vol. abs/1605.01701, 2016.

[5] M. Al-Fares, A. Loukissas, and A. Vahdat, "A Scalable, Commodity Data Center Network Architecture," in *Proceedings of the ACM SIGCOMM 2008 Conference on Data Communication (SIGCOMM)*. ACM, 2008. [Online]. Available: http://doi.acm.org/10.1145/1402958.1402967 pp. 63–74.

[6] A. Greenberg, J. R. Hamilton, N. Jain, S. Kandula, C. Kim, P. Lahiri, D. A. Maltz, P. Patel, and S. Sengupta, "VL2: A Scalable and Flexible Data Center Network," in *Proceedings of the ACM SIGCOMM 2009 Conference on Data Communication*. ACM, 2009. [Online]. Available: http://doi.acm.org/10.1145/1592568.1592576 pp. 51–62.

[7] S. Ghorbani, Z. Yang, P. B. Godfrey, Y. Ganjali, and A. Firoozshahian, "DRILL: Micro Load Balancing for Low-latency Data Center Networks," in *Proceedings of the Conference of the ACM Special Interest Group on Data Communication (SIGCOMM)*. ACM, 2017. [Online]. Available: http://doi.acm.org/10.1145/3098822.3098839 pp. 225–238.

[8] A. Kabbani, B. Vamanan, J. Hasan, and F. Duchene, "FlowBender: Flow-level Adaptive Routing for Improved Latency and Throughput in Datacenter Networks," *CoNEXT 2014 - Proceedings of the 2014 Conference on Emerging Networking Experiments and Technologies*, pp. 149–159, 12 2014.

[9] K. He, E. Rozner, K. Agarwal, W. Felter, J. Carter, and A. Akella, "Presto: Edge-based Load Balancing for Fast Datacenter Networks," in *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication (SIGCOMM)*. ACM, 2015. [Online]. Available: http://doi.acm.org/10.1145/2785956.2787507 pp. 465–478.

[10] J. Cao, R. Xia, P. Yang, C. Guo, G. Lu, L. Yuan, Y. Zheng, H. Wu, Y. Xiong, and D. Maltz, "Per-packet Load-balanced, Low-latency Routing for Clos-based Data Center Networks," in *Proceedings of the Ninth ACM Conference on Emerging Networking Experiments and Technologies (CoNEXT)*. ACM, 2013. [Online]. Available: http://doi.acm.org/10.1145/2535372.2535375 pp. 49–60.

[11] M. Al-Fares, S. Radhakrishnan, B. Raghavan, N. Huang, and A. Vahdat, "Hedera: Dynamic flow scheduling for data center networks," in *Proceedings of the 7th USENIX Conference on Networked Systems Design and Implementation (NSDI)*. USENIX Association, 2010. [Online]. Available: http://dl.acm.org/citation.cfm?id=1855711.1855730 pp. 19–19.

[12] P. Gill, N. Jain, and N. Nagappan, "Understanding Network Failures in Data Centers: Measurement, Analysis, and Implications," in *Proceedings of the ACM SIGCOMM 2011 Conference*. ACM, 2011. [Online]. Available: http://doi.acm.org/10.1145/2018436.2018477 pp. 350–361.

[13] M. Alizadeh, T. Edsall, S. Dharmapurikar, R. Vaidyanathan, K. Chu, A. Fingerhut, V. T. Lam, F. Matus, R. Pan, N. Yadav, and G. Varghese, "CONGA: Distributed Congestion-aware Load Balancing for Datacenters," in *Proceedings of the 2014 ACM Conference on SIGCOMM*. ACM, 2014. [Online]. Available: http://doi.acm.org/10.1145/2619239.2626316 pp. 503–514.

[14] J. Perry, A. Ousterhout, H. Balakrishnan, D. Shah, and H. Fugal, "Fastpass: A Centralized "Zero-queue" Datacenter Network," in *Proceedings of the 2014 ACM Conference on SIGCOMM*. ACM, 2014. [Online]. Available: http://doi.acm.org/10.1145/2619239.2626309 pp. 307–318.

[15] M. Mitzenmacher, "The Power of Two Choices in Randomized Load Balancing," *IEEE Trans. Parallel Distrib. Syst.*, vol. 12, no. 10, pp. 1094–1104, Oct. 2001. [Online]. Available: https://doi.org/10.1109/71.963420

[16] C. Raiciu, S. Barre, C. Pluntke, A. Greenhalgh, D. Wischik, and M. Handley, "Improving Datacenter Performance and Robustness with Multipath TCP," in *Proceedings of the ACM SIGCOMM 2011 Conference*. ACM, 2011. [Online]. Available: http://doi.acm.org/10.1145/2018436.2018467 pp. 266–277.

[17] A. Dixit, P. Prakash, Y. C. Hu, and R. R. Kompella, "On the impact of packet spraying in data center networks," in *2013 Proceedings IEEE INFOCOM*, April 2013, pp. 2130–2138.

[18] M. Alizadeh, S. Yang, M. Sharif, S. Katti, N. McKeown, B. Prabhakar, and S. Shenker, "pFabric: Minimal Near-optimal Datacenter Transport," in *Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM*. ACM, 2013. [Online]. Available: http://doi.acm.org/10.1145/2486001.2486031 pp. 435–446.

[19] A. Singla, C.-Y. Hong, L. Popa, and P. B. Godfrey, "Jellyfish: Networking Data Centers Randomly," in *Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation (NSDI)*. USENIX Association, 2012. [Online]. Available: http://dl.acm.org/citation.cfm?id=2228298.2228322 pp. 17–17.

[20] C. Guo, G. Lu, D. Li, H. Wu, X. Zhang, Y. Shi, C. Tian, Y. Zhang, and S. Lu, "BCube: A High Performance, Server-centric Network Architecture for Modular Data Centers," in *Proceedings of the ACM SIGCOMM 2009 Conference on Data Communication*. ACM, 2009. [Online]. Available: http://doi.acm.org/10.1145/1592568.1592577 pp. 63–74.

[21] "6800 Series 10 Gigabit and Gigabit Ethernet Interface Modules for Cisco 6500 Series Switches Data Sheet," Accessed: 2019-04-11. [Online]. Available: https://www.cisco.com/c/en/us/products/collateral/switches/catalyst-6500-series-switches/datasheet-c78-451794.html

[22] "Arista 7300 Series," Accessed: 2019-04-11. [Online]. Available: https://www.arista.com/en/products/7300-series

[23] "Micro-Burst Monitoring," Accessed: 2019-04-11. [Online]. Available: https://www.cisco.com/c/en/us/td/docs/switches/datacenter/nexus6000/sw/qos/7x/b_6k_QoS_Config_7x/micro_burst___monitoring.pdf

[24] "Latency Analyzer (LANZ) A New Dimension in Network Visibility," Accessed: 2019-04-11. [Online]. Available: https://www.arista.com/assets/data/pdf/TechBulletins/Lanz.pdf

[25] "The Network Simulator NS-2." [Online]. Available: http://www.isi.edu/nsnam/ns/

[26] "DRILL under Delayed Sampling." [Online]. Available: https://github.com/fatimat2/drill-with-delayed-sampling

[27] "Arista 7150 Series Network Switch," Accessed: 2019-04-11. [Online]. Available: https://www.arista.com/en/products/7150-series-network-switch-datasheet

[28] "Cisco Catalyst 4500 Series Line Cards Data Sheet," Accessed: 2019-04-11. [Online]. Available: https://www.cisco.com/c/en/us/products/collateral/interfaces-modules/catalyst-4500-series-line-cards/product_data_sheet0900aecd802109ea.html

[29] "Understanding Queue Limits and Output Drops on Cisco IOS Software Platforms," Accessed: 2019-04-11. [Online]. Available: https://www.cisco.com/c/en/us/support/docs/routers/7200-series-routers/110850-queue-limit-output-drops-ios.html