Micro Load Balancing in Data Centers with DRILL

Soudeh Ghorbani[§] Brighten Godfrey[§] Yashar Ganjali[#] Amin Firoozshahian[*]
[§]University of Illinois at Urbana-Champaign [#]University of Toronto [*]Intel

Abstract

The trend towards simple data center network fabric strips most network functionality, including load balancing capabilities, out of the network core and pushes them to the edge. We investigate a different direction of incorporating minimal load balancing intelligence into the network fabric and show that this slightly smarter fabric significantly enhances performance. We provide a very simple in-network load balancing scheduling algorithm called DRILL which is purely local to each switch. DRILL leverages local load sensing and randomization concepts to distribute load among multiple paths. Through simulation, we show that this simple approach outperforms CONGA, a recent global edge-based load balancing scheme for data centers. We also formally prove the switch-level stability and throughput-efficiency of DRILL's scheduling algorithm.

Categories and Subject Descriptors

C.2.1 [**Computer-Communication Networks**]: Network Architecture and Design

General Terms

Algorithms, Design, Performance

1. INTRODUCTION

Data centers are overwhelmingly built as topologies that are characterized by large path diversity such as Clos networks (Figure 1) [18, 7, 19, 27, 23, 5, 24, 27, 4]. A critical issue in such networks is the design of an efficient algorithm that can evenly balance the load among available paths. While Equal Cost Multi Path (ECMP) is extensively used in practice [19], it is known to be far from optimal for efficiently exploiting all available paths [19, 7, 11]. Data center measurement studies, for instance, indicate that a significant fraction of core links regularly experience congestion despite the fact that there is enough spare capacity elsewhere [8].

Many proposals have recently tried to address this need [7, 19, 6, 15, 11]. Aligned with the recent trend of moving functionality out of the network fabric [12], these proposals strive to delegate load balancing to centralized controllers [28, 6, 10, 13], to the network edge [7], or even to end-hosts [19, 11]. This recent move of the load balancing functionality is motivated largely by the *perceived necessity of having global congestion or load information* about the potential paths for evenly balancing the load among them [7, 28, 6, 10, 13]. Collecting global traffic information and routing based on that information could more easily be managed at separate controllers or at the edge. A notable example is CONGA [7], a recent in-network load balancing scheme that gathers and analyzes congestion feedback from the network at the network edge (leaf switches in Clos networks) to make load balancing decisions. CONGA's central thesis is that *global congestion information is fundamentally necessary for evenly balancing the load.*

We explore a different direction: What can be achieved with decisions that are local to each switch? We refer to this approach as *micro load balancing* because it makes "microscopic" decisions within each switch without global information, and because this in turn allows decisions on microsecond (packet-by-packet) timescales.

Micro load balancing has hope of offering 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 data centers (§4). It has been shown that majority of congestion events in data centers are short-lived [8, 21]. The bulk of microbursts that are responsible for over 90% of packet loss, for instance, last for less than 3 microseconds [9]. Systems that attempt to collect and react based on global congestion information typically have orders of magnitude slower control loops than the duration of the majority of congestion events [29, 7]. For example, even though CONGA adds mechanisms to leaf and spine switches to assist in obtaining congestion information, it still typically requires a few RTTs (tens to hundreds of microseconds), by which time the congestion event is likely already over. In addition, we

HotNets-XIV, November 16 - 17, 2015, Philadelphia, PA, USA

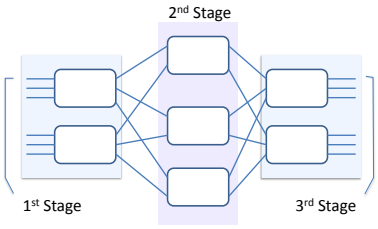
ACM 978-1-4503-4047-2/15/11 ...$15.00
DOI: http://dx.doi.org/10.1145/2834050.2834107.

found that amassing macroscopic traffic information can lead to a pitfall: feeding global traffic information to distributed, non-coordinating sources of traffic (input ports of all the leaf switches in CONGA) can cause them to select the same set of least-congested paths in a synchronized manner which in turn leads to bursts of traffic in those paths.

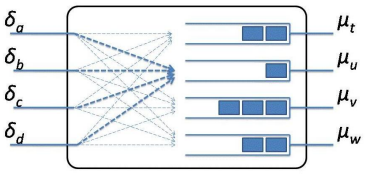
To study whether micro load balancing offers a viable solution, we designed and evaluated DRILL (Distributed Randomized In-network Localized Load-balancing). DRILL is in essence a switch scheduling algorithm that acts only based on local switch queue length information without any coordination among switches or any controllers. Even within a single switch, deciding how to route and schedule packets is nontrivial. DRILL's scheduling algorithm is inspired by the "power of two choices" paradigm [26]. To make it practical for packet routing within a data center switch, we extend the classic design to accommodate a distributed set of sources (input ports) and show that the key stability result holds in the distributed version as well (§2, §3.2, §5). More concretely, DRILL assumes that a set of candidate next-hops for each destination have been installed in the forwarding table, using well-known mechanisms such as the shortest paths (as in ECMP). Next, upon arrival of each packet at any input port, that input port, independently and with no coordination with other input ports, compares the queue lengths of two randomly-chosen candidate output ports and the port that was least loaded during the previous samplings, and sends the packet to the least loaded of these three candidates. Note that this is unlike ECMP since the decision is based on local load rather than static hashing of the packet header. We show how to optimize DRILL's parameters—number of choices and amount of memory—so as to avoid damaging synchronization effects where many input ports choose the same output.

In contrast to the works that operate on a global "macroscopic" view of the network, DRILL's micro load balancing enables it to instantly react to load variations as the queues start building up. DRILL results in dramatically better performance than CONGA in heavily loaded systems (78% improvement in average flow completion time §4) and in incast scenarios (2.5× improvement in average flow completion time §4). In addition, DRILL offers a simpler switch implementation than CONGA since DRILL does not need to detect flowlets or send and analyze feedback.

Presto [19], a very recent host based load balancing scheme, offers an interesting comparison point to DRILL. Unlike schemes with global information, Presto is congestion-oblivious. Presto argues that the main culprit of inefficiencies in schemes like ECMP is the coarse granularity: each flow, even a large one, hashes all its packets onto one path. Therefore, Presto partitions flows into equal size chunks of 64KB, called flowcells, and "sprays" them in a round-robin fashion among available paths. This can be executed by the source with a form of source routing, releasing the network from that burden. A key assumption in this design is that *the small size and size-uniformity of data units is sufficient for preserving balanced load in symmetric topologies*. Our simulations confirm that Presto outperforms CONGA in non-incast scenarios, but DRILL in turn performs better than Presto (§4). DRILL's improved performance even for identical small size flows (Presto's ideal setting) results from (a) the load adaptation of DRILL, in contrast to the load-agnostic nature of Presto, and (b) balancing finer granularity of load: packets vs. flowcells. We also show that DRILL has significantly better flow completion time in an incast scenario (9.5× better than Presto) because of its fast reaction to congestion (§4).



(a) A 3-stage Clos network.



(b) An arbitrary switch in the first-stage of a Clos network.

Figure 1: Clos networks.

DRILL's micro load balancing raises several concerns. First, how can we deal with packet reordering that results from load balancing at sub-flow granularities? Interestingly, we find that in a symmetric Clos data center network, DRILL balances load so well that packets nearly always arrive in order despite traversing different paths. This is because queue lengths have very small variance and hence packets have almost identical queueing delays, even under heavy load (§4.2). Regardless, the occasional re-orderings could still adversely affect TCP's performance. Hence, similar to prior work [15, 19], we deploy a buffer under TCP to restore correct ordering of packets. Practical challenges of deploying such a technique are solidly addressed and solved by Presto [19]. Compared to Presto, DRILL causes significantly less frequent out of order delivery of packets, shorter buffers, and smaller buffering latencies (§4.2).

The second concern is that load-based scheduling algorithms within a switch could result in instability and hence low throughput [25]. Therefore, we formally prove DRILL's switch-level stability and show that it guarantees 100% throughout for admissible traffic (§5).

Third, is DRILL's micro load balancing sufficient alone, or is some macroscopic information necessary? Our goal in this short paper is to demonstrate that micro load balancing can achieve much better performance than schemes that use more global information, and indeed DRILL achieves this with no global information whatsoever. For topological changes such as link failures that reduce the number of paths, however, it is likely that more global path planning is necessary. Since these changes, unlike congestion, happen in slow time scales [17], leveraging existing distributed or centralized topology-information dissemination techniques might be sufficient to augment DRILL. We leave the study of such techniques for future work.

In summary, our results strongly indicate that micro load balancing belongs in the data center fabric to achieve the key goal of high performance traffic delivery; and that a significant and interesting question for future research is when and how micro load balancing and macroscopic information should be combined to get the best of both worlds.

2. RELATED WORK

The Power of Two Choices: DRILL is inspired by the seminal "power of two choices" work on using randomized load-sensitive algorithms for load balancing [26]. Mitzenmacher shows that in the *supermarket model*, with a single input queue and many output queues, the load balancing performance greatly improves with using a small amount of choice, *e.g.*, $d = 2$, over random placement $d = 1$, and deploying a single unit of memory along with random sampling to save the identity of the output queue with the shortest queue length from the previous time slots guarantees stability [26]. Central to this design is having *one* arbiter responsible for balancing the load among multiple servers [26]. The *ideal* load balancing approach with optimal performance in this model, for instance, is to have the single arbiter compare the length of *all* queues before making load balancing decisions for each packet.

In large scale networks with high transmission rates, deploying a central arbiter to make load balancing decision for each packet faces great scalability challenges and even in highly optimized situations could cause reduction in throughput [28]. Ideally, we would like to make this approach completely distributed: whenever a packet arrives at any input port, that input port, independently and with no coordination with other input ports, assigns the packet to the least loaded of d randomly chosen output queues, where $d \ll N$ (N is the number of output ports).

This very simple distributed extension, however, while ideal for scalability and simplicity of design, could have a vastly different performance compared to the original model, given that the load balancing decisions of different sources can interfere with each other. As a simple illustrative example, consider Figure 1 (b): If all input ports independently execute the ideal algorithm in the supermarket model by comparing the length of all output queues, they could all direct their packets to the same output resulting in huge bursts and load imbalance in the network (more in §3.2). It is therefore not obvious what guarantees of the original model remain valid under the distributed adoption.

Via extensive simulations with various workloads and switches of various sizes, we show that having two samples not only balances the load near-optimally (standard deviation of queues always remain small), but also, unlike the original model, performs more efficiently than having a large number of samples (§3.2). Intuitively, this results from the reduced risk of many input queues making identical choices, overwhelming one output port while leaving others underutilized.

Load Balancing in Data Centers: Recent works attribute the poor performance of ECMP to two key factors: (a) its lack of global congestion information, and (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 [29]. Fastpass positions that each sender should delegate control to a centralized arbiter to dictate when and through which path each packet should be transmitted [28]. Hedera [6], MicroTE [10], and Mahout [13] re-route "elephant" flows to compensate for the inefficiency of ECMP hashing them onto the same path.

In the second category, Presto argues that in a symmetric network topology where all flows are "mice", ECMP provides near optimal load balancing [19]. Presto then divides flows into equal-size "flowcells" at the hosts. The hosts then proactively route the cells via source routing. They also deploy a centralized controller to react to occasional asymmetries in topology such as failures. Other efforts in this category include dividing flows into "flowlets" [20, 7] and balancing flowlets instead of flows, or per-packet spreading of traffic in a round robin fashion [11, 15]. Presto's choice of flowcells (64KB pieces of flows) is motivated by the fact that flowlets are coarse grained, and is dictated by the practical challenges of performing per-packet load balancing *in the hosts*. The assumption among the work in the second category is that ECMP's inefficiency is mainly caused by the large flows, and therefore could be addressed by hosts splitting flows into small chunks and routing them separately in a proactive manner, with no need for congestion feedback.

CONGA takes a hybrid approach by both splitting traffic into flowlets and using in-network congestion feedback mechanisms to estimate load and allocate flowlets to paths based on the congestion feedback. Their main thesis is that efficient load balancing requires global information. Our experiments indicate that DRILL's micro load balancing outperforms these proposals, in symmetric topologies.

3. DESIGN AND ALGORITHMS

In this section, we first explain our assumptions before formalizing our algorithms and showing that while having two choices is critical to efficiency of our algorithms, having too many choices (*i.e.*, $d \gg 2$) degrades performance.

3.1 Assumptions and model

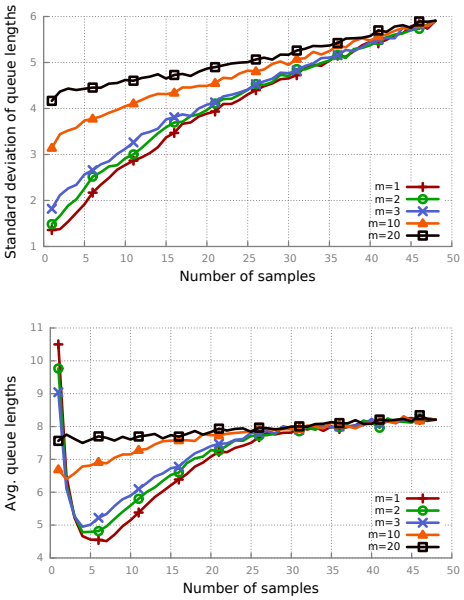


Figure 2: Setting parameters.

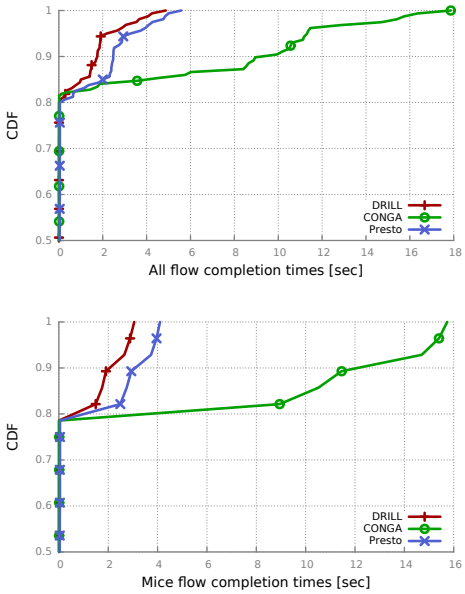


Figure 3: Flow completion time.

We consider an $M \times N$ combined input output queued switch with FIFO queues in which the arrivals are independent, *i.e.*, arrivals at each input are independent and identically distributed ($i.i.d$) and arrival processes at each input are independent of arrivals at other inputs, and arriving packets could be forwarded to any of the N output ports. Traffic is also assumed to be 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 ($1 \leq i \leq M$) and μ_j is the service rate of output queue j ($1 \leq j \leq N$).

We place *no restriction on the heterogeneity of arrival rates or service rates*. These rates could 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 [17].

3.2 Algorithms: (d, m) scheduling policies

We consider randomized scheduling in which, upon each packet arrival, the input port chooses d random outputs out of possible N queues, finds the queue with minimum occupancy between these d samples and m least loaded samples from previous time slot, and routes its packet to that port. Finally, the input port updates the contents of its m memory units with the identity of the least loaded output queues. Such policies are called (d, m) policies, hereafter.

3.3 Setting the right parameters: the pitfalls of choice and memory

We show in §5 that for stability it is necessary to set $m \geq 1$. Our experiments with various loads and different network sizes show that while there is a significant improvement in the load balancing performance for $d > 1$ compared to $d = 1$, increasing d and m to values $\gg 2$ and $\gg 1$, respectively, degrades the performance. The reason, as explained earlier, is that the larger number of random samples or memory units makes it more likely for a large number of input ports to simultaneously select the same output port.

Figure 2 demonstrates this for a 48-port switch under 100% load[1]. We measure average standard deviation of output queue lengths (an ideal load balancing scheme should be able to persistently preserve low standard deviation of queue lengths), queueing delay, average queue lengths and throughput over 10 trials. We observe optimal performance for $d \ll N$ schemes for this setup and with other switch sizes (32-port and 256-port) and different loads (10% to 100%). In our experiments, we use the $(2, 1)$ scheme.

4. SIMULATIONS

We measure the flow completion time and throughput under DRILL, CONGA, and Presto via simulation.

We use the OMNET++ simulator [3], the INET framework [1] and the Network Simulation Cradle library which contains real-world TCP implementations taken from Linux 2.6 [2] for our experiments. We use the INET's `Ethernet Switch` and `Standard Host` modules that implement regular Ethernet switches and hosts with standard networking stacks, and add DRILL, CONGA, and Presto functionalities to them. The topology used in our experiments is a Clos network with 4 and 6 leaf and spine switches respectively, and 16 hosts attached to each leaf switch. A heavy-tailed distribution is used for generating flow sizes, with median flow size of 1KB similar to prior works [7, 14]. Hosts run two TCP applications each. Similar to [7], each TCP con-

[1]More experimental details in §4.

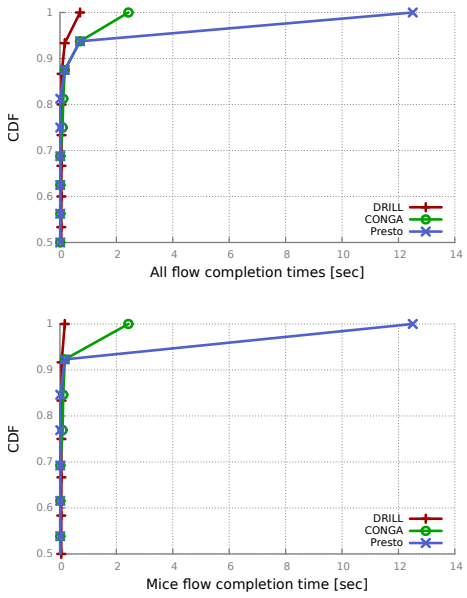


Figure 4: Flow completion time in an incast scenario.

nection request flows according to a Poisson process from randomly chosen hosts under a different rack.

4.1 DRILL improves flow completion time in heavily loaded networks

In lightly loaded networks, DRILL, CONGA, and Presto result in almost similar performance. With flow sizes and arrival rates drawn from [14], for instance, the network will be around 20% loaded and the flow completion time in DRILL is 30% and 11% smaller than, respectively, CONGA and Presto (not shown in figures).

As the load increases, however, the gap between these schemes widens. Similar to [19], we scale the distributions by a factor of 5 to emulate a heavier workload and observe 32% and 78% improvement for flow completion time compared to Presto and CONGA, respectively. The improvement of flow completion times of mice flows ($< 100KB$) is respectively 33% and 82%, compared to Presto and CONGA, and 41% and 83% for elephant flows ($> 10M$). The CDFs of flow completion times for all flows and for only mice flows are shown in Figure 3.

We also observe a significant improvement in incast scenarios common in data centers [7]. We use a simple application that generates incast traffic patterns by making requests to 16 servers located under a different rack. The servers reply in a synchronized fashion causing incast. There is background traffic as explained above, and servers' flow sizes are generated the same way. Figure 4 shows the CDF of all and mice flow completion times. We observe $9.5\times$ and $2.5\times$ improvement compared to Presto and CONGA. The improvement is more significant for mice flows ($21.1\times$ and $4.8\times$ compared to Presto and CONGA), but is negligible for elephant flows.

4.2 Out of order delivery of packets

Well-balanced load and extremely low variance among queue lengths with DRILL imply that it rarely causes re-ordering of packets. In lightly and moderately loaded systems, usually transmission delays of consecutive packets are sufficient for making them be delivered in order. In our experiments, we observe no packet re-ordering under low load. As the load increases, however, hosts occasionally receive out of order packets. Such packets are buffered and re-ordered in DRILL before passing them to TCP. In the experiments of the previous section with heavy load (Figure 3), we observe around 0.02% of packets (2 packets in every 10,000 packets) being delivered out of order.

We observe about 2 times more packets being delivered out of order with Presto (0.04%) despite the fact that one of their main motivations of balancing "flowcells" (rather than packets) is to avoid excessive out of order delivery of packets. Moreover, when there is any packets being buffered, most often, there is only one packet buffered with DRILL (median buffer size $= 1$), and on average, the buffered packets spend $8\mu sec.$ in the buffer. With Presto, however, the median buffer size is 5, and buffered packets spend $22\mu sec.$ in the buffer before being passed to TCP.

We observe similar trends in the incast scenario (Figure 4). With DRILL, 0.01% of packets arrive out of order; the median buffer size is 2, and the average buffering latency for the few out-of-order packets is $23\mu sec.$. Under Presto, 12 times more packets (0.12% of packets) are delivered out of order, the median buffer size is 26, and it adds $74\mu sec.$ of delay to the buffered packets to restore their correct ordering.

4.3 Turning flows into equal-size mice is not enough

A key assumption in some recent load balancing proposals, is that turning flows into equal size pieces is enough for achieving optimally balanced load in symmetric topologies [19, 11, 15]. Presto, for instance, divides flows into flowcells and routes each flowcell in a round robin fashion, irrespective of congestion feedback,

We try to test this hypothesis by measuring various load balancing performance metrics (queue length standard deviation, average queue length, latency, throughput) when flows have exactly identical sizes, under DRILL and Presto. Our results show significant improvement in balancing the load (substantially lower standard deviation of queues), and consistent and higher throughput under DRILL compared to Presto even for identical size flows in symmetric topologies (the ideal case for Presto). Queue length standard deviation, for instance, is about three orders of magnitude larger under Presto compared to DRILL in a symmetric system with 100% load (Figure 5; error bars show 5th and 95th percentiles).

The improvement comes from the congestion-aware nature of DRILL and balancing the load in finer granularity (packets instead of flowcells).

5. DRILL GUARANTEES STABILITY

In this section, we first prove that purely randomized algorithms without memory, *i.e.*, $(d, 0)$ schemes are unstable, before proving the stability of (d, m) schemes for $d, m > 0$.

5.1 Instability of random sampling without memory

First, we consider the $(d, 0)$ policy, *i.e.*, the algorithm in which every input port chooses d random outputs out of possible N queues, finds the queue with minimum occupancy between them and routes its packet to it. By Theorem 1, we prove that such algorithm cannot guarantee stability, *i.e.*, the expected length of the output queues can grow without bound.

THEOREM 1. *For admissible i.i.d. parallel arrival processes, $(d, 0)$ policy cannot guarantee stability for any arbitrary number of samples d, where d is less than the number of outputs.*

PROOF. Let δ_i be the arrival rate to input port i, and μ_j be the service rate of output queue j. Now consider output queue N. For any input port, the probability that it chooses output queue N as a sample is $\frac{d}{N}$. So, maximum arrival rate to queue N is $\frac{d}{N} \times \sum_i^M \delta_i$. Thus, the minimum arrival rate to the remaining $N-1$ output queues is

$$\zeta = \sum_i^M \delta_i - \frac{d}{N} \times \sum_i^M \delta_i = (1 - \frac{d}{N}) \times \sum_i^M \delta_i.$$

Clearly, if ζ is larger than the sum of the service rates of these $N-1$ queues, the system will be unstable. □

It should be noted that the argument does not hold: (a) when there are some restrictions regarding the arrival or service rates, like when the service rates are equal, or (b) when $d = N$. These two special cases, however, are of little interest, since the former opts out some admissible patterns of traffic, and the latter nullifies the benefit of randomization and as shown in §3.2 results in bursty and unbalanced load. The results of our experiments suggest that the system will perform well with $d \ll N$.

5.2 Stability of random sampling with memory

In §5.1, we showed that randomized policy cannot guarantee stability without using unit of memory. Similar to [25] and using the results of Kumar and Meyn [22], we prove that DRILL's scheduling algorithms are stable for all uniform and nonuniform independent arrival processes up to a maximum throughput of 100%.

THEOREM 2. *$(1, 1)$ policy is stable for all admissible i.i.d. parallel arrival processes.*

To prove that the algorithm is stable, we show that for an $M \times N$ switch scheduled using the $(1, 1)$ policy, 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 $k > 0$ and $\epsilon > 0$ are some constants.

We do so by defining $V(n)$ as:

$V(n) = V_1(n) + V_2(n)$, where $V_1(n) = \sum_{i=1}^N V_{1,i}(n)$, $V_{1,i}(n) = (\tilde{q}_i(n) - q^*(n))^2$, and $V_2(n) = \sum_{i=1}^N q_i^2(n)$. $q_k(n)$, $\tilde{q}_i(n)$ and $q^*(n)$, respectively, represent the length of the k-th output queue, the length of the output queue chosen by the input i, and the length of the shortest output queue in the system under $(1, 1)$ policy, at time instance n. Details of the proof are included in [16].

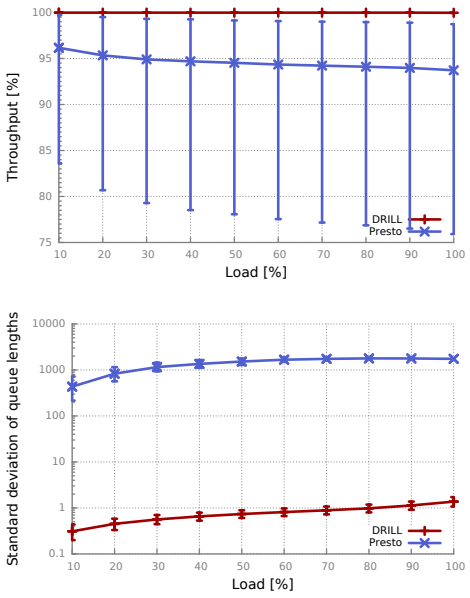


Figure 5: Turning flows into equal-size mice is not enough.

6. CONCLUSION

In contrast to the currently pervasive approach of balancing the load based on global and macroscopic view of traffic, we explore an alternative approach of *micro load balancing*: enabling the network fabric to make load balancing decisions at microsecond time scales based on traffic information local to each switch. Our experiments show that our simple provably-stable switch scheduling algorithm, DRILL, outperforms the state-of-the-art load balancing schemes in Clos networks, particularly under heavy load. We leave the investigation of micro load balancing in other (especially asymmetric) topologies to future work.

Acknowledgments

We would like to thank the anonymous reviewers and Anduo Wang for their feedback. We gratefully acknowledge the support from NSF grant 1423452. Soudeh was supported by a VMWare Graduate Fellowship.

7. REFERENCES

[1] INET Framework. https://inet.omnetpp.org/.

[2] Network Simulation Cradle Integration. https://www.nsnam.org/wiki/Network_Simulation_Cradle_Integration.

[3] OMNeT++ Discrete Event Simulator. https://omnetpp.org/.

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

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

[6] M. Al-Fares, S. Radhakrishnan, B. Raghavan, N. Huang, and A. Vahdat. Hedera: Dynamic flow scheduling for data center networks. In *NSDI*, 2010.

[7] M. Alizadeh, T. Edsall, S. Dharmapurikar, R. Vaidyanathan, K. Chu, A. Fingerhut, F. Matus, R. Pan, N. Yadav, G. Varghese, et al. CONGA: Distributed congestion-aware load balancing for datacenters. In *SIGCOMM*, 2014.

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

[9] T. Benson, A. Anand, A. Akella, and M. Zhang. Understanding data center traffic characteristics. *CCR*, 2010.

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

[11] 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 *CoNEXT*. ACM, 2013.

[12] M. Casado, T. Koponen, S. Shenker, and A. Tootoonchian. Fabric: a retrospective on evolving SDN. In *HotSDN*, 2012.

[13] A. R. Curtis, W. Kim, and P. Yalagandula. Mahout: Low-overhead datacenter traffic management using end-host-based elephant detection. In *INFOCOM*, 2011.

[14] A. R. Curtis, J. C. Mogul, J. Tourrilhes, P. Yalagandula, P. Sharma, and S. Banerjee. Devoflow: Scaling flow management for high-performance networks. In *CCR*, 2011.

[15] A. Dixit, P. Prakash, Y. C. Hu, and R. R. Kompella. On the impact of packet spraying in data center networks. In *INFOCOM*, 2013.

[16] S. Ghorbani, B. Godfrey, Y. Ganjali, and A. Firoozshahian. Micro Load Balancing in Data Centers with DRILL. Technical report. web.engr.illinois.edu/~ghorban2/papers/drill.

[17] P. Gill, N. Jain, and N. Nagappan. Understanding network failures in data centers: measurement, analysis, and implications. In *CCR*, 2011.

[18] 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. *Commun. ACM*, 54(3), 2011.

[19] K. He, E. Rozner, K. Agarwal, W. Felter, J. Carter, and A. Akella. Presto: Edge-based load balancing for fast datacenter networks. In *SIGCOMM*, 2015.

[20] S. Kandula, D. Katabi, S. Sinha, and A. Berger. Dynamic load balancing without packet reordering. *CCR*, 2007.

[21] S. Kandula, S. Sengupta, A. Greenberg, P. Patel, and R. Chaiken. The nature of data center traffic: measurements & analysis. In *IMC*, 2009.

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

[23] X. Li and M. J. Freedman. Scaling IP Multicast on Datacenter Topologies. *CoNEXT*, 2013.

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

[25] A. Mekkittikul and N. McKeown. A practical scheduling algorithm to achieve 100% throughput in input-queued switches. In *INFOCOM*, 1998.

[26] M. Mitzenmacher. The power of two choices in randomized load balancing. *IEEE Transactions on Parallel and Distributed Systems*, 12, 2001.

[27] R. Niranjan Mysore, A. Pamboris, N. Farrington, N. Huang, P. Miri, S. Radhakrishnan, V. Subramanya, and A. Vahdat. PortLand: a scalable fault-tolerant layer 2 data center network fabric. *CCR*, 2009.

[28] J. Perry, A. Ousterhout, H. Balakrishnan, D. Shah, and H. Fugal. Fastpass: A centralized zero-queue datacenter network. In *SIGCOMM*, 2014.

[29] J. Rasley, B. Stephens, C. Dixon, E. Rozner, W. Felter, K. Agarwal, J. Carter, and R. Fonseca. Planck: millisecond-scale monitoring and control for commodity networks. In *SIGCOMM*, 2014.