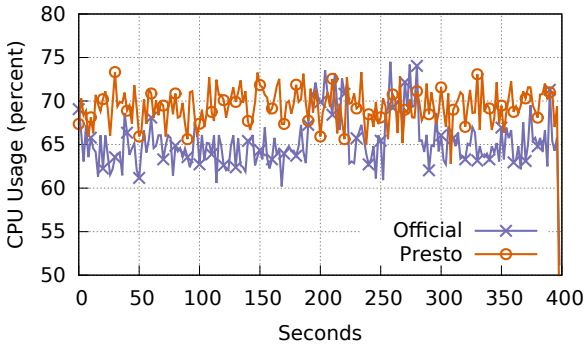


Figure 6: Presto incurs 6% CPU overhead on average.

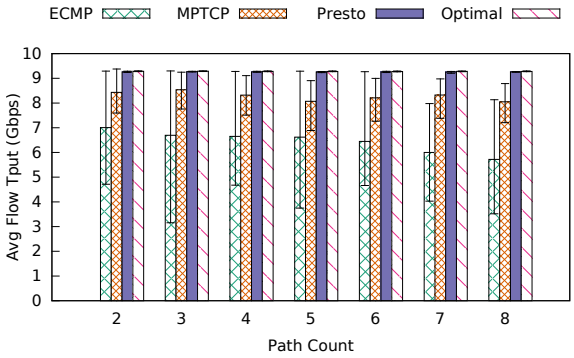


Figure 7: Throughput comparison in scalability benchmark. We denote the non-blocking case as Optimal.

cells between the first packet and last packet of each flowcell. A value of zero means there is no reordering and larger values mean more reordering. The figure shows Presto's GRO can completely mask reordering while official GRO incurs significant reordering. As shown in Section 2, reordering can also cause smaller segments to be pushed up the networking stack, causing significant processing overhead. Figure 5b shows the received TCP segment size distribution. Presto's GRO pushes up large segments, while the official GRO pushes up many small segments. The average TCP throughputs in official GRO and Presto GRO are 4.6 Gbps (with 86% CPU utilization) and 9.3 Gbps (with 69% CPU utilization), respectively. Despite the fact that official GRO only obtains about half the throughput of Presto's GRO, it still incurs more than 24% higher CPU overhead. Therefore, an effective scheme must deal with both reordering and small segment overhead.

Presto Imposes Limited CPU Overhead We investigate Presto's CPU usage by running the stride workload on a 2-tier Clos network as shown in Figure 3. For comparison, official GRO is run with the stride workload using a non-blocking switch (so there is no reordering). Note both official GRO and Presto GRO can achieve 9.3 Gbps. The receiver CPU usage is sampled every 2 seconds over a 400 second interval, and the time-series is shown in Figure 6. On average, Presto GRO only increases CPU usage by 6% compared with the official GRO. The minimal CPU overhead comes from Presto's careful design and implementation. At the sender, Presto needs just two `memcpy` operations (1 for shadow MAC rewriting, 1 for flowcell ID encoding). At the receiver, Presto needs one `memcpy` to rewrite the shadow MAC back to the real MAC and also incurs slight overhead because multiple segments are now kept per flow. The overhead of the latter is reduced because these segments are largely kept in reverse sorted order, which means `merge` on an incoming packet is usually $\mathcal{O}(1)$. The insertion sort is done at the beginning of each `flush` event over a small number of mostly in-order segments, which amortizes overhead because it is called infrequently compared to `merge`.

Presto Scales to Multiple Paths We analyze Presto's ability to scale in the number of paths by setting the number of flows (host pairs) equal to the number of available paths in

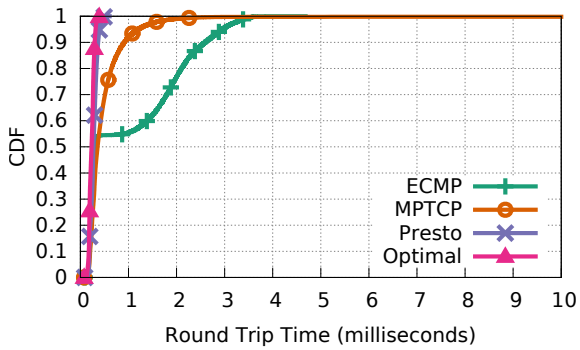
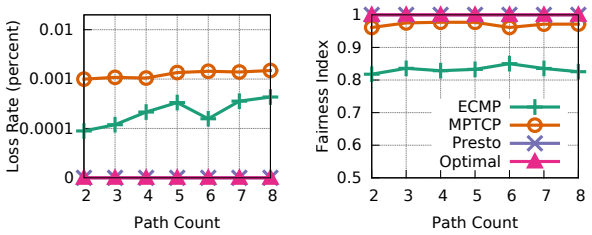


Figure 8: Round trip time comparison in scalability benchmark.

the topology shown in Figure 4a. The number of paths is varied from 2 to 8, and Presto always load-balances over all available paths. Figure 7 shows Presto's throughput closely tracks Optimal. ECMP (and MPTCP) suffer from lower throughput when flows (or subflows) are hashed to the same path. Hashing on the same path leads to congestion and thus increased latency, as shown in Figure 8. Because this topology is non-blocking and Presto load-balances in a near optimal fashion, Presto's latency is near Optimal. Packet drop rates are presented in Figure 9a and show Presto and Optimal have no loss. MPTCP has higher loss because of its bursty nature [4] and its aggression in the face of loss: when a single loss occurs, only one subflow reduces its rate. The other schemes are more conservative because a single loss reduces the rate of the whole flow. Finally, Figure 9b shows Presto, Optimal and MPTCP achieve almost perfect fairness.

Presto Handles Congestion Gracefully Presto's ability to handle congestion is analyzed by fixing the number of spine and leaf switches to 2 and varying the number of flows (host pairs) from 2 to 8, as shown in Figure 4b. Each flow sends as much as possible, which leads to the network being oversubscribed by a ratio of 1 (two flows) to 4 (eight flows). Figure 10 shows all schemes track Optimal in highly oversubscribed environments. ECMP does poorly under moderate congestion because the limited number of flows can be hashed to the same path. Presto does no worse in terms of latency (Figure 11) and loss (Figure 12a). The long tail latency



(a) (b)

Figure 9: (a) Loss rate and (b) Fairness index comparison in scalability benchmark.

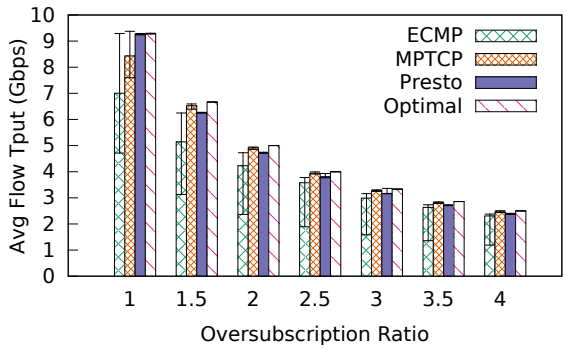


Figure 10: Throughput comparison in oversubscription benchmark.

for MPTCP is caused by its higher loss rates. Both Presto and MPTCP have greatly improved fairness compared with ECMP (Figure 12b).

Comparison to Flowlet Switching We first implemented a flowlet load-balancing scheme in OVS that detects inactivity gaps and then schedules flowlets over disjoint paths in a round robin fashion. The receiver for flowlets uses official GRO. Our flowlet scheme is not a direct reflection of CONGA because (i) it is not congestion-aware and (ii) the flowlets are determined in the software edge instead of the networking hardware. Presto is compared to 500 µs and 100 µs inactivity timers in the stride workload on the 2-tier Clos network (Figure 3). The throughput of the schemes are 9.3 Gbps (Presto), 7.6 Gbps (500 µs), and 4.3 Gbps (100 µs). Analysis of the 100 µs network traces show 13%-29% packets in the connection are reordered, which means 100 µs is not enough time to allow packets to arrive in-order at the destination and thus throughput is severely impacted. Switching flowlets with 500 µs prevents most reordering (only 0.03%-0.5% packets are reordered), but creates very large flowlets (see Figure 1). This means flowlets can still suffer from collisions, which can hurt throughput (note: while not shown here, 500 µs outperforms ECMP by over 40%). Figure 13 shows the latencies. Flowlet 100 µs has low throughput and hence lower latencies. However, since its load balancing isn't perfect, it can still cause increased congestion in the tail. Flowlet 500 µs also has larger tail latencies because of more pronounced flowlet collisions. As compared to the

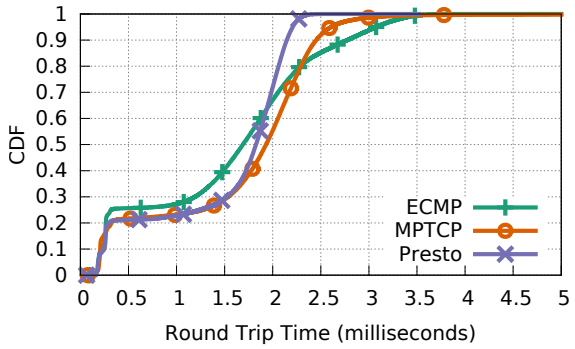
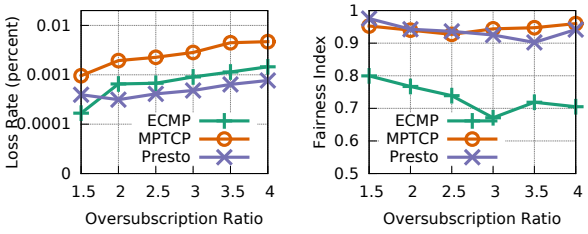


Figure 11: Round trip time comparison in oversubscription benchmark.



(a) (b)

Figure 12: (a) Loss rate and (b) Fairness index comparison in oversubscription benchmark.

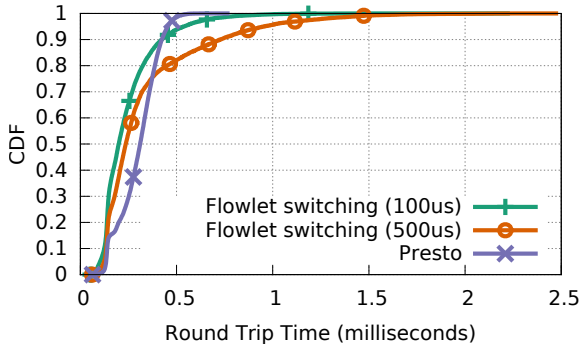


Figure 13: Round trip time comparison of flowlet switching and Presto in Stride workload. The throughputs of Flowlet switching with 100 µs gap, 500 µs gap and Presto are 4.3 Gbps, 7.6 Gbps and 9.3 Gbps respectively.

flowlet schemes, Presto decreases 99.9th percentile latency by 2x-3.6x.

Comparison to Local, Per-Hop Load Balancing Presto sends flowcells in a round robin fashion over pre-configured end-to-end paths. An alternative is to have ECMP hash on flowcell ID and thus provide per-hop load balancing. We compare Presto + shadow MAC with Presto + ECMP using a stride workload on our testbed. Presto + shadow MAC's average throughput is 9.3 Gbps while Presto + ECMP's is 8.9 Gbps. The round trip time CDF is shown in Figure 14. Presto + shadow MAC gives better latency performance compared with Presto + ECMP. The performance difference comes from the fact that Presto + shadow MAC provides better fine-

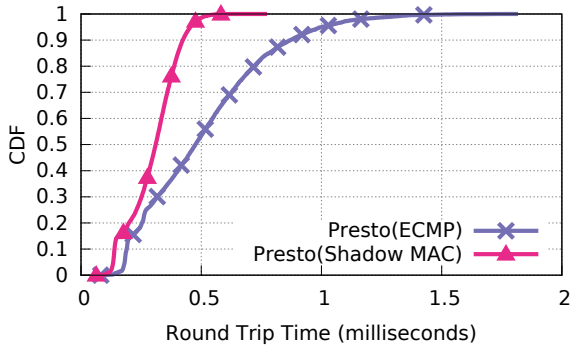


Figure 14: Round trip time comparison between Presto + shadow MAC and Presto + ECMP.

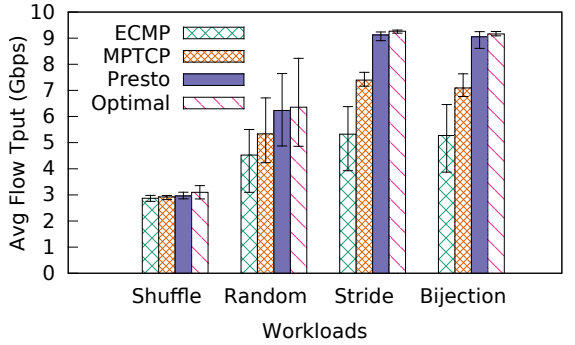


Figure 15: Elephant flow throughput for ECMP, MPTCP, Presto and Optimal in shuffle, random, stride and random bijection workloads.

grained flowcell load balancing because randomization in per-hop multipathing can lead to corner cases where a large fraction of flowcells get sent to the same link over a small timescale by multiple flows. This transient congestion can lead to increased buffer occupancy and higher delays.

6. EVALUATION

In this section, we analyze the performance of Presto for (i) synthetic workloads, (ii) trace-driven workloads, (iii) workloads containing north-south cross traffic, and (iv) failures. All tests are run on the topology in Figure 3.

Synthetic Workloads Figure 15 shows the average throughputs of elephant flows in the shuffle, random, stride and random bijection workloads. Presto's throughput is within 1-4% of Optimal over all workloads. For the shuffle workload, ECMP, MPTCP, Presto and Optimal show similar results because the throughput is mainly bottlenecked at the receiver. In the non-shuffle workloads, Presto improves upon ECMP by 38-72% and improves upon MPTCP by 17-28%.

Figure 16 shows a CDF of the mice flow completion time (FCT) for each workload. The stride and random bijection workloads are non-blocking, and hence the latency of Presto closely tracks Optimal: the 99.9th percentile FCT for Presto is within 350 µs for these workloads. MPTCP and ECMP suffer from congestion, and therefore the tail FCT is much worse than Presto: ECMP's 99.9th percentile FCT is over

Percentile	ECMP	Optimal	Presto
50%	1.0	−12%	−9%
90%	1.0	−34%	−32%
99%	1.0	−63%	−56%
99.9%	1.0	−61%	−60%

Table 1: Mice (<100KB) FCT in trace-driven workload [33]. Negative numbers imply shorter FCT.

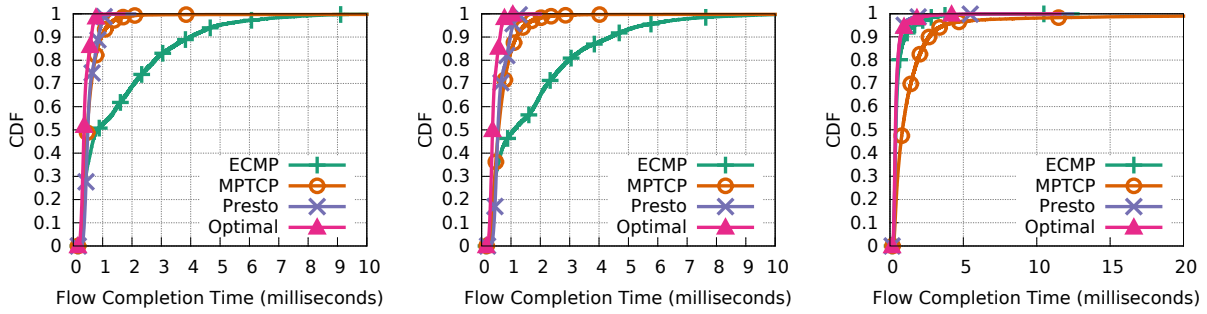
7.5x worse (∼11ms) and MPTCP experiences timeout (because of higher loss rates and the fact that small sub-flow window sizes from small flows can increase the chances of timeout [53]). We used the Linux default timeout (200 ms) and trimmed graphs for clarity. The difference in the random and shuffle workloads is less pronounced (we omit random due to space constraints). In these workloads elephant flows can collide on the last-hop output port, and therefore mice FCT is mainly determined by queuing latency. In shuffle, the 99.9th percentile FCT for ECMP, Presto and Optimal are all within 10% (MPTCP again experiences TCP timeout) and in random, the 99.9th percentile FCT of Presto is within 25% of Optimal while ECMP's is 32% worse than Presto.

Trace-driven Workload We evaluate Presto using a trace-driven workload based on traffic patterns measured in [33]. Each server establishes a long-lived TCP connection with every other server in the testbed. Then each server continuously samples flow sizes and inter-arrival times and each time sends to a random receiver that is not in the same rack. We scale the flow size distribution by a factor of 10 to emulate a heavier workload. Mice flows are defined as flows that are less than 100 KB in size, and elephant flows are defined as flows that are greater than 1 MB. The mice FCT, normalized to ECMP, is shown in Table 1. Compared with ECMP, Presto has similar performance at the 50th percentile but reduces the 99th and 99.9th percentile FCT by 56% and 60%, respectively. Note MPTCP is omitted because its performance was quite unstable in workloads featuring a large number of small flows. The average elephant throughput (not shown) for Presto tracks Optimal (within 2%), and improves upon ECMP by over 10%.

Percentile	ECMP	Optimal	Presto	MPTCP
50%	1.0	−34%	−20%	−12%
90%	1.0	−83%	−79%	−73%
99%	1.0	−89%	−86%	−73%
99.9%	1.0	−91%	−87%	TIMEOUT

Table 2: FCT comparison (normalized to ECMP) with ECMP load balanced north-south traffic. Optimal means all the hosts are attached to a single switch.

Impact of North-South Cross Traffic Presto load balances on "east-west" traffic in the datacenter, *i.e.*, traffic originating and ending at servers in the datacenter. In a real datacenter environment "north-south" traffic (*i.e.*, traffic with an endpoint outside the datacenter) must also be considered. To study the impact of north-south traffic on Presto, we attach



(a) Stride (b) Random Bijection (c) Shuffle

Figure 16: Mice FCT of ECMP, MPTCP, Presto and Optimal in stride, random bijection, and shuffle workloads.

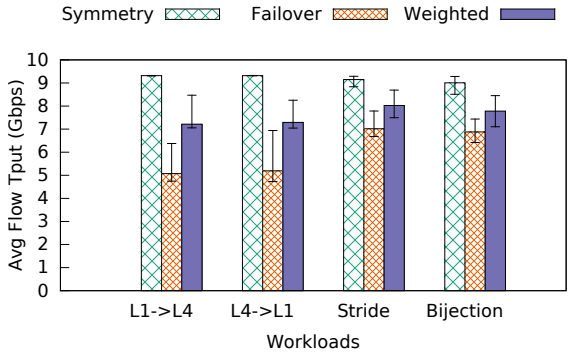


Figure 17: Presto's throughput in symmetry, fast failover and weighted multipathing stages for different workloads.

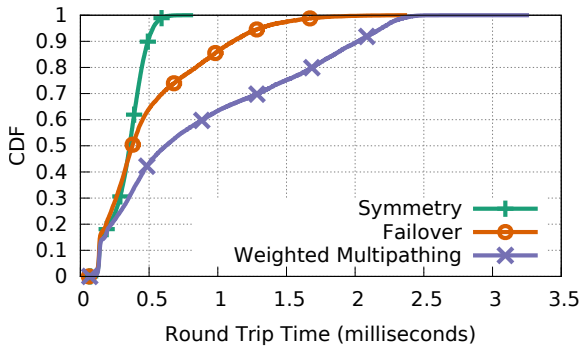


Figure 18: Presto's RTT in symmetry, fast failover and weighted multipathing stages in random bijection workload.

an additional server to each spine switch in our testbed to emulate remote users. The 16 servers establish a long-lived TCP connection with each remote user. Next, each server starts a flow to a random remote user every 1 millisecond. This emulates the behavior of using ECMP to load balance north-south traffic. The flow sizes for north-south traffic are based on the distribution measurement in [29]. The throughput to remote users is limited to 100Mbps to emulate the limitation of an Internet WAN. Along with the north-south flows, a stride workload is started to emulate the east-west traffic. The east-west mice FCT is shown in Table 2 (normalized to ECMP). ECMP, MPTCP, Presto, and Optimal's average throughput is 5.7, 7.4, 8.2, and 8.9Gbps respectively. The experiment shows Presto can gracefully co-exist with north-south cross traffic in the datacenter.

Impact of Link Failure Finally, we study the impact of link failure. Figure 17 compares the throughputs of Presto when the link between spine switch S1 and leaf switch L1 goes down. Three stages are defined: symmetry (the link is up), failover (hardware fast-failover moves traffic from S1 to S2), and weighted (the controller learns of the failure and prunes the tree with the bad link). Workload L1–>L4 is when each node connected to L1 sends to one node in L4 (L4–>L1 is the opposite). Despite the asymmetry in the topology, Presto still achieves reasonable average throughput at each stage. Figure 18 shows the round trip time of each stage in a random bijection workload. Due to the fact that the network is no longer non-blocking after the link failure, failover and weighted multipathing stages have larger round trip time.

7. RELATED WORK

We summarize the related work into three categories: datacenter traffic load balancing, reducing tail latency and handling packet reordering.

Load Balancing in Datacenters MPTCP [53, 61] is a transport protocol that uses subflows to transmit over multiple paths. CONGA [4] and Juniper VCF [28] both employ congestion-aware flowlet switching [58] on specialized switch chipsets to load balance the network. RPS [22] and DRB [17] evaluate per-packet load balancing on symmetric 1 Gbps networks at the switch and end-host, respectively. The CPU load and feasibility of end-host-based per-packet load balancing for 10+ Gbps networks remains open. Hedera [3], MicroTE [12] and Planck [54] use centralized traffic engineering to reroute traffic based on network conditions. FlowBender [32] reroutes flows when congestion is detected by end-hosts and Fastpass [49] employs a centralized arbiter to schedule path selection for each packet. As compared to these schemes, Presto is the only one that proactively load-balances at line rate for fast networks in a near uniform fashion without requiring additional infrastructure or changes to network hardware or transport layers. Furthermore, to the best of our knowledge, Presto is the first work to explore

the interactions of fine-grained load balancing with built-in segment offload capabilities used in fast networks.

Reducing Tail Latency DeTail [63] is a cross-layer network stack designed to reduce the tail of flow completion times. DCTCP [5] is a transport protocol that uses the portion of marked packets by ECN to adaptively adjust sender's TCP's congestion window to reduce switch buffer occupancy. HULL [6] uses Phantom Queues and congestion notifications to cap link utilization and prevent congestion. In contrast, Presto is a load balancing system that naturally improves the tail latencies of mice flows by uniformly spreading traffic in fine-grained units. QJUMP [25] utilizes priority levels to allow latency-sensitive flows to "jump-the-queue" over low priority flows. PIAS [9] uses priority queues to mimic the Shortest Job First principle to reduce FCTs. Last, a blog post by Casado and Pettit [19] summarized four potential ways to deal with elephants and mice, with one advocating to turn elephants into mice at the edge. We share the same motivation and high-level idea and design a complete system that addresses many practical challenges of using such an approach.

Handling Packet Reordering TCP performs poorly in the face of reordering, and thus several studies design a more robust alternative [14, 15, 64]. Presto takes the position that reordering should be handled below TCP in the existing receive offload logic. In the lower portion of the networking stack, SRPIC [62] sorts reordered packets in the driver after each interrupt coalescing event. While this approach can help mitigate the impact of reordering, it does not sort packets across interrupts, have a direct impact on segment sizes, or distinguish between loss and reordering.

8. CONCLUSION

In this paper, we present Presto: a near uniform sub-flow distributed load balancing scheme that can near optimally load balance the network at fast networking speeds. Our scheme makes a few changes to the hypervisor soft-edge (vSwitch and GRO) and does not require any modifications to the transport layer or network hardware, making the bar for deployment lower. Presto is explicitly designed to load balance the network at fine granularities and deal with reordering without imposing much overhead on hosts. Presto is flexible and can also deal with failures and asymmetry. Finally, we show the performance of Presto can closely track that of an optimal non-blocking switch, meaning elephant throughputs remain high while the tail latencies of mice flow completion times do not grow due to congestion.

Acknowledgement

We would like to thank Jon Crowcroft (our shepherd) and the anonymous reviewers for their valuable feedback. This work is supported in part by IBM Corporation, National Science Foundation (grants CNS-1302041, CNS-1330308 and CNS-1345249) and the Wisconsin Institute on Software-Defined Datacenters of Madison.

References

[1] K. Agarwal, C. Dixon, E. Rozner, and J. Carter. Shadow MACs: Scalable Label-switching for Commodity Ethernet. In *HotSDN*, 2014.

[2] M. Al-Fares, A. Loukissas, and A. Vahdat. A Scalable, Commodity Data Center Network Architecture. In *SIGCOMM*, 2008.

[3] M. Al-fares, S. Radhakrishnan, B. Raghavan, N. Huang, and A. Vahdat. Hedera: Dynamic Flow Scheduling for Data Center Networks. In *NSDI*, 2010.

[4] 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.

[5] M. Alizadeh, A. Greenberg, D. A. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta, and M. Sridharan. Data Center TCP (DCTCP). In *SIGCOMM*, 2010.

[6] M. Alizadeh, A. Kabbani, T. Edsall, B. Prabhakar, A. Vahdat, and M. Yasuda. Less Is More: Trading a Little Bandwidth for Ultra-Low Latency in the Data Center. In *NSDI*, 2012.

[7] G. Antichi, C. Callegari, and S. Giordano. Implementation of TCP Large Receive Offload on Open Hardware Platform. In *Proceedings of the First Edition Workshop on High Performance and Programmable Networking*, HPPN '13, 2013.

[8] Arista 7250QX Series 10/40G Data Center Switches. http://www.arista.com/assets/data/pdf/Datasheets/7250QX-64_Datasheet.pdf.

[9] W. Bai, L. Chen, K. Chen, D. Han, C. Tian, and H. Wang. Information-Agnostic Flow Scheduling for Commodity Data Centers. In *NSDI*, 2015.

[10] H. Ballani, P. Costa, C. Gkantsidis, M. P. Grosvenor, T. Karagiannis, L. Koromilas, and G. O'Shea. Enabling End Host Network Functions. In *SIGCOMM*, 2015.

[11] T. Benson, A. Akella, and D. A. Maltz. Network Traffic Characteristics of Data Centers in the Wild. In *IMC*, 2010.

[12] T. Benson, A. Anand, A. Akella, and M. Zhang. MicroTE: Fine Grained Traffic Engineering for Data Centers. In *CoNEXT*, 2011.

[13] C. Benvenuti. *Understanding Linux Network Internals*. " O'Reilly Media, Inc.", 2006.

[14] E. Blanton and M. Allman. On Making TCP More Robust to Packet Reordering. *ACM SIGCOMM Computer Communication Review*, 2002.

[15] S. Bohacek, J. P. Hespanha, J. Lee, C. Lim, and K. Obraczka. TCP-PR: TCP for Persistent Packet Reordering. In *ICDCS*, 2003.

[16] Broadcom Extends Leadership with New StrataXGS Trident II Switch Series Optimized for Cloud-Scale Data Center Networks. http://www.broadcom.com/press/release.php?id=s702418, 2012.

[17] 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*, 2013.

[18] M. Casado, T. Koponen, S. Shenker, and A. Tootoonchian. Fabric: A Retrospective on Evolving SDN. In *HotSDN*, 2012.

[19] M. Casado and J. Pettit. Of Mice and Elephants. http://networkheresy.com/2013/11/01/of-mice-and-elephants/, November 2013.

[20] W. Chou and M. Luo. Agile Network, Agile World: the SDN Approach. http://huaweiempresas.com/img_eventos/140314/SDN_Workshop_2.pdf.

[21] S. Das. Smart-Table Technology—Enabling Very Large Server, Storage Nodes and Virtual Machines to Scale Using

Flexible Network Infrastructure Topologies. http://www.broadcom.com/collateral/wp/StrataXGS_SmartSwitch-WP101-R.pdf, July 2012.
[22] A. Dixit, P. Prakash, Y. C. Hu, and R. R. Kompella. On the Impact of Packet Spraying in Data Center Networks. In *INFOCOM*, 2013.
[23] 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 *SIGCOMM*, 2009.
[24] L. Grossman. Large Receive Offload Implementation in Neterion 10GbE Ethernet Driver. In *Linux Symposium*, 2005.
[25] M. P. Grosvenor, M. Schwarzkopf, I. Gog, R. N. Watson, A. W. Moore, S. Hand, and J. Crowcroft. Queues Don't Matter When You Can JUMP Them! In *NSDI*, 2015.
[26] H. Chen and W. Song. Load Balancing without Packet Reordering in NVO3. Internet-Draft. https://tools.ietf.org/html/draft-chen-nvo3-load-banlancing-00, October 2014.
[27] S. Ha, I. Rhee, and L. Xu. CUBIC: A New TCP-friendly High-speed TCP Variant. *SIGOPS Oper. Syst. Rev.*, 2008.
[28] D. R. Hanks. *Juniper QFX5100 Series: A Comprehensive Guide to Building Next-Generation Networks*. " O'Reilly Media, Inc.", 2014.
[29] K. He, A. Fisher, L. Wang, A. Gember, A. Akella, and T. Ristenpart. Next Stop, the Cloud: Understanding Modern Web Service Deployment in EC2 and Azure. In *IMC*, 2013.
[30] R. Jain, D. Chiu, and W. Hawe. A Quantitative Measure Of Fairness And Discrimination For Resource Allocation In Shared Computer Systems. *CoRR*, 1998.
[31] JLS2009: Generic Receive Offload. https://lwn.net/Articles/358910/.
[32] A. Kabbani, B. Vamanan, J. Hasan, and F. Duchene. FlowBender: Flow-level Adaptive Routing for Improved Latency and Throughput in Datacenter Networks. In *CoNEXT*, 2014.
[33] S. Kandula, S. Sengupta, A. Greenberg, P. Patel, and R. Chaiken. The Nature of Data Center Traffic: Measurements & Analysis. In *IMC*, 2009.
[34] R. Kapoor, A. C. Snoeren, G. M. Voelker, and G. Porter. Bullet Trains: A Study of NIC Burst Behavior at Microsecond Timescales. In *CoNEXT*, 2013.
[35] R. Khalili, N. Gast, M. Popovic, U. Upadhyay, and J.-Y. Le Boudec. MPTCP is Not Pareto-optimal: Performance Issues and a Possible Solution. In *CoNEXT*, 2012.
[36] T. Koponen, K. Amidon, P. Balland, M. Casado, A. Chanda, B. Fulton, I. Ganichev, J. Gross, N. Gude, P. Ingram, et al. Network Virtualization in Multi-tenant Datacenters. In *NSDI*, 2014.
[37] K.-C. Leung, V. O. Li, and D. Yang. An Overview of Packet Reordering in Transmission Control Protocol (TCP): Problems, Solutions, and Challenges. *Parallel and Distributed Systems, IEEE Transactions on*, 2007.
[38] Linux Kernel. https://www.kernel.org.
[39] A. Menon and W. Zwaenepoel. Optimizing TCP Receive Performance. In *USENIX Annual Technical Conference*, 2008.
[40] J. C. Mogul and K. Ramakrishnan. Eliminating Receive Livelock in An Interrupt-driven Kernel. *ACM Transactions on Computer Systems*, 1997.
[41] T. P. Morgan. A Rare Peek Into The Massive Scale of AWS. http://www.enterprisetech.com/2014/11/14/rare-peek-massive-scale-aws/, November 2014.
[42] MPTCP Linux Kernel Implementation. http://www.multipath-tcp.org.
[43] J. Mudigonda, P. Yalagandula, J. Mogul, B. Stiekes, and Y. Pouffary. NetLord: A Scalable Multi-tenant Network Architecture for Virtualized Datacenters. In *SIGCOMM*, 2011.
[44] Nuttcp. http://www.nuttcp.net/Welcome%20Page.html.
[45] Open vSwitch. http://openvswitch.org.
[46] C. Paasch, R. Khalili, and O. Bonaventure. On the Benefits of Applying Experimental Design to Improve Multipath TCP. In *CoNEXT*, 2013.
[47] V. Paxson. End-to-end Internet Packet Dynamics. In *ACM SIGCOMM Computer Communication Review*, 1997.
[48] I. Pepelnjak. NEC+IBM: Enterprise OpenFlow You can Actually Touch. http://blog.ipspace.net/2012/02/necibm-enterprise-openflow-you-can.html.
[49] J. Perry, A. Ousterhout, H. Balakrishnan, D. Shah, and H. Fugal. Fastpass: A Centralized "Zero-Queue" Datacenter Network. In *SIGCOMM*, 2014.
[50] J. Pettit. Open vSwitch and the Intelligent Edge. In *OpenStack Summit*, 2014.
[51] B. Pfaff, J. Pettit, K. Amidon, M. Casado, T. Koponen, and S. Shenker. Extending Networking into the Virtualization Layer. In *HotNets*, 2009.
[52] B. Pfaff, J. Pettit, T. Koponen, E. J. Jackson, A. Zhou, J. Rajahalme, J. Gross, A. Wang, J. Stringer, P. Shelar, et al. The Design and Implementation of Open vSwitch. In *NSDI*, 2015.
[53] C. Raiciu, S. Barre, C. Pluntke, A. Greenhalgh, D. Wischik, and M. Handley. Improving Datacenter Performance and Robustness with Multipath TCP. In *SIGCOMM*, 2011.
[54] 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.
[55] Receive Segment Coalescing (RSC). http://technet.microsoft.com/en-us/library/hh997024.aspx.
[56] Receive Side Scaling (RSS). https://technet.microsoft.com/en-us/library/hh997036.aspx.
[57] M. Scott, A. Moore, and J. Crowcroft. Addressing the Scalability of Ethernet with MOOSE. In *Proc. DC CAVES Workshop*, 2009.
[58] S. Sinha, S. Kandula, and D. Katabi. Harnessing TCPs Burstiness using Flowlet Switching. In *HotNets*, 2004.
[59] Sockperf: A Network Benchmarking Utility over Socket API. https://code.google.com/p/sockperf/.
[60] VXLAN Performance Evaluation on VMware vSphere 5.1. http://www.vmware.com/files/pdf/techpaper/VMware-vSphere-VXLAN-Perf.pdf.
[61] D. Wischik, C. Raiciu, A. Greenhalgh, and M. Handley. Design, Implementation and Evaluation of Congestion Control for Multipath TCP. In *NSDI*, 2011.
[62] W. Wu, P. Demar, and M. Crawford. Sorting Reordered Packets with Interrupt Coalescing. *Computer Networks*, 2009.
[63] D. Zats, T. Das, P. Mohan, D. Borthakur, and R. Katz. DeTail: Reducing the Flow Completion Time Tail in Datacenter Networks. In *SIGCOMM*, 2012.
[64] M. Zhang, B. Karp, S. Floyd, and L. Peterson. RR-TCP: A Reordering-robust TCP with DSACK. In *ICNP*, 2003.
[65] J. Zhou, M. Tewari, M. Zhu, A. Kabbani, L. Poutievski, A. Singh, and A. Vahdat. WCMP: Weighted Cost Multipathing for Improved Fairness in Data Centers. In *EuroSys*, 2014.