CLOVE: How I Learned to Stop Worrying About the Core and Love the Edge

Naga Katta[1,2], Mukesh Hira[2], Aditi Ghag[2], Changhoon Kim[3],
Isaac Keslassy[2,4], Jennifer Rexford[1]

[1]Princeton University, [2]VMware, [3]Barefoot Networks, [4]Technion

Abstract

Multi-tenant datacenters predominantly use equal-cost multipath (ECMP) routing to distribute traffic over multiple network paths. However, ECMP static hashing causes unequal load-balancing and collisions, leading to low throughput and high latencies. Recently proposed alternatives for load-balancing perform better, but are impractical as they require either changing the tenant VM network stacks (*e.g.,* MPTCP) or replacing all the network switches (*e.g.,* CONGA).

In this paper, we argue that the end-host hypervisor provides a sweet spot for implementing a spectrum of load-balancing algorithms that are fine-grained, congestion-aware, and reactive to network dynamics at round-trip timescales. We propose CLOVE, a scalable hypervisor-based load-balancer that requires no changes to guest VMs or to physical network switches. CLOVE uses standard ECMP in the physical network, learns about equal-cost network paths using a traceroute mechanism, and learns about congestion state along these paths using standard switch features such as ECN. It then manipulates packet header fields in the hypervisor virtual switch to route traffic over less congested paths. We introduce different variants of CLOVE that differ in the way they learn about congestion in the physical network. Using extensive simulations, we show that CLOVE captures some 80% of the performance gain of best-of-breed hardware-based load-balancing algorithms without the need for expensive hardware replacement.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

HotNets-XV, November 09-10, 2016, Atlanta, GA, USA

© 2016 ACM. ISBN 978-1-4503-4661-0/16/11...$15.00

DOI: http://dx.doi.org/10.1145/3005745.3005751

1 Introduction

Today, multitenant datacenter operators aggressively scale their networks to accommodate new tenants and applications. They build large multi-rooted leaf-spine or fat-tree topologies with multiple paths to provide high bisection bandwidth. They also deploy sophisticated software-defined-networking (SDN) control planes to manage the fast-growing number of tenants.

Surprisingly, despite the increasing scale and complexity, multitenant datacenter operators still rely on a fossilized routing mechanism to load-balance traffic. Flows are routed using equal-cost multi-path (ECMP), which uses static hash functions to assign network flows to paths. Unfortunately, in ECMP, hash collisions between large flows can often cause applications to experience low throughput and/or high latency, while there may be significant unutilized capacity to route the colliding flows on different paths.

In spite of its well-known flaws, ECMP continues to be widely deployed in datacenter networks due to its simplicity. The current literature has not provided multitenant datacenter operators with a viable alternative that they could readily try without a potentially-costly leap of faith. Network load-balancing algorithms that address the shortcomings of ECMP can be broadly divided into three categories.

Centralized Scheduling. Several solutions use a centralized controller for load balancing (Hedera, MicroTE, SWAN, Fastpass [1–4]). However, they are prohibitively slow to react to latency-sensitive short flows.

Host-based Load Balancing. A second category of solutions is based on changing the networking stack of each tenant virtual machine (VM) by implementing a variant of TCP called Multipath TCP (MPTCP) [5] that splits each application flow into multiple TCP subflows. These TCP subflows have distinct five-tuples and hence are routed independently by ECMP in the network. MPTCP has a number of important shortcomings. Firstly, it is challenging to deploy in multi-tenant environments in which operators do not control the end-host networking stack. Secondly, MPTCP is shown to perform poorly in incast scenarios [6]. The TCP subflows use independent congestion windows, which results in large bursts of data and aggravated pressure on the last-hop link in incast scenarios. Thirdly, MPTCP carries sequence number offsets in TCP options, which means that middleboxes that need to establish context for packets belonging to a flow have to understand these TCP options and tie together different TCP subflows into an application flow.

In-Network Per-Hop Load Balancing The third category of solutions replaces ECMP with utilization-aware routing at each network switch (CONGA, HULA, DRB, FLARE, LocalFlow, DRILL, EAR, DiFS [6–13]), or needs to change the end-host stacks in addition to changing the switches (DARD, FlowBender [14, 15]). This requires every single network switch to be upgraded and comes at a high capital and operational cost.

Instead, we believe that in virtualized multitenant datacenters, the virtual switch at each hypervisor provides a unique opportunity to exploit multiple equal-cost paths without requiring any special capability in the physical network or any changes in the guest VMs' TCP/IP stacks. This has been previously explored in Presto [16], which divides a flow into large flowcells, and spreads flowcells across multiple paths. However, Presto has two main shortcomings. Firstly, it is based on MAC address based forwarding in the network and uses shadow MAC addresses to route flowcells over multiple Spanning Trees. In reality, most data center networks use ECMP forwarding based on IP addresses. Secondly, Presto cannot adapt to congestion caused by network asymmetry unless it relies on a centralized controller.

CLOVE. In this paper, we present CLOVE (***C**ongestion-aware **LO**ad-balancing from the **V**irtual **E**dge*), an adaptive and scalable hypervisor-based load-balancing solution. Implemented entirely in the virtual switches of hypervisors, CLOVE effectively virtualizes load-balancing. It uses standard ECMP in the physical network, and can be deployed in any environment regardless of the guest VM TCP/IP stack and the underlying physical infrastructure.

CLOVE is based on the key observation that since ECMP relies on static hashing, the virtual switch at the source hypervisor can change the packet header to directly dictate the path that each packet takes in the ECMP-based physical network. CLOVE relies on three important components:

(1) Path discovery. First, CLOVE uses the virtual switch in the hypervisor to control packet routing. To do so, we assume at first that the datacenter is based on a network overlay (*e.g.,* STT, VxLAN, NV-GRE, GENEVE [17]), and later discuss non-overlay networks. In such an ECMP-based overlay network, by sending many probes with varying source ports in the probe encapsulation headers, the source hypervisor can discover a subset of source ports per destination, that lead to distinct paths to the destination. The source hypervisor can then choose the encapsulation header source port for each outgoing packet such that the packet takes the desired path in the network.

(2) Flowlets. The second component of CLOVE is its reliance on edge-based flowlet-switching. Since CLOVE needs to be able to load-balance ongoing flows while avoiding out-of-order packets, it divides these flows into flowlets, *i.e.,* groups of packets in a flow separated by a sufficiently large idle gap such that the groups of packets can be routed on independent paths with very low probability of arriving out of order at the destination.

(3) Congestion-aware routing. Finally, the last component of CLOVE is an algorithm for reacting to congestion or asymmetry in the network by increasing the probability of picking uncongested paths for new flowlets. CLOVE schedules new flowlets onto different paths by rotating through the source ports in a weighted round-robin fashion. The scheduling weights are continuously adjusted in response to congestion. We consider two variants of CLOVE that differ in how they learn about the real-time state of the network. The first variant, denoted *CLOVE-ECN*, learns about the path congestion states using Explicit Congestion Notification (ECN), and forwards new flowlets on uncongested paths. The second variant, called *CLOVE-INT*, learns about the exact path utilization using In-band Network Telemetry (INT) [18], a technology likely to be supported by datacenter network switches in the near future, and proactively routes new flowlets on the least utilized path.

Experiments. Finally, we run packet-level simulations to see how our edge-based load balancing schemes fare compared to ECMP and to advanced hardware load-balancing schemes like CONGA. We measure the average flow completion times [19] of an empirical web search workload on a leaf-spine topology and conclude that (i) CLOVE-ECN does 3× better than ECMP at 70% network load on an asymmetric network. (ii) CLOVE-ECN comes close to CONGA in terms of average and 99th-percentile flow completion time—it captures 80% of the performance gained by CONGA without needing the expensive hardware. (iii) If CLOVE were to use INT support (with small additional cost on switches) to learn path utilization values, then CLOVE-INT comes 95% close to CONGA's performance. Overall, we illustrate that there are several edge-based load-balancing schemes that can be built in the end-host hypervisor and attain strong load-balancing performance without the limitations of existing schemes.

2 CLOVE Design

In this section, we describe the three important components of CLOVE—the first virtualized, congestion-aware dataplane load-balancer for datacenters.

2.1 Path Discovery using Traceroute

With overlays, the source hypervisor encapsulates packets received from a VM in an overlay encapsulation header. The overlay header utilizes its own set of transport-layer protocol fields that can be manipulated without disrupting the actual inner VM traffic. Our goal is to be able to implement *v*irtualized source routing, *i.e.,* to enable the hypervisor to dictate the path followed by each packet in the datacenter.

We do not assume the existence of an SDN controller or of any general switch functionality. We exploit the widely available ECMP implementations in off-the-shelf commodity switches to build and control the multiple network paths from the hypervisor. Specifically, given a form of overlay encapsulation, CLOVE aims to discover the multiple ECMP paths between any given tunnel end-points used in the outer IP header. In practice, implementations of ECMP are proprietary, and therefore the ECMP hash functions used in the switches are unknown, which makes the problem of mapping multiple network paths challenging.

Path discovery. Inspired by Paris traceroute [20], the source virtual switch discovers a set of equal-cost paths to a destination hypervisor by sending multiple probes where the transport protocol source port of the encapsulation header is randomized, so that the probes travel on different paths using ECMP. The rest of the five-tuple is fixed: the source and destination IP addresses are those of the source and destination hypervisors, the transport protocol and its destination port number are dictated by the encapsulation protocol in use.

Each path discovery probe consists of multiple packets with the same transport protocol source port but with the TTL incremented. This gives the list of IP addresses of the switch interfaces at each hop on that path. The result of the probing is a per-destination set of encapsulation header transport protocol source ports that map to distinct paths to the destination. As an optimization, paths may be discovered only to the subset of hypervisors that have active traffic being forwarded to them from the source hypervisor.

We want CLOVE to rely on a final set of k source ports leading to k distinct (ideally disjoint) paths. To pick these k paths, we simply use a heuristic whereby we greedily add the path that shares the least number of links with the existing picked paths.

Finally, in order to adapt to the changes and failures in the network topology, the path discovery daemon sends periodic probes to be up to date with the latest set of paths to each destination hypervisor. Probing is done on the order of hundreds of milliseconds so that the probe traffic does not overwhelm the available bandwidth at any point in time. Probes to different destination hypervisors may be staggered over this interval.

2.2 Detecting and Routing Flowlets

Having mapped a set of k source ports onto k distinct paths, CLOVE needs to be able to let the new packets of a flow abandon a congested path and instead adopt an uncongested one. Unfortunately, CLOVE cannot simply send each new packet on an arbitrary path, as it would lead to a potentially high packet reordering.

Flowlet table

Flowlet ID	SPort
45	50001
234	50002
505	50003
818	50004

Path weight table

Dst	SPort	Wt
H2	50001	0.1
H2	50002	0.3
H2	50003	0.3
H2	50004	0.3

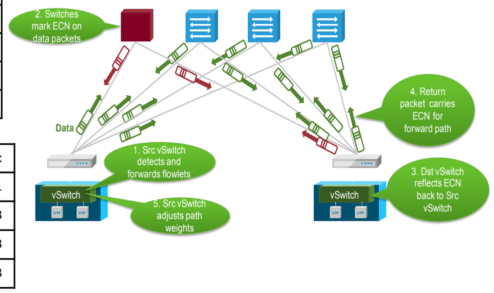

2. Switches mark ECN on data packets
Data
4. Return packet carries ECN for forward path
1. Src vSwitch detects and forwards flowlets
vSwitch
VM
VM
5. Src vSwitch adjusts path weights
vSwitch
VM
VM
3. Dst vSwitch reflects ECN back to Src vSwitch

Figure 1: CLOVE-ECN congestion-aware routing. Upon congestion on the first path, the ECN bit of a data packet is marked to reflect congestion (steps 1-2), and the information is reflected back to the source (steps 3-4). Finally, the path weights are adjusted so that future flowlets are more inclined to take other paths (step 5).

Instead, CLOVE divides each flow into flowlets, *i.e.,* tiny groups of packets in a flow separated by a sufficient idle gap so that when they are routed on distinct paths, the probability that they are received out of order at the receiver is very low. Flowlet splitting is a well-known idea that has often been implemented at the level of switches and routers (*e.g.,* in FLARE [9] and in Cisco's ACI fabric [21]). CLOVE implements it only at the end-points.

Edge-Flowlet. In this paper, we denote by *Edge-Flowlet* the basic algorithm that simply combines the above two techniques, *i.e.,* first generates source ports for distinct paths, and then decomposes flows into flowlets and *randomly* picks a new path for each new flowlet. (As presented below in Section 2.3, the difference between Edge-Flowlet and CLOVE is that CLOVE directly relies on a congestion-avoidance mechanism to pick a new path instead of picking it randomly.)

Even though Edge-Flowlet is a simple flowlet-based ECMP-like algorithm, it turns out that it indirectly reacts to congestion. Specifically, note that in a flow, the inter-packet gap that triggers a new flowlet can be due to two main reasons. First, the application may simply not have something to send. Second, and more importantly, the packets of the previous flowlet may have adopted a congested path, and as a result the TCP ACKs take time to come back and no new packets are sent for a while. In such a case, the new flowlet is in fact a sign of congestion. Thus, the Edge-Flowlet algorithm is expected to perform better than flow-based ECMP, because it can react to congestion on a given path by generating a new flowlet which may be sent on an uncongested path.

2.3 Congestion-Aware Load Balancing

We presented above how CLOVE can (1) establish a set of distinct paths, and (2) use flowlets to be able to change paths. We now describe how (3) CLOVE can react to congestion by preferentially sending new flowlets on uncongested paths.

Routing based on ECN feedback (CLOVE-ECN): As illustrated in Figure 1, our first CLOVE algorithm, denoted *CLOVE-ECN*, relies on the Explicit Congestion Notification (ECN) mechanism. We assume that network switches implement ECN and mark packets as congested whenever the average queue length exceeds some threshold. We first present how CLOVE-ECN collects congestion information, and then explain how it uses it.

Recognizing congestion. The CLOVE-ECN mechanism masks the ECN markings from the sender and receiver VMs, and only uses the congestion information between the sending and receiving hypervisors. The sending hypervisor sets ECN Capable Transport (ECT) bits in the encapsulation IP header. The receiving hypervisor intercepts ECN information and relays it back to the sending hypervisor. It also needs to indicate for which source port (*i.e.,* on which path) the congestion happened. The destination hypervisor uses reserved bits in the encapsulation header of reverse traffic (towards the source) to encode the source-port value that experienced congestion in the forward direction (*e.g.,* the Context field [22] in STT header may be used for this purpose).

Load balancing. CLOVE-ECN uses weighted round robin (WRR) to load balance flowlets on network paths. The weights associated with the distinct paths are continuously adapted based on the congestion feedback obtained from ECN messages. Every time ECN is seen on a certain path, the weight of that path is reduced by a certain factor (we used a factor of $1/3$ to quickly react to continuous ECN markings without being aggressive to those caused by occasional bursts). The weight remainder is then spread equally across all the other uncongested paths. Once the weights are readjusted, the WRR simply rotates through the ports that map to distinct paths (for each new flowlet) according to the new set of weights.

As an optimization, instead of relaying the ECN information on every packet back to the sender, the receiver could relay ECN only once every few RTTs for any given path. The effect of this is that there will be fewer ECNs being relayed and some may be missed entirely. However, this leads to a more calibrated response to the ECN bits (as opposed to unnecessarily aggressive manipulation of path weights) and also amortizes the cost (number of software cycles spent) for processing each packet in the dataplane.

CLOVE-ECN uses three important parameters:

Flowlet time-gap: This is the inter-packet time gap between subsequent packets of a flow that triggers the creation of a new flowlet [9]. Based on previous work [6, 23], we recommend twice the network round trip-time as the flowlet gap for optimal performance.

ECN threshold: This is the threshold in terms of queue length on a switch-port beyond which switches start marking the packets with ECN. Similar to the recommendations by DCTCP [24], we use a threshold of 20 MTU-sized packets so that the load balancer keeps the queues low, and at the same time allows room for TSO-based bursts at high bandwidth.

ECN relay frequency: This is the frequency at which the receiver relays congestion marking to the sender. The receiver should send feedback more frequently than the frequency at which load balancing decisions are being made, as recommended in TexCP [23]. We use half the RTT as the ECN relay frequency in our design.

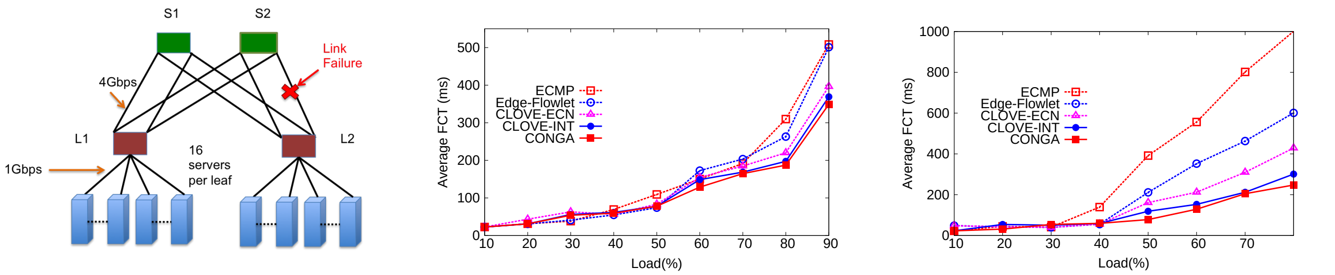
Routing based on in-band network state (CLOVE-INT): Technologies such as In-band Network Telemetry (INT) in next-generation network switches [18] will enable the source virtual switch to embed INT instructions in data packets and probes and collect real-time network state at line-rate. For instance, a source hypervisor could request each network element to insert egress link utilization in the packet headers. When the CLOVE probe is received at the destination hypervisor, it would therefore have real-time link utilization of each link, which may be sent back to the source hypervisor. The source virtual switch can thus make proactive routing decisions to route flowlets on the least utilized paths. Note that while this requires new capability at each switch and hence a physical network upgrade, this approach may be used when INT becomes a standard feature in switches for collection of network state. With CLOVE-INT, the intelligence of calculating end-to-end path utilization and path selection still resides in software in the hypervisor virtual switch. Algorithms such as CONGA on the other hand implement a proprietary state propagation and path calculation algorithm at each switch and require all switches to have the proprietary implementation.

3 Evaluation

In this section, using packet-level simulations in NS2 [25], we illustrate the effectiveness of our various edge-based load balancers.

Algorithms: We compare our three edge-based load-balancing schemes (*Edge-Flowlet*, *CLOVE-ECN*, and *CLOVE-INT*) against the following two extremes of the spectrum of load-balancing schemes: *ECMP*, which uses static hashing and is congestion-oblivious; and *CONGA* [6], which modifies switches to collect switch-based measurements and immediately change the switch-based routing, and therefore is considered the higher end of the spectrum.

Topology: As shown in Figure 2a, we use a 2-tier Clos topology with two spines (S1 and S2) connecting two leaf switches (L1 and L2). Each leaf switch is connected to either spine by two 4Gbps links, yielding a total bisection bandwidth of 16Gbps. Each leaf is connected to 16 servers with 1G links. Therefore, the network is not oversubscribed, and the 16 servers on one leaf can saturate the 16Gbps bandwidth available for traffic across the two leaves. Each packet can take up to four disjoint paths to travel from one leaf to another. Later, in order to simulate asymmetry in the base-



(a) Topology used in evaluation

(b) Symmetric topology - avg FCT

(c) Asymmetric topology - avg FCT

Figure 2: Average FCT for the web-search workload on a network testbed. CLOVE-ECN, which is implementable on existing networks, captures about 80% of the performance gain between ECMP and CONGA in both topologies.

line symmetric topology, we disable one of the 4Gbps links connecting the spine *S*2 with the leaf switch *L*2.

Empirical workload: To generate traffic for our experiments, we use a realistic web search workload [24] obtained from the production datacenters of Microsoft. The workload is long-tailed. Most of its flows are small, and the small fraction of large flows contributes to a substantial portion of the traffic. We simulate a simple client-server communication model where each client connected to L1 chooses a server connected to L2 at random and initiates persistent TCP connections to the server. The client sends a flow with size drawn from the empirical CDF of the web search workload. The inter-arrival rate of the flows on a connection is also taken from an exponential distribution whose mean is tuned to the desired load on the network. Similarly to previous work [6], we look at the average flow completion time (FCT) [19] as the overall performance metric so that all flows including the majority of small flows are given equal consideration. We run each experiment with a total job count of 10K with three random seeds and then measure the average of the three runs.

3.1 Symmetric Topology

First, we compare the various load-balancing schemes on the baseline symmetric topology to make sure that CLOVE-ECN performs at least as well as ECMP.

Figure 2b shows the average flow completion time for all flows as the load on the network increases. At lower loads, the performance of all the load-balancing schemes is nearly the same, because when there is enough bandwidth available in the network, there is a greater tolerance for congestion-oblivious path forwarding. At higher loads, CLOVE-ECN performs better than ECMP or Edge-Flowlet, but underperforms CLOVE-INT and CONGA. ECMP performs the worst because it makes congestion oblivious load balancing at a very coarse granularity. Edge-Flowlet does slightly better because it still does congestion-oblivious load balancing but at the granularity of flowlets. CLOVE-ECN does better than both because of its fine-grained congestion-aware load balancing. CLOVE-ECN achieves 1.4x lower FCT (better performance) compared to ECMP and 1.2x better compared to Edge-Flowlet at 80% network load. However, CLOVE-INT and CONGA do slightly better (by 1.1X) because they are utilization-aware instead of just being congestion-aware. Therefore, CLOVE-ECN, which is implementable on existing networks, captures 82% of the performance gain between ECMP and CONGA at 80% load.

3.2 Topology Asymmetry

When a 4G link between the spine switch *S*2 and switch *L*2 is removed, the effective bandwidth of the network drops by 25% for traffic going across the pods. This means that the load balancing schemes have to carefully balance paths at even lower network loads compared to the baseline topology scenario. In particular, the load balancing scheme has to make sure that the bottleneck link connecting *S*2 to *L*2 is not overwhelmed with a disproportionate amount of traffic.

Figure 2c shows how various schemes perform with the web search workload as the network load is varied. As expected, the overall FCT for ECMP shoots up pretty quickly after 50% network load. Once the network load reaches 50%, the bottleneck link gets pressurized by the flows hashed to go through *S*2. In fact, had we used an infinite-time workload, ECMP would not have theoretically converged. But since we used a finite workload as in [6] to measure the FCT, we obtain finite delays. Edge-Flowlet does slightly better than ECMP because it still does congestion-oblivious load balancing but at the granularity of flowlets. In particular, flows sent on congested paths see more flowlets being created due to the delay caused by queue growth on the bottleneck link, and their new flowlets are therefore more likely to be load-balanced to a different path.

CLOVE-ECN does better than ECMP and Edge-Flowlet because of its fast congestion-aware path selection, which decreases pressure on the bottleneck link once the queues start growing. This helps CLOVE-ECN achieve 3x better performance than ECMP and 1.8x better FCT than Edge-Flowlet at 70% network load. However, it still has to catch up with CLOVE-INT and CONGA, which do 1.2X better than CLOVE-ECN. The important take-away

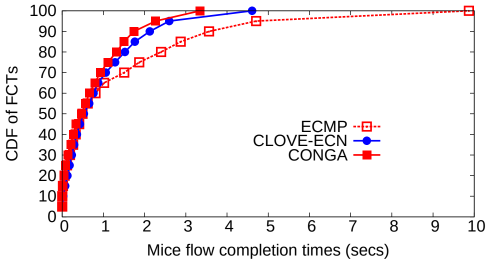


Figure 3: CDF of FCTs at 70% load with asymmetry. CLOVE-ECN captures 80% of the performance gain between the 99th percentiles of ECMP and CONGA.

is that CLOVE-ECN, which is implementable on existing networks, captures 80% of the performance gain between ECMP and CONGA at 70% network load.

99th Percentile. Figure 3 illustrates similar results by plotting the CDFs for the flow completion times of mice flows (of size < 100KB) for the asymmetric topology at 70% load. The 99th percentile FCT for CLOVE-ECN captures 80% of the performance gain between the 99th percentiles of ECMP and CONGA.

CLOVE-ECN vs. CONGA. The main difference between the performance of CLOVE-ECN and CONGA comes from the fact that while CONGA is network utilization-aware, CLOVE-ECN is only congestion-aware. In other words, CLOVE-ECN will deflect flowlets from a path only when its queues start growing beyond the ECN threshold. This means that the flowlets will be sent on paths which are preferred till they reach 100% utilization and beyond. On the other hand, CONGA ensures that the utilization on all paths in the network stays nearly the same. This keeps the queues on the bottleneck paths near zero at all times unless the traffic load exceeds the total network capacity. The results also show that if CLOVE were to potentially use a feature like INT to learn utilization at the edge, then CLOVE-INT captures 95% of CONGA's performance. Therefore, empirically, it is clear that it helps to be utilization-aware in order to make the best load balancing decision, whether it is inside the network or at the edge. However, by just being congestion-aware (which is what is possible with existing switches), CLOVE-ECN still manages to come very close to the performance of CONGA.

4 Discussion

In this section, we address potential deployment concerns and areas of future improvement

Stability: A major concern with adaptive routing schemes is that of route flapping and instability. However, recent efforts like CONGA [6] and HULA [7] have demonstrated that as long as network state is collected at fine-grained timescales, and processed in the dataplane, the resulting scheme is stable in practice. CLOVE similarly collects and acts on network state directly in the datataplane, and makes routing decisions in the virtual switch based on state that is as close to real-time as possible.

Scalability: CLOVE is highly scalable due to its distributed nature. (a) *State space*: Each hypervisor keeps state for k network paths to N destinations. The amount of state is not a concern for software implementations in x86 CPUs even in the largest datacenter networks, with k typically between 4 and 256 and N of the order of thousands. (b) *Network load:* In CLOVE-ECN, congestion indications are collected using data traffic. Periodic probes can detect an (infrequent) change in network topology. Today, a virtual switch in an overlay network typically generates Bi-directional Forwarding Detection (BFD) probes to all overlay destinations, at the timescale of a few hundred ms. Therefore, if CLOVE probes are sent with varying source ports but every few seconds, the overall load should be similar. (c) *Locks:* Another important aspect in implementing CLOVE is that of updating network state. In a multi-core multi-threaded environment, this has to be done using efficient locking mechanisms such as RCU locks to minimize blocking of threads when updating state — a mechanism already used for updating per-connection state in the Open vSwitch today.

CLOVE vs. MPTCP: While CLOVE uses several source ports to load balance traffic like MPTCP, it differs from MPTCP in three major ways: (a) It does not need to change the VM networking stacks; (b) It is less vulnerable to incast because it does not have multiple sub-flows that contend with each other in an incast bottleneck link; and (c) It keeps sequence numbers in the original application TCP flow unchanged, and hence is transparent to middleboxes.

5 Conclusion

In this paper, we showed how the end-host hypervisor can provide a sweet spot for implementing a spectrum of load-balancing algorithms that are fine-grained, congestion-aware, and reactive to network dynamics at round-trip timescales. We proposed CLOVE, a scalable hypervisor-based load-balancer that requires no changes to existing guest VMs or to existing physical network switches. Using extensive simulations, we showed that CLOVE captures some 80% of the performance gain of best-of-breed hardware-based load-balancing algorithms without the need for expensive hardware replacement.

In future work, we plan to study the fundamental load-balancing gap between obtaining exact utilization values and obtaining only congestion indications as in ECN. Our suggestion of CLOVE-INT is a first step in that direction.

Acknowledgments: We thank the reviewers for their valuable feedback. We would also like to thank Aran Bergman, Ben Pfaff, Martin Casado, Guido Appenzeller, Jim Stabile, and Bruce Davie who gave comments on earlier draft versions of this paper. This work was supported in part by the NSF under the grant CNS-1162112 and the ONR under award N00014-12-1-0757.

6 References

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

[2] T. Benson, A. Anand, A. Akella, and M. Zhang, "Microte: Fine grained traffic engineering for data centers," *ACM CoNEXT*, 2011.

[3] C.-Y. Hong, S. Kandula, R. Mahajan, M. Zhang, V. Gill, M. Nanduri, and R. Wattenhofer, "Achieving high utilization with software-driven WAN," *SIGCOMM CCR*, vol. 43, no. 4, pp. 15–26, 2013.

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

[5] D. Wischik, C. Raiciu, A. Greenhalgh, and M. Handley, "Design, implementation and evaluation of congestion control for multipath TCP," *NSDI*, 2011.

[6] 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," *ACM SIGCOMM*, 2014.

[7] N. Katta, M. Hira, C. Kim, A. Sivaraman, and J. Rexford, "Hula: Scalable load balancing using programmable data planes," *SOSR*, 2016.

[8] 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 *ACM CoNEXT*, 2013.

[9] S. Kandula, D. Katabi, S. Sinha, and A. Berger, "Dynamic load balancing without packet reordering," *ACM SIGCOMM Computer Communication Review*, vol. 37, no. 2, pp. 51–62, 2007.

[10] S. Sen, D. Shue, S. Ihm, and M. J. Freedman, "Scalable, optimal flow routing in datacenters via local link balancing," *ACM CoNEXT*, 2013.

[11] S. Ghorbani, B. Godfrey, Y. Ganjali, and A. Firoozshahian, "Micro load balancing in data centers with DRILL," *ACM HotNets*, 2015.

[12] E. Zahavi, I. Keslassy, and A. Kolodny, "Distributed adaptive routing convergence to non-blocking DCN routing assignments," *IEEE JSAC*, 2014.

[13] W. Cui and C. Qian, "Difs: Distributed flow scheduling for adaptive routing in hierarchical data center networks," *ACM/IEEE ANCS*, 2014.

[14] X. Wu and X. Yang, "Dard: Distributed adaptive routing for datacenter networks," *IEEE ICDCS*, 2012.

[15] A. Kabbani, B. Vamanan, J. Hasan, and F. Duchene, "Flowbender: Flow-level adaptive routing for improved latency and throughput in datacenter networks," *ACM CoNEXT*, 2014.

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

[17] S. Guenender, K. Barabash, Y. Ben-Itzhak, A. Levin, E. Raichstein, and L. Schour, "NoEncap: overlay network virtualization with no encapsulation overheads," *ACM SOSR*, 2015.

[18] C. Kim, , A. Sivaraman, N. Katta, A. Bas, A. Dixit, and L. J. Wobker, "In-band network telemetry via programmable dataplanes," Demo paper at SIGCOMM '15.

[19] N. Dukkipati and N. McKeown, "Why flow-completion time is the right metric for congestion control," *SIGCOMM Comput. Commun. Rev.*, vol. 36, pp. 59–62, Jan. 2006.

[20] B. Augustin, X. Cuvellier, B. Orgogozo, F. Viger, T. Friedman, M. Latapy, C. Magnien, and R. Teixeira, "Avoiding traceroute anomalies with Paris traceroute," *IMC*, 2006.

[21] Cisco, "ACI Fabric Fundamentals." http://www.cisco.com/c/en/us/td/docs/switches/datacenter/aci/apic/sw/1-x/aci-fundamentals/b_ACI-Fundamentals/b_ACI_Fundamentals_BigBook_chapter_0100.html.

[22] "A stateless transport tunneling protocol for network virtualization." See https://tools.ietf.org/html/draft-davie-stt-01, 2012.

[23] S. Kandula, D. Katabi, B. Davie, and A. Charny, "Walking the tightrope: Responsive yet stable traffic engineering," *ACM SIGCOMM*, 2005.

[24] M. Alizadeh, A. Greenberg, D. A. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta, and M. Sridharan, "Data center tcp (DCTCP)," SIGCOMM 2010.

[25] T. Issariyakul and E. Hossain, *Introduction to Network Simulator NS2*. Springer Publishing Company, Incorporated, 1st ed., 2010.