Clove: Congestion-Aware Load Balancing at the Virtual Edge

Naga Katta[1,2], Aditi Ghag[3], Mukesh Hira[3], Isaac Keslassy[3,4],
Aran Bergman[3,4], Changhoon Kim[5], Jennifer Rexford[2]

[1] *Salesforce.com* [2] *Princeton University* [3] *VMware* [4] *Technion* [5] *Barefoot Networks*

ABSTRACT

Most datacenters still use Equal Cost Multi-Path (ECMP), which performs congestion-oblivious hashing of flows over multiple paths, leading to an uneven distribution of traffic. Alternatives to ECMP come with deployment challenges, as they require either changing the tenant VM network stacks (*e.g.,* MPTCP) or replacing all of the switches (*e.g.,* CONGA). We argue that the hypervisor provides a unique point for implementing load-balancing algorithms that are easy to deploy, while still reacting quickly to congestion. We propose Clove, a scalable load-balancer that (i) runs entirely in the hypervisor, requiring no modifications to tenant VM networking stacks or physical switches, and (ii) works on any topology and adapts quickly to topology changes and traffic shifts. Clove relies on standard ECMP in physical switches, discovers paths using a novel traceroute mechanism, uses software-based flowlet-switching, and continuously learns congestion (or path utilization) state using standard switch features. It then manipulates packet-header fields in the hypervisor switch to direct traffic over less congested paths. Clove achieves 1.5 to 7 times smaller flow-completion times at 70% network load than other load-balancing algorithms that work with existing hardware. Clove also captures some 80% of the performance gain of best-of-breed hardware-based load-balancing algorithms like CONGA that require new equipment.

ACM Reference format:

Naga Katta, Aditi Ghag, Mukesh Hira, Isaac Keslassy, Aran Bergman, Changhoon Kim, Jennifer Rexford. 2017. Clove: Congestion-Aware Load Balancing at the Virtual Edge. In *Proceedings of CoNEXT '17: The 13th International Conference on emerging Networking EXperiments and Technologies, Incheon, Republic of Korea, December 12–15, 2017 (CoNEXT '17),* 13 pages.

https://doi.org/10.1145/3143361.3143401

1 INTRODUCTION

The growth of cloud computing over recent years has led to the deployment of large datacenter networks based on multi-rooted leaf-spine or fat-tree topologies. These networks rely on multiple paths between pairs of endpoints to provide a large bisection bandwidth, and are able to handle a large number of end-points together with high switching capacities. Moreover, they have stringent performance requirements from a diverse set of applications with conflicting needs. For example, streaming and file transfer applications require high throughput, whereas applications that rely on the composition of several subroutines, such as map-reduce paradigms and/or microservice architectures, require low latency, not only in the average case but also in the 95th percentile and beyond.

An efficient distribution of traffic over multiple paths between endpoints is key to achieving good network performance in datacenter environments. However, a vast majority of datacenters continue to use Equal-Cost Multi-Path (ECMP), which performs static hashing of flows to paths and is known to provide uneven distribution and poor performance. As summarized in Figure 1, a number of alternatives have been proposed to address the shortcomings of ECMP. These come with significant deployment challenges and limitations that largely prevent their adoption. Centralized schemes are too slow for the volatile traffic patterns in datacenters. Host-based methods such as MPTCP [33] require changes to kernel network stack in guest virtual machines, and hence, are challenging to deploy because operators of multi-tenant datacenters often do not control the end-host stack. In-network per-hop load-balancing algorithms such as CONGA [2] require replacing every network switch with one that implements a new state-propagation and load-balancing algorithm.

It behooves us to ask the question: "Can network traffic be efficiently load-balanced over multiple paths in a dynamically varying network topology, without changing either the end-host transport layer or the standard off-the-shelf ECMP-based network switches?". We believe that the virtual switch in the hypervisor provides a unique opportunity to achieve this goal. The inefficiencies of uneven traffic distribution on equal-cost paths can be addressed to a large extent by dividing long-lived flows into small units, and routing these units independently instead of routing the entire long-lived flow on the same path. Indeed, this has been done in Presto [16], which divides a flow into fixed-size *flowcells*, routes the flowcells independently, and re-assembles out-of-order flowcells back in order before delivering them to the guest VM. However, Presto uses a non-standard

CoNEXT '17, December 12–15, 2017, Incheon, Republic of Korea
© 2017 Association for Computing Machinery.
ACM ISBN 978-1-4503-5422-6/17/12...$15.00
https://doi.org/10.1145/3143361.3143401

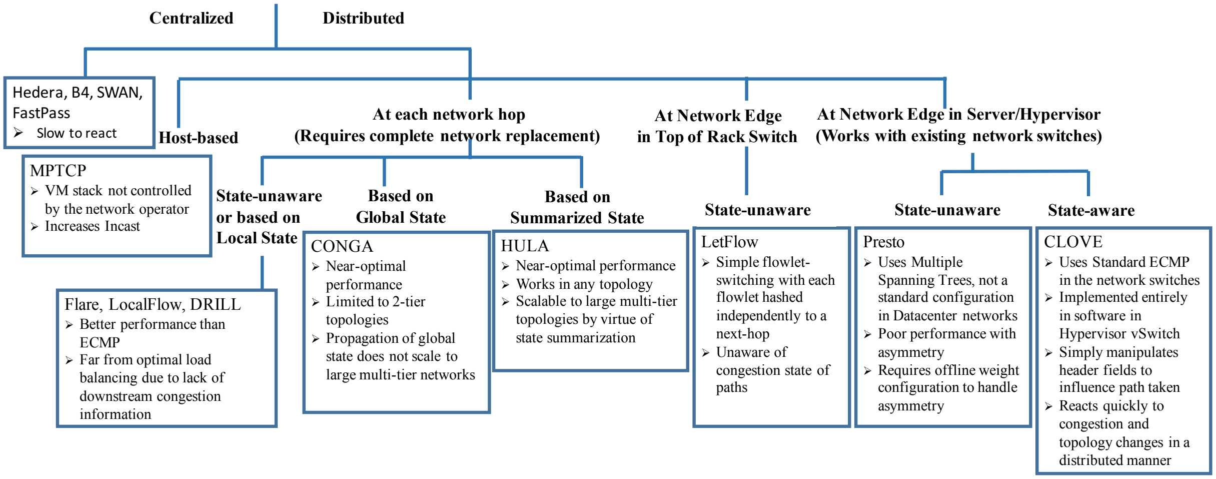

Centralized
Distributed
Hedera, B4, SWAN, FastPass
➢ Slow to react
Host-based
At each network hop (Requires complete network replacement)
At Network Edge in Top of Rack Switch
At Network Edge in Server/Hypervisor (Works with existing network switches)
MPTCP
➢ VM stack not controlled by the network operator
➢ Increases Incast
State-unaware or based on Local State
Based on Global State
Based on Summarized State
State-unaware
State-unaware
State-aware
Flare, LocalFlow, DRILL
➢ Better performance than ECMP
➢ Far from optimal load balancing due to lack of downstream congestion information
CONGA
➢ Near-optimal performance
➢ Limited to 2-tier topologies
➢ Propagation of global state does not scale to large multi-tier networks
HULA
➢ Near-optimal performance
➢ Works in any topology
➢ Scalable to large multi-tier topologies by virtue of state summarization
LetFlow
➢ Simple flowlet-switching with each flowlet hashed independently to a next-hop
➢ Unaware of congestion state of paths
Presto
➢ Uses Multiple Spanning Trees, not a standard configuration in Datacenter networks
➢ Poor performance with asymmetry
➢ Requires offline weight configuration to handle asymmetry
CLOVE
➢ Uses Standard ECMP in the network switches
➢ Implemented entirely in software in Hypervisor vSwitch
➢ Simply manipulates header fields to influence path taken
➢ Reacts quickly to congestion and topology changes in a distributed manner

Figure 1: Landscape of Network Load Balancing Algorithms

Multiple Spanning Trees-based approach to routing traffic in the network fabric, and requires centralized computation of path weights to accommodate asymmetric network topologies. Such a centralized computation does not react fast enough to a dynamically varying topology. Section 8 describes in more detail important drawbacks of prior work on network load balancing. It is challenging to optimally route flowlets on arbitrary network topologies while continuously adapting to (i) rapidly varying congestion state and (ii) changes in the topology due to link failures and/or background traffic.

Clove. We present Clove, an adaptive and scalable hypervisor-based load-balancing solution implemented entirely in the virtual switches of hypervisors. Clove 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 and network topology.

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 influence the path that each packet takes in the ECMP-based physical network. Clove then attempts to pick paths that avoid congestion. Specifically, it relies on three important components:

(1) Indirect Source Routing. Clove uses the virtual switch in the hypervisor to control packet routing. We assume at first that the datacenter is based on a network overlay [14] (*e.g.,* STT, VxLAN, NV-GRE, GENEVE), and later discuss non-overlay environments. In such an ECMP-based overlay network, the source hypervisor does not know in advance how a new packet header will impact the ECMP routing decided by the existing network switches. However, by sending probes with varying source ports in the probe encapsulation headers, the source hypervisor can discover a subset of source ports that lead to distinct paths. Then, for each outgoing packet, the hypervisor can modify the encapsulation header by setting the appropriate source port, and thereby effectively influence the path taken by the packet.

(2) Flowlet Switching. The second component of Clove is its introduction of software-based flowlet-switching [21]. 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.,* tiny groups of packets in a flow separated by a sufficient idle gap. It can then independently pick a new path for each new flowlet.

(3) Congestion-aware load-balancing. The last component of Clove is an algorithm that reacts to both short-term congestion, *e.g.,* resulting from poor load-balancing, and long-term network asymmetry, *e.g.,* resulting from failures or from asymmetrical workloads, by increasing the probability of picking uncongested paths for new flowlets. Clove schedules new flowlets on different paths by rotating through source ports in a weighted round-robin fashion, while continuously adjusting path weights in response to congestion.

In order to study the incremental gain from tracking congestion accurately, we evaluate three algorithms in increasing order of congestion-awareness of the algorithm.

First, we introduce *Edge-Flowlet*, a simple routing scheme that only uses the first two components, without any congestion-avoidance component: the source virtual switch simply picks a new source port for each flowlet in a random manner, unaware of network path state. Interestingly, we show that it still manages to indirectly take congestion into account and outperform ECMP, mainly because congestion tends to delay ACK clocking and increase the inter-packet gap, thus leading to the creation of new flowlets that get routed on different paths.

We then present 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), 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. We have implemented Clove in the Open vSwitch (OVS) datapath of Linux hypervisors in a VMware NSX network virtualization deployment. We test Clove on a two-tier leaf-spine testbed with multiple paths in presence and absence of topology asymmetry caused by link failures. When compared with schemes like ECMP and Presto [16] that work with existing network hardware, Clove obtains 1.5x to 7x smaller flow completion times at 70% network load, mainly because these schemes do not take congestion and asymmetry into account. An interesting result from our testbed evaluation is that Edge-Flowlet alone helps achieve 4.2x better performance than ECMP at 80% load.

In order to compare our schemes with more complex hardware-based alternatives such as CONGA that we could not deploy since it requires custom ASIC fabric, we also run packet-level simulations in NS2. We show that our edge-based schemes help improve upon ECMP in terms of average and 99th-percentile flow completion time, and that their performance gains get increasingly close to those of hardware-based CONGA. Specifically, (i) Edge-Flowlet already captures some 40% of the performance gained by CONGA over ECMP; (ii) Clove-ECN captures 80%; and (iii) Clove-INT comes 95% close to CONGA's performance. Overall, we illustrate that there are a set of 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.

This paper makes the following novel contributions:

- We present a spectrum of variations of a novel network load balancing algorithm, Clove, that works with off-the-shelf network switches, requires no changes to tenant VM network stack, and handles topology asymmetry.
- We present the design and implementation of Clove in Open Virtual Switch, and provide an in-depth discussion of its implementation challenges.
- We evaluate our Clove implementation against other load balancing schemes in a testbed with a 2-tier leaf-spine topology and 32 servers imitating client-server RPC workloads. We show that Clove outperforms all comparable alternatives by 1.5x-7x in terms of average flow completion time (FCT) at high load.
- Finally, using packet-level simulations, we show that our hypervisor-based load-balancing schemes capture most of the improvements provided by the best hardware-based schemes, while being immediately deployable and not requiring complete network replacement.

2 HYPERVISOR-BASED LOAD BALANCING

Network load balancing is difficult in datacenters when there is an asymmetry between the network topology and the traffic traversing the network. In some cases, this is due to topologies (like BCube [15]) that are asymmetric by design. In most cases, datacenters deploying symmetric topologies like Clos and Fat-tree show asymmetry due to frequent link failures [25, 34] that can reduce delivered traffic by up to 40% [13] or due to heterogeneous switching equipment (*e.g.,* switch ports from different vendors with different link speeds) that occur in large deployments. The resultant asymmetry makes it difficult to load balance packets because optimal scheduling of datacenter flows requires real-time information about changes in traffic demands and shifts in path congestion.

2.1 Design Goals

An ideal hypervisor-based load balancing solution should satisfy the following goals to achieve optimal performance, yet be simple to deploy.

Path discovery and indirect source routing: The source virtual switch can indirectly influence the routes taken by the packets when the network switches are based on a standard ECMP. To do so, for each destination, it should first identify a set of 5-tuple header values that the network switches will map to distinct (ideally disjoint) paths using ECMP, and later should appropriately set these 5-tuple values for each packet. The mapping has to be discovered in any network topology, with no knowledge of the ECMP hashing functions used by the network switches. This mapping also has to be kept up-to-date and updated after any network topology changes.

Granularity of routing decisions: In order to achieve optimal load balancing, routing decisions have to be imposed at the level of fine-grained flow chunks, without causing out-of-order delivery at the receiving VM.

Network state awareness: The source hypervisor should monitor the state of the identified paths (*e.g.,* utilization or congestion) at round-trip timescales using standard switch features, and then make routing decisions based on a state that is as real-time as possible.

Minimal packet processing overhead: Dataplane operations of keeping network state information up-to-date, identifying flow segments that may be independently routed, making state-aware routing decisions, and manipulating packet header fields appropriately, should all be achieved with minimal packet processing overhead.

2.2 Opportunities

The confluence of a number of recent trends in datacenter networking makes it feasible to implement network load balancing entirely at the network edge without requiring any changes to guest VMs or network switches, yet achieve good load balancing performance.

Adoption of network overlays: Network overlays have been recently widely adopted in multi-tenant datacenter networks to enable provisioning of per-tenant virtual topologies on top of a shared physical network topology, and achieve isolation between these virtual topologies. In overlay networks, the source virtual switch appends to each packet an encapsulation header, which contains a new 5-tuple. This "outer" 5-tuple is used by ECMP-based switches to route the packet in the physical network. Since the source port in the encapsulation header is essentially arbitrary and the destination port is fixed as well, by selecting specific source ports, the virtual switch gains the ability to influence the path of the packet.

Real-time network monitoring: An ideal load balancer needs a way to monitor network state such as link utilization and adapt to it at round-trip timescales. The emergence of In-band Network Telemetry (INT) [24] provides the virtual switch with an additional set of previously-unavailable telemetry features that can be used to efficiently load-balance from the edge. INT is being increasingly adopted by the industry [7, 30] to get better visibility into the network state. There have also been multiple IETF standards [9, 10] put forward by various industry participants recognizing the need for supporting telemetry standards across multiple vendors.

Stateful packet processing in the virtual switch: An algorithm that routes flowlets dynamically based on network state at the beginning of a flowlet needs to keep state so that all packets of the flowlet are routed on to the same path. Recent advances in performance optimization of the Open Vswitch make it possible to do stateful packet processing at line rate.

3 CLOVE DESIGN

In this section, we describe the design of Clove, the first virtualized, congestion-aware dataplane load-balancer for datacenters that achieves the above design goals.

3.1 Path Discovery using Traceroute

In overlay networks, the source hypervisor encapsulates packets received from a VM using an overlay encapsulation header. Our goal is to use standard off-the-shelf ECMP-based network switches and influence the packet paths by manipulating the 5-tuple fields in the encapsulation header, since ECMP pre-dominantly determines the path by computing a hash on these fields.

We implement a traceroute mechanism in the source hypervisor, so as to discover, for each destination, a set of encapsulation-header transport protocol source ports that map to distinct network paths. Specifically, for each destination, the source hypervisor sends periodic probes with a randomized encapsulation-header transport protocol source port, so that the probes travel on different paths using ECMP. The rest of the 5-tuple is typically fixed: the source and destination IP addresses are those of the source and destination hypervisors, the transport protocol and its destination port number are typically dictated by the encapsulation protocol in use (depending on the overlay protocol). 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 switch interfaces along 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. The path discovery mechanism can work with any topologies with ECMP-based layer-3 routing.

Once we have mapped all these random source ports to specific paths, we want Clove to select a set of k source ports leading to k distinct (ideally disjoint) paths. To pick these k paths, we use a heuristic whereby we greedily add the path that shares the least number of links with paths already picked.

Probes are sent periodically to adapt to the changes and failures in the network topology. Probing is done on the order of hundreds of milliseconds to limit the network bandwidth used by probe traffic. Probes to different destination hypervisors may be staggered over this interval. As a topology change causes the number of ECMP-nexthops for a destination to change at a switch hop, the same static hash function at this hop will now map source ports differently. Thus, any change in the network topology that affects even a single path to a particular destination requires the entire mapping of source ports to the destination to be rediscovered. As an optimization, network state (path utilization, congestion state, etc.) learned for a path may be maintained through such a transition, with only the source port mapping to the path changing through the transition.

Note that the concept of tracing the route of a particular application by sending probes with specific transport-protocol header fields is well understood, *e.g.,* in the Paris traceroute [4]. However, this has not been used before in the context of discovering distinct equal-cost paths and load-balancing network traffic over these paths.

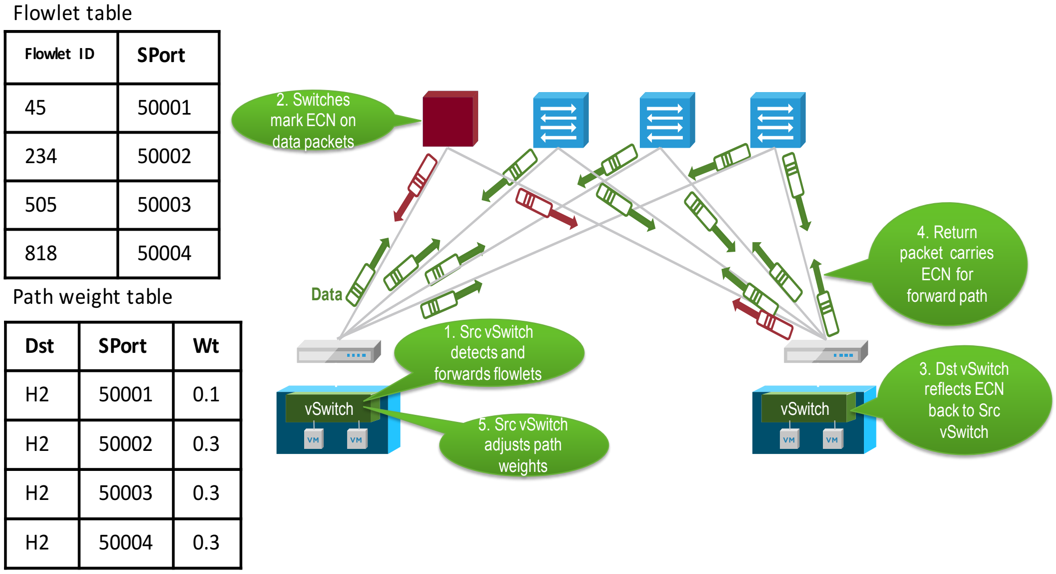
3.2 Routing Flowlets

In order to evenly distribute flows over the mapped network paths at a finer granularity, Clove divides each flow into flowlets. Flowlets are bursts of packets in a flow that are 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 in physical switches (*e.g.,* in FLARE [21] and in Cisco's ACI fabric [8]), but to the best of our knowledge not in virtual switches. *Flowlet time-gap*, the inter-packet time gap between subsequent packets of a flow that triggers the creation of a new flowlet [21], is an important parameter. Based on previous work [2, 20], we recommend that the flowlet gap be between $1\times$–$2\times$ the network round trip-time (RTT), for effective performance. We propose three schemes with varying path selection techniques for distributing flowlets from the network edge in increasing order of sophistication and performance gain.

Edge-Flowlet: We first consider a very simple routing scheme wherein the source virtual switch simply picks a new source port for each flowlet in a random manner, unaware of network path congestion. We refer to this simple scheme as *Edge-Flowlet*. 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, even though the source virtual switch is not learning about network state, it is indirectly re-routing flows experiencing congestion. Besides, breaking up large elephant flows into flowlets also helps break persistent conflicts between elephant flows sharing a common bottleneck link. For all these reasons, the Edge-Flowlet algorithm is expected to perform better than flow-based load balancing using ECMP.

Clove-ECN: Next, we consider learning about congestion along network paths using Explicit Congestion Notification (ECN), which has been a standard feature in network switches for many years. ECN was primarily designed to indicate congestion to the source transport stack, and have it throttle back in the event of congestion. A source indicates that it is ECN capable by setting the ECN-Capable-Transport (ECT) bit in the IP header. ECN-enabled network switches set Congestion-Experienced (CE) bits in the IP header when a packet experiences an egress queue length greater than a configured threshold. The receiving transport stack relays ECN back to the source transport stack, which in turn throttles back in response, until the congestion on the switch port clears.

Figure 2 illustrates how Clove-ECN, implemented in the hypervisor virtual switch, exploits the ECN capability in network switches



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
1. Src vSwitch detects and forwards flowlets
vSwitch
VM
VM
5. Src vSwitch adjusts path weights
4. Return packet carries ECN for forward path
vSwitch
VM
VM
3. Dst vSwitch reflects ECN back to Src vSwitch

Figure 2: Clove-ECN congestion-aware routing.

to learn about congestion on specific paths, and route flowlets along alternate uncongested paths to the destination. It consists of two distinct mechanisms: (a) *detecting congestion* along a given path, and (b) *reacting to congestion* on this path by favoring other paths for future new flowlets.

Detecting Congestion: The source virtual switch sets ECT bits in the encapsulation IP header. The receiving hypervisor intercepts ECN information and relays it back to the sending hypervisor, indicating the source port mapped to the network path that experienced congestion. Reserved bits in the encapsulation header of reverse traffic (towards the source) are used to encode the source-port value that experienced congestion in the forward direction. For instance, in the Stateless Transport Tunneling (STT) protocol, the Context field in the STT header may be used for this purpose.

Reacting to Congestion: Clove-ECN uses weighted round robin (WRR) to load balance flowlets on 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 some predefined proportion, *e.g.,* by a third. 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 (for each new flowlet) according to the new set of weights. As long as there is at least one uncongested path to the destination, the source virtual switch masks the ECN marking from the sending VM. Only when all network paths to a destination are sensed to be congested, it relays ECN to the sending VM, triggering it to throttle back.

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 an 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 two important parameters:

ECN threshold: This is the threshold in terms of queue length on a switch-port beyond which switches start marking the packets with ECN. Similarly to the recommendations by DCTCP [3], 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 in a flow relays congestion marking to the associated sender in that flow. The receiver should send feedback more frequently than the frequency at which load balancing decisions are being made, as recommended in TexCP [20]. We use half the RTT as the ECN relay frequency in our design.

Clove-INT: Finally, we consider a variation of Clove based on proactively monitoring the exact utilization of each path, and routing flowlets along the least-utilized path. We want to prevent congestion from occurring along any path, instead of reacting after congestion has occurred on specific paths.

In-band Network Telemetry (INT) [24], a technology likely to be available in datacenter network switches in the near future, enables network endpoints to embed instructions in packets, requesting every network hop to insert network state in packets as they traverse the network, potentially at line-rate. As the packets arrive at the destination endpoint, the endpoint has access to the state at each link along the hop that is as close to real-time as possible.

In Clove-INT, the source virtual switch requests each network element to insert egress link utilization in packet headers. When the packet is received at the destination hypervisor, it relays back the maximum link utilization along the path to the source virtual

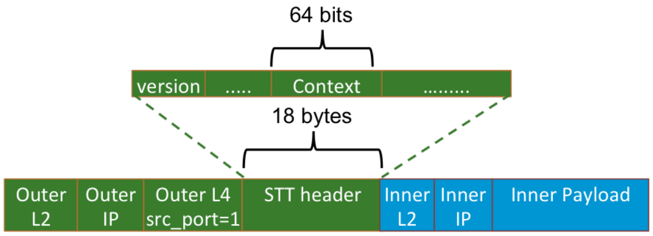


Figure 3: Encapsulation with STT headers

switch together with the encapsulation header source port in the packet. As in Clove-ECN, it uses reserved bits in the overlay encapsulation header, the difference being that in this case, real-time path utilization is relayed back instead of binary congestion state. The source virtual switch proactively routes new flowlets on the least utilized path. Note that while this requires a new capability at each switch and hence a physical network upgrade, this approach can be used when INT becomes a standard feature in datacenter network switches. Load balancing decisions are still made in software in the hypervisor virtual switch. This is in contrast to algorithms such as CONGA, which implement the proprietary state propagation and load balancing algorithms at each hop, thus requiring all switches to have the same proprietary implementation from the same vendor.

4 IMPLEMENTATION

We have implemented Edge-Flowlet and Clove-ECN in the Open vSwitch (OVS, version 2.4.0.0) kernel datapath. The first component for Clove consists of a user-space traceroute daemon that periodically sends probes (with rotating source ports) to various destination hypervisors. The daemon collects the path traces and distills a set of disjoint paths and the corresponding source ports to be used for data traffic.

Indirect source routing using STT encapsulation. As indicated earlier, Clove exploits tunnel encapsulation, typically used in network virtualization, to isolate Clove's mechanisms from affecting the actual tenant traffic. Open vSwitch supports the Stateless Transport Tunneling (STT) protocol (see Figure 3), which we use for encapsulating the tenant VM traffic before sending it onto the physical underlay. Currently, the STT protocol encapsulates each TCP segment (with a maximum size of 64KB) received from a VM with an outer TCP header whose TCP source port is set to the hash of the inner TCP packet header fields (apart from other fields that are fixed for each source-destination hypervisor pair). Instead, the Clove implementation in the OVS kernel datapath picks one of the encapsulation TCP source ports that were previously identified by the traceroute daemon in a congestion-aware manner, as described in Section 3. Subsequently, this segment is sent to the NIC for segmentation offload, breaking the segment into MTU-sized packets before sending them onto the physical network.

Communicating Clove metadata amongst hypervisors. When a packet is marked with the ECN bit by switches in the network and reaches the destination, the receiver has to relay this information back to the sender. However, we cannot rely on the receiver VM TCP stack to do this, since our objective is to keep the VM stack unmodified and hence unaware of any ECN marking in the underlay. Instead, the receiver hypervisor intercepts the ECN state and feeds it back to the sender using some reserved bits in the STT header of the return packets, as previously shown in Figure 2. A hypervisor encodes the ECN information in bits borrowed from the STT context (shown in Figure 3) — the encapsulation header source port it received and the *ecnSet* bit indicating whether or not the received packet experienced congestion. Note that this information cannot be relayed back to the sender using the typical ECN echo mechanism, because the receiver cannot use the sender's source port to be its outer destination port (which is set to fixed STT port). Hence, Clove uses a separate header space (the STT context bits) to encode this information.

Stateful packet processing. An important aspect of implementing Clove is that of maintaining network state in the hypervisor based on Clove's metadata about congestion on various paths. In a multicore multi-threaded environment (for processing multiple packets in parallel), this has to be done using efficient locking mechanisms such as Read-Copy-Update (RCU) [26] locks to minimize blocking of threads when updating state—a mechanism already used for updating per-connection state in the Open vSwitch today. We use RCU hash lists supported by the kernel libraries to maintain state for (i) detecting flowlets and (ii) storing per-path congestion state. The lookups and updates to these data structures happen in the datapath while maintaining the line rate throughput of at least 40Gbps per hypervisor.

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 in the order of thousands. In addition, the number of flowlet entries is in the order of the number of destination hypervisors that the source is actively talking to at any point, *i.e.,* typically in the order of at most a thousand entries.

(b) Probe overhead: Clove sends periodic probes that map source ports to network paths in order to 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, with negligible overhead. Therefore, if Clove probes are sent every few seconds, the overall load should be similar. The probe frequency only determines the reaction time to a change in network topology, which is an infrequent occurrence.

Tuning algorithm parameters: An effective deployment of Clove needs proper tuning of several key parameters that influence its performance as discussed in the previous section.
(i) Low flowlet time-gap increases packet reordering at the receiver and large flowlet time-gap leads to coarse-grained flowlets, increasing the possibility of congestion.
(ii) At low ECN-relay frequency, Clove makes suboptimal choices based on stale ECN information, while if it is too high, it would incur high overhead for processing ECN information in the software datapath.
(iii) In our experience, Clove is robust to small shifts in the flowlet time-gap and the ECN relay frequency (both within a couple of RTTs), but is more sensitive to the ECN threshold.
(iv) We noticed that if the ECN threshold is a few segments above

the threshold of the 20 packet limit, Clove reacts very slowly to elephant flow collisions. However, if we set the threshold lower, then Clove would over-react to the typically bursty traffic sent by the TCP segmentation offload.

5 TESTBED EVALUATION

In this subsection, we illustrate the effectiveness of the Clove load-balancer by testing our implementation of Clove in OVS against the following schemes.

ECMP: Each packet's outer TCP source port is determined by taking a hash of the inner packet's TCP 5-tuple (src IP, dest IP, src port, dest port, protocol).

Edge-flowlet: This method also uses hash-based TCP source ports on the outer packet header, except that the hash changes for each new flowlet. This involves taking a hash of the 6-tuple that includes the flow's 5-tuple plus the flowlet ID (which is incremented every time a new flowlet is detected at a switch).

MPTCP: In order to compare MPTCP and Clove, we deployed MPTCP v0.89 on Linux kernel 3.18. We disabled the *mptcp_-checksum* option since we noticed significant drop in the throughput in our experiments, and set the sub-flow count to 4 which gave us the best results in terms of network path utilization and application performance. Additionally, we enabled Large Receive Offload (LRO) for improving throughput and latency. The MPTCP implementation was occasionally unstable, and incurred high CPU utilization even when running as few as 4 iPerf sessions in parallel between two end-points.

Presto: We implemented Presto [16] in OVS with modifications that adapt Presto to Layer-3 ECMP routing in today's data centers. Unlike the implementation described in the paper, we do not use a centralized controller for configuring multiple spanning trees and shadow-MAC-based forwarding in the dataplane. Instead, we rotate through a pre-calculated set of encapsulation header source ports for flowcells (TSO segments) to route them on distinct network paths in a round-robin manner. We implemented flowcell reassembly logic similar to the one discussed in the paper. To assist with the reordering of flowcells at the receiver side, we encode the flow ID (hash of the 5-tuple) and a monotonically increasing flowcell ID in the encapsulation header. At the receiver end, the reassembly algorithm merges out-of-order flowcells in order before they are pushed to the guest VM. We use an empirical static timeout to send buffered segments to the guest VM, and set a limit on the number of flowcells that are buffered in order to recover from packet loss. Note that while this modified version makes certain packet-format changes to adapt to ECMP-based routing, it faithfully reproduces the core elements of Presto. For an asymmetric topology, the Presto paper does not delve into concrete details of how path weights are updated at network RTT timescales using a centralized controller based detection and spanning-trees re-configurations. Hence, we use (ideal) statically configured path weights that reflect the topology in our implementation of Presto to give it the benefit of doubt.

We answer the following questions with evaluation in a real testbed environment, using a realistic workload: (i) How does Clove perform in symmetric topologies compared to other schemes? (ii) How effective is Clove when link failures lead to asymmetry in the network topology? (iii) How does Clove perform under incast?

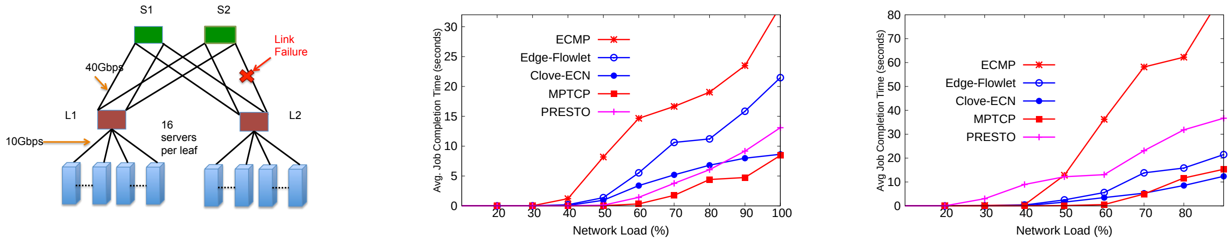
Topology: Our testbed consists of a 2-tier Clos topology as shown in Figure 4a, with two spines (S1 and S2) connecting two leaf switches (L1 and L2). Each leaf switch is connected to either spine by two 40G links. This gives a total of 160G for the bisection bandwidth. Routing is set up such that all traffic received by a spine switch on the first link from one of the leaf switches is forwarded on the first link towards the other leaf switch; same is the case for all traffic received on the second link from a leaf switch. Thus, there are a total of four disjoint paths that a packet could take to travel from one leaf to another. Each leaf is connected to 16 servers with 10G links. This makes sure that the network is not oversubscribed and the 16 servers on one leaf can together saturate the 160G bandwidth. In order to simulate asymmetry in the baseline symmetric topology, we disable one of the 40G links connecting the spine $S2$ with the leaf switch $L2$.

Empirical workload: We use a realistic workload to generate traffic for our experiments. Specifically, we rely on a web search workload [3] that is obtained from production datacenters of Microsoft. The workload is long tailed, and most of its flows are small. The small fraction of large flows contribute to a substantial portion of the traffic. We simulate a simple client-server communication model where each client chooses a server at random and initiates persistent TCP connections to the server. Of the 32 machines connected to the testbed, 16 act as clients and the rest as servers. 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 taken from an exponential distribution whose mean is tuned by the desired load on the network. We run the workload for a total of 50K jobs per client connection. Similarly to previous work [2], we look at the average Flow Completion Time (FCT) as the overall performance metric so that all flows including the majority of small flows are given equal consideration. We run each experiment with three random seeds and report the average of the three runs.

5.1 Baseline symmetric Topology

First, we compare the various load balancing schemes on the baseline symmetric topology. Figure 4b shows the average completion time for all flows as we vary the load on the network. Clove performs better than ECMP or Edge-Flowlet at higher loads but is neck-to-neck with MPTCP and Presto. At lower loads, the performance of all 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. However, as the network load becomes higher, the flows have to be carefully assigned to paths such that collisions do not occur. Given that the flow characteristics change frequently, at high network load, the load balancing scheme has to adapt quickly to the change in link utilizations throughout the network.

ECMP performs the worst because it does congestion-oblivious load-balancing at a very coarse granularity. Edge-Flowlet performs slightly better because it still does congestion-oblivious load-balancing, but at the granularity of flowlets. Two effects improve its performance. First, it is less bursty on each path. More significantly, upon congestion, there are fewer ACKs, and therefore more chances of forming a new flowlet, and therefore it is still indirectly

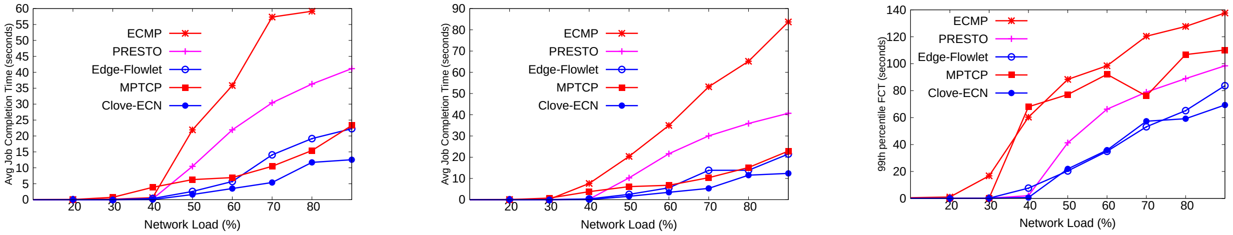


(a) Topology used in evaluation

(b) Symmetric topology - avg. FCT

(c) Asymmetric topology - avg. FCT

Figure 4: Average FCT for the web-search workload on a network testbed. Network load is measured with respect to the full bisection bandwidth of 160 Gbps.



(a) Avg. FCTs for <100 KB flows

(b) Avg. FCTs for >10 MB flows

(c) 99th percentile FCTs

Figure 5: FCT breakdown of small and large jobs for the web-search workload on a network testbed.

congestion-aware. Clove does better than both because of its fine-grained congestion-aware load-balancing. For the web-search workload, Clove achieves 2.5x lower FCT (*i.e.,* better performance) compared to ECMP and 1.8x lower FCT compared to Edge-Flowlet at 80% network load. Amongst all the load balancing schemes compared with here, MPTCP performs the best because of its usage of multiple subflows that help redistribute flow bytes on to subflows mapped to uncongested paths. This advantage of 1.2X over Clove at 80% load comes at the expense of deployment troubles with MPTCP. Presto does nearly the same as Clove-ECN owing to its round-robin flowcell spraying.

5.2 Asymmetric topology

In order to simulate a network failure that creates topological asymmetry, we brought down the 40G link between the spine switch $S2$ and switch $L2$. Subsequently, the effective bandwidth of the network drops by 25% for traffic between clients and servers. This requires the various load balancing schemes to carefully balance paths at even lower network loads compared to the baseline topology scenario. In particular, any given load balancer has to ensure that the bottleneck link connecting $S2$ to $L2$ is not overwhelmed with a disproportionate amount of traffic.

ECMP and Presto. Figure 4c shows how various schemes perform with the web search workload as the network load is varied. Since Presto assumes that the controller infers asymmetry and feeds that information to the hypervisors, we gave Presto a head start by specifying the correct path weights (0.33, 0.33, 0.17, 0.17) for the topology in its source dataplane component so that the 64KB flowcells are distributed on the paths in the appropriate ratio. The overall FCT for ECMP shoots up steeply above 50% network load. This is because, as the network load reaches 50%, the bottleneck link between $S2$ and $L2$ gets pressurized by the flows hashed to go through $S2$. Since ECMP treats all 4 paths from L1 equally, at high loads, S2 receives more traffic than it can forward on the reduced downlink bandwidth to L2. This is why it performs poorly at high network loads. Presto, owing to load balancing at the granularity of flowcells (64KB flow segments), suffers less from congestion and does 1.8X better than ECMP at 70% load. However, it still lags behind Clove-ECN (by 3.8x at 70% load) despite its ideal path weights, due to its congestion-unaware flowcell distribution. This is in line with recent research (*e.g.,* LetFlow [32]) which observes that statically assigning weights to path distribution is not enough to handle the mismatch between workload asymmetry and topology asymmetry.

Edge-Flowlet. The notable result in our experiments is the relatively better performance of Edge-Flowlet over ECMP or Presto. Edge-Flowlet does congestion-oblivious load balancing but at the granularity of flowlets. However, we noticed that new flowlets are created in the workload whenever the corresponding flows travel on a congested path. As previously mentioned, these new flowlets are being created due to delayed ACK clocking caused by congestion. Hence, compared to ECMP or Presto, Edge-Flowlet's random flowlet routing can inherently adapt to congestion. This is why Edge-Flowlet performs 4.2X better than ECMP at 80% load. It also does better than Presto despite the fact that Presto needs complex packet re-assembly logic while Edge-flowlet does not. In fact, Edge-Flowlet captures most of the gain of Clove-ECN which is impressive given the simplicity of its design.

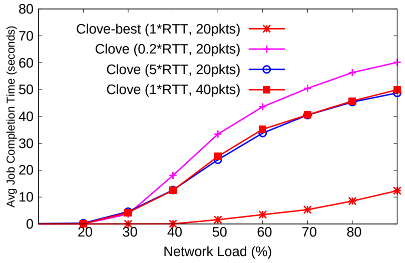


Figure 6: Performance of Clove-ECN under different parameter settings, denoted as *Clove (flowlet-threshold, ECN-threshold)*.

Clove-ECN and MPTCP. Clove-ECN does the best of all schemes because of its fast congestion-aware path selection that avoids pressure on the bottleneck link. This helps Clove-ECN achieve 7.5x better performance than ECMP and 2x better FCT than Edge-Flowlet at 80% network load. MPTCP also does nearly as well as Clove-ECN owing to its use of multiple subflows that can redistribute bytes from congested subflows to uncongested ones dynamically using the MPTCP control loop.

FCT Breakdown. Figure 5 shows the breakdown of performance separately for mice flows (of size less than 100KB) and for large flows (of size greater than 10MB). The average FCTs for both small and large flows largely reflect the *relative* overall FCT performance for each scheme. The relative performance difference between the FCTs for small flows is slightly smaller than that for large flows, because longer flows give more opportunities to react to congestion. For example, Edge-Flowlet does 3.7X better than ECMP for small flows but 4.1X better for large flows at 70% load.

99th percentile. Figure 5c shows 99th percentile FCTs under all the load balancing schemes. Here, interestingly, the relative performance story is different from that of the average FCT. MPTCP does significantly worse compared to Edge-Flowlet or Clove. We believe this is because when all the subflows of a connection get mapped to congested paths, then MPTCP suffers very badly compared to all other schemes except ECMP. While Clove (and Edge-Flowlet, to some extent) can reroute their flowlets onto uncongested paths, MPTCP's subflow mapping is static and hence affects those rare flows that are stuck with congested paths. Hence Clove does 2.7X better than MPTCP at 60% load.

Parameter sensitivity. Figure 6 shows how the performance of Clove-ECN varies with changes to two of its key parameters. For our experiments, the optimal settings were (i) 1 RTT for the flowlet interpacket timegap, and (ii) 20 packets for the switch ECN threshold. As the figure shows, if the flowlet threshold is too low (0.2×RTT), then Clove-ECN behaves closer to per-packet load balancing, sees high packet reordering and hence degrades by 5x. If the threshold is too high (5×RTT), then Clove-ECN suffers from elephant flowlet collision. Similarly, when the ECN threshold is too high (40 packets), Clove-ECN takes much longer to detect congestion and hence sees performance degradation by 4x at 80% load.

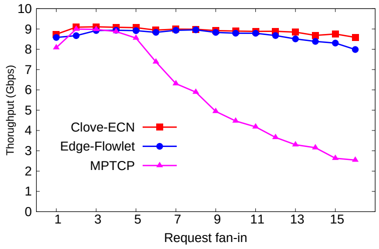


Figure 7: Performance of MPTCP, Edge-Flowlet and Clove-ECN on the incast workload measured in terms of throughput on the client access link.

5.3 Incast Workload

Workload. We designed a workload scenario that creates typical partition-aggregate patterns [2, 3] that induce incast on a machine's access link to the network. A single client sends out requests to a number of servers simultaneously, causing the servers to start sending traffic to the client concurrently. This traffic pattern stresses the queue on the link connected to the client and may result in potential packet drops. The client requests a file of 10 MB split among the n servers, where n is the request fanout. Each of the n servers sends $\frac{10^7}{n}$ bytes to the client at the same time. Once all 10 MB are received by the client, it issues the next request to another set of n servers chosen uniformly from the 16 servers in the testbed. We measure the average throughput seen on the client side over the period of 10K such job requests.

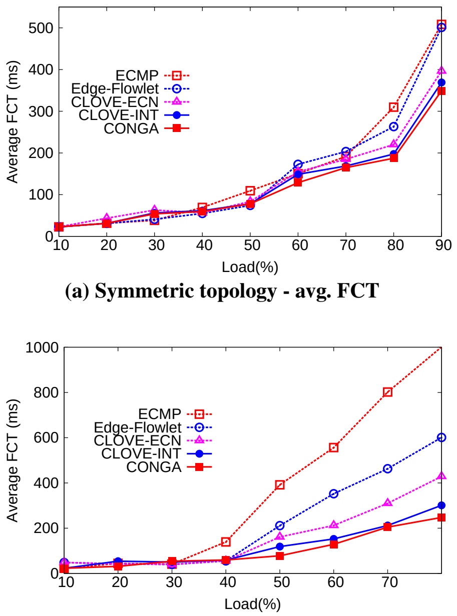
Clove-ECN does better than Edge-Flowlet and MPTCP. The workload experiment shown in Figure 7 essentially stresses the incast behavior of the TCP transport used by Clove and that of MPTCP. The figure shows that the performance of MPTCP degrades badly as the fanout for a job request increases, as confirmed similarly in CONGA [2]. For example, Clove achieves 1.9x better throughput than MPTCP for a fanout of 10 and 3.4x better for a fanout size of 16. This is mostly because MPTCP ramps up the congestion windows of all the subflows simultaneously in this synchronous workload, thereby exacerbating the pressure on the access link queues. The higher the fanout, the more the burstiness of MPTCP flows, which hurts its performance compared to Clove-ECN (which simply relies on the unmodified end-host TCP stack).

6 EXTENSIVE SIMULATIONS

In this section, using packet-level simulations in NS2 [18], we study the effectiveness of 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* [2], which modifies switches to collect switch-based measurements and routes flowlets along the least utilized path at each hop, and therefore is considered the higher end of the spectrum.

Specifically, we compare Clove-ECN with CONGA in our topology setting and investigate whether INT can be used to improve Clove-ECN's performance so that it will match that of CONGA's.



(a) Symmetric topology - avg. FCT

(b) Asymmetric topology - avg. FCT

Figure 8: Average FCT for the web-search workload in NS2. Clove-ECN, which is implementable on existing networks, captures about 80% of the performance gain between ECMP and CONGA in both topologies.

Topology and workload We simulate the same testbed topology used in Section 5. Similar to our methodology in Section 5, in order to simulate asymmetry in the baseline symmetric topology, we disable one of the 40G links connecting the spine $S2$ with the leaf switch $L2$. To generate traffic for our experiments, we use the same realistic web search workload distribution [3] from Section 5 to simulate a simple client-server communication model where each client chooses a server at random and initiates three persistent TCP connections to the server. The way the clients select flow size and inter-arrival rate is also similar. However, we run the experiments for a job count of 20K only since the simulation of these high bandwidth topologies at packet level takes significant amount of compute resources and time. We run each experiment with three random seeds and then measure the average FCT of the three runs.

6.1 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 8a shows the average 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. Clove-ECN does better 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.

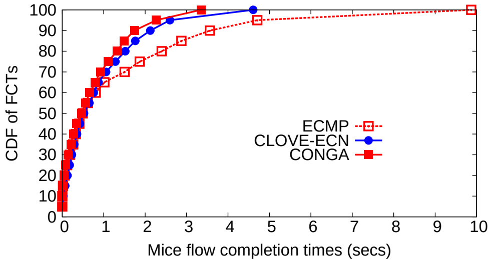


Figure 9: CDF of FCTs at 70% load with asymmetry. Clove-ECN captures 80% of the performance gain between the 99th percentiles of ECMP and CONGA.

6.2 Topology Asymmetry

When a 40G link between the spine switch $S2$ and switch $L2$ 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, ech load balancing scheme has to make sure that the bottleneck link connecting $S2$ to $L2$ is not overwhelmed with a disproportionate amount of traffic.

Figure 8b 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 $S2$. In fact, had we used an infinite-time workload, ECMP would not have theoretically converged. But since we used a finite workload as in [2] to measure the FCT, we obtain finite delays. Edge-Flowlet does slightly better than ECMP as expected, yet still performs relatively poorly (unlike observed in [32] for instance).

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 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 9 illustrates similar results by plotting the CDFs for the flow completion times of mice flows 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.

Congestion-aware vs. utilization-aware. 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.

7 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. A rigorous theoretical study of the stability of such schemes is desirable but is also hard. However, recent efforts like CONGA [2] and HULA [23] 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. As discussed towards the end of Section 5.2, we provide guidelines for stable parameter settings based on empirical evidence during our evaluation. While we did not notice stability issues during our evaluation, a rigorous study of the stability characteristics of Clove's control loop is part of future work.

Incremental Deployment: Clove is easily deployable in today's datacenters because it neither requires new network hardware nor does it require changes to the applications running on the end host. Instead, it relies on patching the hypervisor software with the capabilities to do network load balancing. The major advantage with Clove's software-based approach is that Clove can be deployed at software rollout speeds instead of waiting for slower vendor hardware upgrades. During the incremental deployment, if either the sender or receiver is not aware of Clove's metadata in the outer header, the flow will be treated as a normal TCP flow and will be routed in a congestion-agnostic manner using the standard TCP stack. In the presence of only some Clove-enabled hypervisors, Clove should adjust to bandwidth asymmetries and improve the overall performance measure for all the traffic.

Use of path latency: Since ECN can sometimes be erratic, and INT switches are not shipping yet, another way to infer congestion could be to measure the latency on each forward path to a destination. Timestamping at the NIC layer, combined with time synchronization across hypervisors using a mechanism such as IEEE 1588, would enable the receiving virtual switch to accurately determine forward latency and report it back to the sending hypervisor [27].

Non-overlay environments: In non-overlay environments with TCP applications running on VMs, the virtual switch in the source hypervisor can implement a *hidden overlay* by simply replacing the five-tuple in traffic received from a VM with the five-tuple that would otherwise be in the overlay header, hiding the real values in TCP options. The destination virtual switch copies back the original values into the header, entirely transparent to the TCP application on the destination VM.

Middleboxes. We assume that the destination of traffic being load-balanced by Clove is a Clove-capable node. This would be true in the case of a vast majority of datacenter network deployments with Clove used for load-balancing of intra-DC east-west traffic flowing directly between VMs hosted on hypervisors. In scenarios where the traffic needs to traverse a middlebox (*e.g.,* Packet Inspection Engine), the middlebox would need to be Clove-capable. Obviously, with network overlays, the middlebox would be the overlay tunnel destination and would need to implement Clove in addition to simply decapsulating the overlay header. In non-overlay environments, Clove capability would be needed to translate the transport protocol port numbers back to their original values in the inbound direction, and re-translate them to Clove-determined values in the outbound direction towards the ultimate destination VM.

Flowlet optimization: Our implementation of Clove uses a static value of flowlet time-gap to detect flowlets. Unless this value is set extremely conservatively, flowlets can still arrive out of order due to asymmetric congestion on paths, and hinder TCP performance. The flowlet time-gap may be made adaptive to the variance in RTT measured between different paths to a destination, further decreasing the probability of flowlets arriving out of order at the receiver. Moreover, flowlet sequence numbers may be carried in the encapsulation header, allowing the receiving virtual switch to put flowlets back in order (similarly to Presto [16]) so that the TCP stack in the VM does not see any out of order packets.

DCTCP. DCTCP [3] modifies the end-host transport stack to use a stream of ECN indications to react adequately to path congestion and thereby reduce the tail flow completion times. DCTCP congestion control is complementary to Clove load-balancing, and can improve it in two ways. First, Clove-ECN could use similar techniques to use a stream of ECN indications to fine tune the load-balancing weights and obtain even better performance. Second, if end-hosts are modified to implement DCTCP, datacenter switch buffers can be made significantly smaller, and therefore flowlet time-gaps can be reduced. This could potentially improve the performance of Clove, although the flowlet dynamics would be changed as well. We leave this to future research.

Is ECN slow and erratic? While Explicit Congestion Notification (ECN) is an efficient and convenient mechanism for existing network switches to convey path-level congestion to the sender, it is also seen as slightly unreliable at times. In particular, there was discussion in the past [5] about unsuccessful TCP negotiation for ECN, slow propagation of ECN information to the sender, erroneous marking/-clearing of ECN, etc. While these issues are a matter of concern, our experience with Clove's settings has been very positive with respect to ECN. In Clove, there is no negotiation or echoing of ECN by the end host TCP stack since the ECN bits are marked by the hypervisor itself. Since the RTT is very low in datacenter networks compared to WANs that were studied in the literature, the propagation of congestion information is faster. The Broadcom-based chipsets used in our evaluation seem to be easy to configure with the desired ECN threshold and we have not seen incorrect marking/clearing of ECN bits. There is no strong reason to believe that other switches based on the Broadcom chipsets will be erroneous to a large extent.

8 RELATED WORK

Centralized Algorithms: Hedera, MicroTE, SWAN, Fastpass [1, 6, 17, 19, 29] are based on a centralized scheduler that maintains global network state and calculates routes for network flows. Such algorithms are slow to react for datacenter traffic patterns and come with a prohibitive cost of querying the scheduler for short-lived latency-sensitive flows. Flowtune [28] is an additional recent work that schedules flowlets onto paths from a centralized server. While it is more scalable, it cannot adapt to failures at dataplane timescales.
Host-based Algorithms: There are many potential congestion control algorithms, such as DCTCP [3] and MPTCP [33]. Unfortunately, such algorithms need to modify the end-host transport stack. DCTCP is further discussed in Section 7. MPTCP [33] distributes each application flow over multiple TCP sub-flows with distinct five-tuples that are routed independently by ECMP, although a subset of subflows may end up being routed on the same path due to ECMP hash collisions. The multiple subflows cause burstiness and perform poorly under incast [2]. In addition, it is difficult to deploy MPTCP in datacenters because it requires change to all the end-hosts, which are outside the control of the network operator in multitenant environments. Finally, the number of subflows in MPTCP is static and does not vary in accordance with the number of network paths.

In-Network Per-Hop Distributed Algorithms: *Based on Local State (FLARE, LocalFlow, Drill [12, 21, 31]):* Each hop routes *flowlets* based on local link utilization. Accounting only for local state, these algorithms perform poorly with asymmetric paths.

Based on Complete Global State (CONGA [2]): Utilization of each link is propagated throughout the network at round-trip timescales using proprietary packet formats; each hop chooses the least-utilized path for each flowlet. All network switches have to be replaced with those running this proprietary algorithm. Global propagation of state limits scalability. CONGA is designed specifically for 2-tier leaf-spine networks.

Based on Summarized Global State (HULA [23]): Using In-band Network Telemetry (INT), a technology likely to be available in network switches in the near future, each switch advertises per-destination *best path utilization* to neighbors; each switch routes flowlets on the least utilized path towards the destination. State summarization allows the solution to scale to arbitrarily large topologies, however, the per-hop nature does require complete network replacement.
Algorithms at the Network Edge: Presto [16] is a load balancing algorithm implemented entirely at the network edge. The virtual switch in the source hypervisor forwards fixed-size flow segments (*e.g.,* 64KB) with independent source and destination *shadow* MAC addresses. These *flowcells* are routed independently in the network over multiple spanning trees. This does not work with predominantly deployed Layer-3 based ECMP forwarding in the physical network. Moreover, Presto performs poorly in asymmetric environments as it is oblivious of network state. Additionally, [16] does not provide a detailed analysis of the scheme's performance in failure cases when a centralized controller detects and re-configures the spanning trees. Consequently, it is unclear whether Presto is able to handle such cases at RTT timescales in order to deal with datacenter traffic volatility.
Juggler [11] is a mechanism that helps the network stack deal with packet reordering issues caused by splitting of flows onto multiple paths. Juggler improves upon Presto by reducing the amount of per-connection state required for packet assembly and hence complements Presto. However, it still does not effectively handle topology asymmetry.
LetFlow [32] is a recent work that independently arrived at the conclusion that a simple mechanism that splits flows into flowlets in the network can effectively adapt to topology asymmetry, compared to existing schemes. However, while LetFlow relies on new switch hardware for their implementation, our version of Edge-Flowlet is implemented entirely in the host hypervisors and hence is readily deployable. In addition, Clove-ECN and Clove-INT show how to bridge the gap between schemes like LetFlow/Edge-Flowlet and hardware-based schemes like CONGA.
Finally, in [22], we previously introduced the idea of hypervisor-based network load balancing by discovering a mapping of encapsulation header fields into distinct network paths, and forwarded flowlets over these distinct paths in a congestion-aware manner. However, it fell short of convincing that these ideas are practical and efficient in practical deployments.

9 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. In addition, we presented the Clove algorithm and implemented it in a Open Virtual Switch, showing how it obtains significant performance gains in a real network with realistic workloads. Unlike past algorithms, Clove is essentially ready to be directly implemented in multitenant datacenters without any changes to existing guest VMs or to existing physical network switches.

Acknowledgments: We thank the reviewers and our shepherd, Robert Birke, for their valuable feedback. We would also like to thank Ben Pfaff, Martin Casado, Guido Appenzeller, Jim Stabile, David Tennenhouse, 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 CCF-1535948 and the ONR under award N00014-12-1-0757.

REFERENCES

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

[2] Mohammad Alizadeh, Tom Edsall, Sarang Dharmapurikar, Ramanan Vaidyanathan, Kevin Chu, Andy Fingerhut, Francis Matus, Rong Pan, Navindra Yadav, George Varghese, and others. 2014. CONGA: Distributed congestion-aware load balancing for datacenters. *ACM SIGCOMM* (2014).

[3] Mohammad Alizadeh, Albert Greenberg, David A. Maltz, Jitendra Padhye, Parveen Patel, Balaji Prabhakar, Sudipta Sengupta, and Murari Sridharan. 2010. Data Center TCP (DCTCP). *ACM SIGCOMM* (2010).

[4] Brice Augustin, Xavier Cuvellier, Benjamin Orgogozo, Fabien Viger, Timur Friedman, Matthieu Latapy, Clémence Magnien, and Renata Teixeira. 2006. Avoiding Traceroute Anomalies with Paris Traceroute. *ACM Internet Measurement Conference* (2006).

[5] Steven Bauer, Robert Beverly, and Arthur Berger. 2011. Measuring the State of ECN Readiness in Servers, Clients,and Routers. *ACM Internet Measurement Conference* (2011).

[6] Theophilus Benson, Ashok Anand, Aditya Akella, and Ming Zhang. 2011. MicroTE: Fine grained traffic engineering for data centers. *ACM CoNEXT* (2011).

[7] Broadcom. 2017. New Trident 3 switch delivers smarter programmability. https://www.broadcom.com/blog/new-trident-3-switch-delivers-smarter-programmability-for-enterp. (2017).

[8] Cisco. 2015. 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. (2015).

[9] CISCO. 2017. Requirements for In-situ OAM. https://tools.ietf.org/html/draft-brockners-inband-oam-requirements-02. (2017).

[10] Facebook. 2017. Data-plane probe for in-band telemetry collection. https://tools.ietf.org/html/draft-lapukhov-dataplane-probe-00. (2017).

[11] Yilong Geng, Vimalkumar Jeyakumar, Abdul Kabbani, and Mohammad Alizadeh. 2016. Juggler: a practical reordering resilient network stack for datacenters. *EuroSys* (2016).

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

[13] Phillipa Gill, Navendu Jain, and Nachiappan Nagappan. 2011. Understanding Network Failures in Data Centers: Measurement, Analysis, and Implications. In *Proceedings of the ACM SIGCOMM 2011 Conference (SIGCOMM '11)*. ACM, New York, NY, USA, 350–361. DOI:https://doi.org/10.1145/2018436.2018477

[14] Sergey Guenender, Katherine Barabash, Yaniv Ben-Itzhak, Anna Levin, Eran Raichstein, and Liran Schour. 2015. NoEncap: overlay network virtualization with no encapsulation overheads. *ACM SOSR* (2015).

[15] Chuanxiong Guo, Guohan Lu, Dan Li, Haitao Wu, Xuan Zhang, Yunfeng Shi, Chen Tian, Yongguang Zhang, and Songwu Lu. 2009. BCube: A High Performance, Server-centric Network Architecture for Modular Data Centers. In *Proceedings of the ACM SIGCOMM 2009 Conference on Data Communication (SIGCOMM '09)*. ACM, New York, NY, USA, 63–74. DOI:https://doi.org/10.1145/1592568.1592577

[16] Keqiang He, Eric Rozner, Kanak Agarwal, Wes Felter, John Carter, and Aditya Akella. 2015. Presto: Edge-based load balancing for fast datacenter networks. *ACM SIGCOMM* (2015).

[17] Chi-Yao Hong, Srikanth Kandula, Ratul Mahajan, Ming Zhang, Vijay Gill, Mohan Nanduri, and Roger Wattenhofer. 2013. Achieving high utilization with software-driven WAN. *SIGCOMM CCR* 43, 4 (2013), 15–26.

[18] Teerawat Issariyakul and Ekram Hossain. 2010. *Introduction to Network Simulator NS2* (1st ed.). Springer.

[19] Sushant Jain, Alok Kumar, Subhasree Mandal, Joon Ong, Leon Poutievski, Arjun Singh, Subbaiah Venkata, Jim Wanderer, Junlan Zhou, Min Zhu, and others. 2013. B4: Experience with a globally-deployed software defined WAN. In *ACM SIGCOMM Computer Communication Review*, Vol. 43. ACM, 3–14.

[20] Srikanth Kandula, Dina Katabi, Bruce Davie, and Anna Charny. 2005. Walking the Tightrope: Responsive Yet Stable Traffic Engineering. *ACM SIGCOMM* (2005).

[21] Srikanth Kandula, Dina Katabi, Shantanu Sinha, and Arthur Berger. 2007. Dynamic load balancing without packet reordering. *ACM SIGCOMM Computer Communication Review* 37, 2 (2007), 51–62.

[22] Naga Katta, Mukesh Hira, Aditi Ghag, Isaac Keslassy, Jennifer Rexford, and Changhoon Kim. 2016. CLOVE: How I learned to stop worrying about the core and love the edge. *ACM HotNets* (2016).

[23] Naga Katta, Mukesh Hira, Changhoon Kim, Anirudh Sivaraman, and Jennifer Rexford. 2016. HULA: Scalable Load Balancing Using Programmable Data Planes. *SOSR* (2016).

[24] Changhoon Kim, , Anirudh Sivaraman, Naga Katta, Antonin Bas, Advait Dixit, and Lawrence J. Wobker. In-band Network Telemetry via Programmable Dataplanes *(Demo paper at SIGCOMM '15)*.

[25] Vincent Liu, Daniel Halperin, Arvind Krishnamurthy, and Thomas Anderson. 2013. F10: A Fault-tolerant Engineered Network. In *Proceedings of the 10th USENIX Conference on Networked Systems Design and Implementation (nsdi'13)*. USENIX Association, Berkeley, CA, USA, 399–412. http://dl.acm.org/citation.cfm?id=2482626.2482665

[26] Paul E. McKenney and Jonathan Walpole. 2007. What is RCU, Fundamentally? (17 December 2007). Available: http://lwn.net/Articles/262464/.

[27] Radhika Mittal, Vinh The Lam, Nandita Dukkipati, Emily Blem, Hassan Wassel, Monia Ghobadi, Amin Vahdat, Yaogong Wang, David Wetherall, and David Zats. 2015. TIMELY: RTT-based Congestion Control for the Datacenter. *ACM SIGCOMM* (2015).

[28] Jonathan Perry, Hari Balakrishnan, and Devavrat Shah. 2017. Flowtune: Flowlet Control for Datacenter Networks. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*. USENIX Association, Boston, MA. https://www.usenix.org/conference/nsdi17/technical-sessions/presentation/perry

[29] Jonathan Perry, Amy Ousterhout, Hari Balakrishnan, Devavrat Shah, and Hans Fugal. 2014. Fastpass: A centralized zero-queue datacenter network. *ACM SIGCOMM* (2014).

[30] SDXCentral. 2017. AT&T Picks Barefoot Networks for Programmable Switches. https://www.sdxcentral.com/articles/news/att-picks-barefoot-networks-programmable-switches/2017/04/. (2017).

[31] Siddhartha Sen, David Shue, Sunghwan Ihm, and Michael J Freedman. 2013. Scalable, optimal flow routing in datacenters via local link balancing. *ACM CoNEXT* (2013).

[32] Erico Vanini, Rong Pan, Mohammad Alizadeh, Tom Edsall, and Parvin Taheri. 2017. Let It Flow: Resilient Asymmetric Load Balancing with Flowlet Switching. *Usenix NSDI* (2017).

[33] Damon Wischik, Costin Raiciu, Adam Greenhalgh, and Mark Handley. 2011. Design, Implementation and Evaluation of Congestion Control for Multipath TCP. In *NSDI*, Vol. 11. 8–8.

[34] Junlan Zhou, Malveeka Tewari, Min Zhu, Abdul Kabbani, Leon Poutievski, Arjun Singh, and Amin Vahdat. 2014. WCMP: Weighted Cost Multipathing for Improved Fairness in Data Centers. In *Proceedings of the Ninth European Conference on Computer Systems (EuroSys '14)*. ACM, New York, NY, USA, Article 5, 14 pages. DOI:https://doi.org/10.1145/2592798.2592803