Achieving Optimal Edge-based Congestion-aware Load Balancing in Data Center Networks

Weifeng Zhang[†], Dongfang Ling[†], Yuanrong Zhang[†], Pengfei Li[†] and Guo Chen[†§*]
[†]Hunan University, China [§]Science and Technology on Parallel and Distributed Processing Laboratory (PDL)

***Abstract*—Load balancing is the key to improve the performance of data center networks (DCN). The de facto scheme, Equal Cost MultiPath (ECMP) is well-known for its inferior performance due to the coarse-grained and congestion-oblivious nature. As such, more fine-grained and congestion-aware schemes recently emerge. However, current network-based schemes require special modifications to existing switch hardware that makes them hard to deploy. In addition, current edge-based schemes can not achieve optimal load balancing performance due to the lack of accurate in-network congestion information.**

We propose EMAN, as the first step to achieve optimal load balancing in DCN at the edge. Instead of trying to get the exact network congestion condition (hard to be done at the edge), EMAN directly distributes outbound traffic from a sending endhost proportional to bandwidth of each path, to balance the path utilization. By dynamically updating each paths *available bandwidth* using feedback from the receiver, EMAN can gracefully react to network asymmetries caused by failure and flow competition. Both testbed and simulation results show that EMAN can improve the performance by up to 80% (testbed) compared to ECMP, and 66% (simulation) compared to the latest edge-based congestion-aware schemes. EMAN has been implemented as a hot-pluggable Linux kernel module which is transparent to existing applications and kernel TCP stack.

I. Introduction

To meet the increasingly stringent performance requirement, current data center networks (DCN) have been engineered to provide large bi-section bandwidth with a dense interconnect topology, which comprises many parallel paths between servers [1]–[4]. In practice, Equal Cost Multi Path (ECMP) [5] is commonly used to balance traffic among these paths in production DCNs [2], [4]. However, it is well-known that ECMP does not perform well due to its *coarse-grained (flow-based)* and *congestion-oblivious (static hash)* load balancing manner, which can waste more than 50% network bandwidth [6]. As such, to solve this problem, several works [7]–[10] have recently emerged proposing more fine-grained congestion-aware schemes.

While it is relatively easy to find a more fine-grained load balancing unit (*e.g.* flowlet [7], [8], flowcell [11] or packet [9], [10]), it is much more challenging to balance load across multiple paths based on their dynamic congestion conditions. Previous works try to solve this problem along two directions:

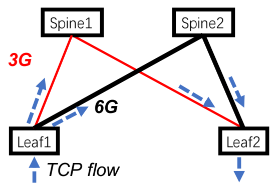
- **Network-based** schemes (*e.g.* [7], [12]) rely on in-network monitoring to get real-time traffic information on each switch port. As such, it can quickly distribute traffic according to each path's current congestion condition. For example, switches may tag information such as queue length and port utilization in the packet header, so the sender can balance traffic accordingly. However, this requires non-trivial switch hardware changes, which hinders their deployment.
- More recent works have proposed **edge-based** schemes (*e.g.* [8], [9]), which only require modifications to end-hosts' software. However, they have very limited path condition information thus can only surmise whether a path is congested or not based on common signals such as explicit congestion notification (ECN) [13] or round-trip-time (RTT). However, ECN or RTT signals do not specify the accurate congestion degree of a path, *i.e.*, how much traffic should be decreased/increased thus to exactly fully utilize this path. As such, current edge-based schemes just *tentatively spread less traffic on a path until the congestion signals disappear*, which makes their load balancing decisions inaccurate.

Thus from a philosophical standpoint, it is worth asking: *Can load balancing be done at the edge without modifying existing network hardware, while still distribute traffic to multiple paths in an accurate congestion-aware manner?* In this paper, we try to answer this question with a novel edge-based scheme called *EMAN* (*E*dge-based load balancing through *M*onitoring *A*vailable ba*N*dwidth). EMAN works based on a simple intuition: *Instead of heuristically increasing/decreasing traffic on a path after detecting congestion on it, EMAN sender directly distributes traffic proportionally to each path's bandwidth. This can balance the utilization of all parallel paths and reach the maximum throughput.*

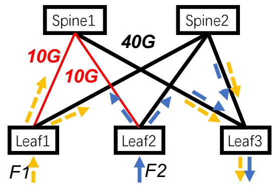
Although this intuition has been proven to be optimal in symmetric networks [10], real data center environment is much more challenging since asymmetric cases often happen [14], [15]. For example, link bandwidth may downgrade. In order to balance the utilization, the traffic distribution ratio needs to be quickly adjusted to the latest path bandwidth. What's more challenging, even we can quickly get the latest bandwidth, only distributing traffic based on path bandwidth but oblivious to other flows still cannot get us good performance. For example in Fig. 1(b), if we keep distributing flow F1 and F2 based on the paths' bandwidth ratio (*i.e.*, 10:40), these two flows both can only get 25 Gbps throughput (5 and 20 Gbps on each path), since link Spine 2↔Leaf 3 will be the bottleneck. However, the maximum throughput should be 60 Gbps.

To address above challenges, we extend the path bandwidth

*Corresponding author.
ISBN978-3-903176-28-7 2020 IFIP



(a) Simple asymmetric case.



(b) Asymmetric case with flow competition.

Fig. 1. Various asymmetric cases.

to a wider concept, called **available bandwidth**. The available bandwidth is defined as the bandwidth that a flow can get on a path considering link speed changes and other flows' competition[1]. Specifically, *the available bandwidth is the path's current physical capacity when the flow is the only one on the path; And it becomes the flow's fair share to the path bandwidth when there are other flows competing with it*. Instead of tentatively decreasing sending rate after seeing congestion signals on a path, whose performance is very sensitive to the predefined rate-decreasing amount and the delay of congestion feedback, *EMAN directly distributes traffic proportional to the path available bandwidth*. As such, it loosens the requirement of accurate real-time congestion degree which is hard to get at the edge.

To dynamically monitor each path's available bandwidth, EMAN leverages the collaboration of the receiver and the existing ECN signals. Particularly, by tagging path ID into each packet at the sender, the receiver can count the arrived packets and feedback the real-time transmission rate on each path back to the sender. Since ECN signals indicates that a path has currently been saturated (either this flow alone saturates it or this flow has saturated its fair-share bandwidth), EMAN can detect a path's available bandwidth from the current transmission rate when receiving ECNs.

The major contributions of this work are summarized as follows:

- We present EMAN, a novel edge-based congestion-aware load balancing scheme for DCN. By monitoring path available bandwidth using the information from ECN and the receiver, EMAN can well balance the network load at the edge without in-network monitoring.
- We have implemented EMAN as a hot-pluggable Linux kernel module based on netfilter [18] framework, which is transparent to applications and system stack. The implementation incurs little CPU processing overhead ($\sim$8% of a 2 GHz CPU core).
- Micro and macro benchmarks in simulation and testbed show that EMAN can improve the performance by up to 80% compared to ECMP, and 66% compared to the latest edge-based congestion-aware schemes.

II. Motivation and Insight

Before we go into the detailed design, we first overview the motivation and design insights of EMAN. Specifically:

[1]This paper considers load balancing for TCP flows since it contributes the most majority traffic (if not all) in DCNs [16], [17].

1) We first introduce how existing edge-based load balancing schemes detect congestion and distribute traffic, and then analyze why their performance is inferior through a simple example;
2) Next, we show that load balancing based on path bandwidth can well address the problem and achieve superior performance, but facing practical challenges;
3) Finally, we discuss the possible solutions to above challenges, and come out with our idea of load balancing based on *available bandwidth* at the edge.

A. Why existing edge-based schemes not enough?

Clove [8] and Hermes [9] are two representative edge-based load balancing schemes. Different from network-based schemes, they have no in-network information to accurately detect the congestion degree in real-time. As such, they use RTT or ECN signals to surmise a path's congestion condition:

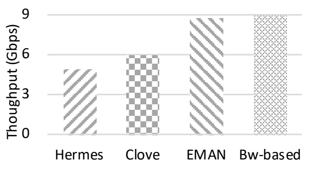
- Clove maintains a weight for each parallel path and distributes traffic on each path proportionally to its weight. At the initial state, each path has an equal weight. When ECN appears on a path which indicates congestion, Clove will decrease the path's weight with a certain amount (Clove paper recommends to decrease by 1/3), so less traffic will be distributed on this path.
- Hermes combines both ECN and RTT to estimate a path's condition. A path is considered to be good when both the frequency of ECN and RTT are within some threshold. Packets of a flow will be sent to the same good path unless the path is detected to be not good, and switched to another good path[2].

We use the following simple experiment to detailedly analyze Clove and Hermes' behavior. Consider the case shown in Fig. 1(a). There are two parallel paths in the network, where one's bandwidth is 6 Gbps, and the other's bandwidth downgrades to 3 Gbps due to failure. A TCP flow is passing through these two paths, coming from switch Leaf 1 and destined to Leaf 2. We assume that the input and output bandwidth is big enough and the bottleneck of the flow's throughput would only appear in the network paths.

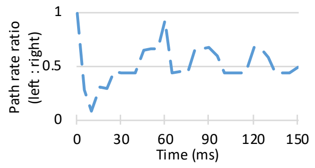
We use NS2 [19] to simulate above scenario and evaluate the performance using Clove and Hermes for load balancing, respectively (detailed simulation settings in Sec. V). Fig. 2(a) shows the average flow throughput. We can see that both Hermes and Clove can only utilize less than 70% of the whole network capacity (9 Gbps). The detailed reasons of their inferior performance are as follows:

- Clove initially distributes traffic to these two paths with a ratio of 1:1 (equal weight). After the TCP flow's throughput increases to 6 Gbps (*i.e.* 3 Gbps for each path), it will get ECN from the left path. Now the weight of left path will decay to $\frac{2}{3}$. Later, the traffic will be distributed to the two paths with a ratio of 2:3 ($\frac{2}{3}$:1). The

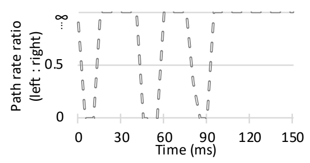
[2]More specifically, Hermes will not switch path too aggressively, and only switch flow with enough bytes left to be sent and the current path has very low speed, thus to ensure that there will be performance gain considering the packets out-of-order after switching path.



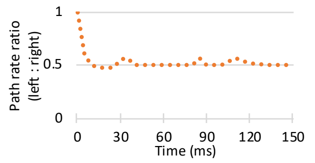
(a) Throughput.



(b) Traffic rate ratio on the two paths: Clove



(c) Traffic rate ratio on the two paths: Hermes



(d) Traffic rate ratio on the two paths: EMAN

Fig. 2. Simple asymmetric case (Fig. 1(a)): The flow throughput and the ratio of traffic rate on the two paths (left path rate : right path rate).

TCP flow will not decrease its throughput now (Clove will intercept ECN until it sees ECN on all parallel paths) and the current rate on the two paths become 2.4 Gbps and 3.6 Gbps, respectively. Next, when the flow grows up to 7.5 Gbps (3 Gbps on the left and 4.5 Gbps on the right), the left path will get ECN again and its weight will decay to $\frac{4}{9}$. So the rate on the two paths are 2.3 Gbps and 5.2 Gbps respectively. Following this procedure, the right path will finally get ECN when the flow throughput reaches 8.6 Gbps (2.6 Gbps on the left and 6 Gbps on the right), and the TCP flow starts to decrease its rate. After that, the weight of these two paths return to 2:3 ($\frac{4}{9}$:$\frac{2}{3}$). Therefore, the traffic ratio among these two paths will continue varying between 4:9 and 2:3 as shown in Fig. 2(b) (the pattern starts from time $\sim$25 ms), but do not converge to the right ratio of 1:2. This explains why Clove can not fully utilize these two paths simultaneously.

- Hermes will randomly pick up one path at the initial status, since no path is estimated as bad path. Next, when the flow's throughput grows up and saturates that path, ECN appears and RTT grows up. As such, this path will be considered as bad path and traffic will be switched to the other path. Such greedy path selection scheme can only choose one path during a period of time. As we can see in Fig. 2(c), traffic oscillates between these two paths[3]. As such, Hermes also can not fully utilize the two paths simultaneously.

To summarize, although ECN and RTT can reflect congestion, **it is hard to tell the exact congestion degree** only with these two signals. Particularly, ECN will only appear on a path or a path's RTT will only grow up evidently when the path is already saturated and congested[4]. As such, switching to a new path only after these signals appear is often too late, since TCP flows will have already decreased their throughput reacting to congestion. Moreover, even one like Clove can deliberately intercept and hide the ECN signals, we do not know how much load should be switched from this path and how much load other paths can accept.

[3]Note that to show the oscillation clear, we do not enable the parameter R in Hermes here. We set R to be 40% in all the rest experiments.

[4]Note that the small variance of RTT may be caused by host stack's processing delay, so only evidently increasing of RTT can be considered as congestion signals.

B. Balance the parallel paths' utilization

Actually, it is not difficult to show that if a load balancing scheme *keeps the utilization of all parallel paths always equal*, the TCP flow can get maximal throughput which reaches the aggregate bandwidth of all paths (assuming the packets out-of-order has been handled). Apparently, in above simple case (Fig.1(a)), we can reach this by *always distributing traffic to parallel paths with the proportion of their bandwidth*. The results in Fig. 2(a) (denoted as Bw-based) show that such traffic distribution manner indeed reaches the optimal performance.

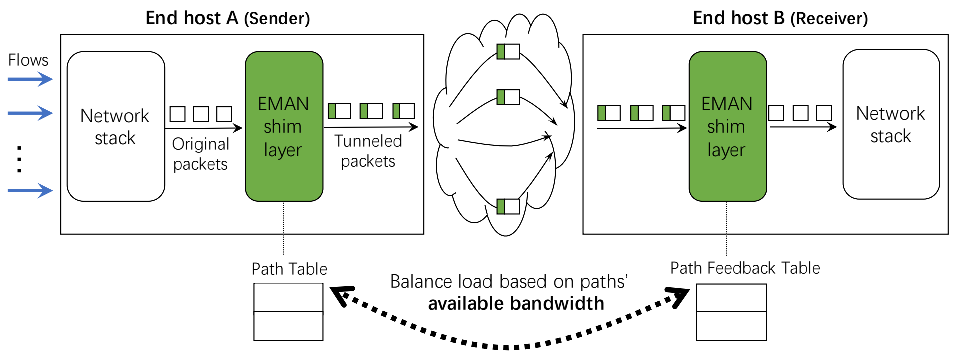
However, balancing traffic according to path bandwidth faces practical challenges in real DCN, especially at the edge:

- Path bandwidth would often change, so we need to get the accurate path bandwidth quickly. Particularly, failure events are norm rather than exception in large-scale DCNs [14], [15], [20]. Links can often have speed degradation or be totally broken due to failures. Waiting for routing plane to collect global view is always too slow (*e.g.*, seconds and minutes level [4], [6], [21]).
- A path may be contented by multiple flows between different pairs of end hosts, so the bandwidth is shared. It would cause unbalanced load if a flow only distributes traffic according to the path bandwidth without considering other flows' influence (It's not practical for different end hosts to get each other's real-time traffic information in data-center scale). For example, in Fig. 1(b), if flow F1 and F2 both distribute traffic with ratio of 1:4 (10 G:40 G) but oblivious to each other's existence, they will both get ECN back and slow down when the flow speed only reaches 25 Gbps, since the right path is congested (F1 and F2 each contributes 20 Gbps, respectively).

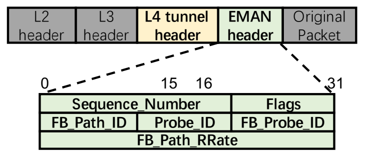
C. Load balancing based on available bandwidth

Based on above discussion, we can see that a dynamic *available bandwidth* probing scheme is necessary. As introduced before (§I), the *available bandwidth* is the actual bandwidth a flow can get on a path, considering the impact of failure and other flows' contention. Specifically in our EMAN, *a path's available bandwidth is detected as its current transmission rate when receiving ECNs*. We now discuss that why splitting traffic proportionally to the available bandwidth can well balance the network utilization (experiments in §V):

- It can quickly react to link speed degradation. For example in Fig. 1(a), before failure happens (all links are 6 Gbps), the TCP flow distributes traffic equally to the two paths, and the total throughput is 12 Gbps. The transmission rate on each path is 6 Gbps. After the left link downgrades from 6 Gbps to 3 Gbps, the left link



(a) Overview.



(b) Packet format.

Fig. 3. EMAN architecture.

will soon get ECN back. Since the transmission rate will also downgrade to the path capacity (*i.e.* 3 Gbps), then the sender will update its available bandwidth to 3 Gbps, and distribute traffic with the right ratio (*i.e.* 1:2).

- It can also quickly react to flow contention. For example in Fig. 1(b), the two flows first distribute their traffic according to the two paths' capacity, *i.e.* 10:40. Then the right path will get ECN when both flows' speed grow up to 25 Gbps. Now F1 and F2 each has 5 Gbps on the left and 20 Gbps on the right. Then both flow update their available bandwidth of the right path to 20 Gbps, and distributes later traffic with ratio of 10:20. And later on, each flow can reach 30 Gbps and simultaneously fully utilize the two paths (10 Gbps on the left path and 20 Gbps on the right).

Next, we will detailedly introduce how we design EMAN based on above intuition.

III. EMAN Design

A. Overview

Fig. 3(a) shows the architecture of EMAN. EMAN works as a transparent shim layer below the end hosts' existing network stack. It pro-actively selects path for each packet by encapsulating/decapsulating packet through overlay tunnels.

Specifically, since current DCN fabric uses ECMP for underlying multipath routing (hash the 5-tuple), we can pro-actively select network path at the end hosts by changing the packets' 5-tuple. So at the sender side, for path selection, EMAN encapsulates each packet with a new layer 4 (L4) tunnel header, and existing techniques (*e.g.*, traceroute [8] and XPath [22]) are leveraged to enforce explicit routing path control. EMAN maintains a path table for each pair of communication ends, tracking each path's available bandwidth, and dynamically distributes traffic to parallel paths according to the table. Path information can be shared among all transport flows between the same sender and the receiver.

At the receiver side, EMAN decapsulates the tunnel header and restores the packets coming from different paths into their original sending order, and then passes them to the host stack. EMAN receiver maintains a path feedback table for each pair of communication ends, and feeds paths' information back to the sender for calculating the path available bandwidth.

EMAN can handle multiple communication pairs by simply duplicating the processing scheme for a single pair, so for clarity of presentation (except where otherwise noted), our discussion is in the context of a single pair. Note that all EMAN end hosts are bi-directional, *i.e.*, EMAN shim layer deals with sending and receiving packets simultaneously. We only consider TCP traffic in this paper which is the vast majority in production DCNs [4], [17], [23].

B. Packet format

As Fig. 3(b) shows, EMAN encapsulates a packet into a transport (L4) tunnel (*i.e.* UDP or TCP) for path selection. There are plenty of mature tunnel techniques which are widely used in DCNs [24], [25], so we omit the introduction of the tunnel header here.

An EMAN header is also inserted after the L4 tunnel header which carries necessary information for monitoring each path's available bandwidth. We briefly introduce the EMAN header fields here, and discuss how to use them for making load balancing decisions later in more details. Note that EMAN receiver piggybacks path information in data packets back to the sender, so EMAN header simultaneously contains information both set by the sender and receiver. Specifically, EMAN header contains the following fields.

- **Sequence_Number (24 bits)**: This field uniquely identifies an EMAN packet between a communication pair. It is set by the sender and incremented by one for each packet sent out between this pair.
- **Flags (8 bits)**: It contains three flag bits, **FBE**, **P** and **FBP**. FBE is set by the receiver, feeding back whether this path has recently encountered ECN signals. P is set by the sender, indicating whether it is a rate probing packet. FBP is set by the receiver, indicating whether this packet contains the feedback probing rate.
- **FB_Path_ID (16 bits)** and **FB_Path_RRate (32 bits)**: These two fields are set by the receiver to feedback the current receiving rate on a certain path. FB_Path_ID identifies a path, and FB_Path_RRate is the receiving rate.
- **Probe_ID (8 bits)**, **FB_Probe_ID (8 bits)**: Probe_ID is only valid when P bit is set. This field is set by the sender, uniquely identifying a probing round. FB_Probe_ID is only valid when FBP bit is set. This field is set by the

receiver, feeding back the current probing round and its receiving rate (in field FB_Path_RRate).

C. EMAN sender

Path selection: At the sender, EMAN records the current available bandwidth for each path in the path table, and selects path for each packet based on it. Specifically, for each packet, EMAN will select a path in a smooth weighted round-robin manner using the path's available bandwidth as its weight.

Detecting transmission rate: As introduced before, to measure a path's available bandwidth, EMAN needs to measure the transmission rate on it. Specifically, whenever EMAN receiver sees an ECN or probing packet on a path, it starts to record the number of bytes received from this path for the successive $C\_ratecalc$ packets. Then it calculates the actual packet receiving rate during this time period and feedbacks it to the sender (in the FB_Path_ID and FB_Path_RRate fields), then stops recording for this path. EMAN relies on instant ECN marking as described in [23]. Note that an ECN packet will reset and restart the rate calculating period even when it has been enabled by a probing packet, since the receiving rate can reflect the available bandwidth more accurately when ECN indicates that the path has been fully utilized.

Updating available bandwidth: Then we can dynamically monitor each path's available bandwidth based on its current transmission rate. Specifically, a path's available bandwidth is initialized to be its physical capacity, and will be updated in the following two conditions:

- **Seeing ECNs:** When ECN appears on a path (*i.e.*, FBE bit is set in feedback packets), its available bandwidth is updated to its current transmission rate.
- **Proactive probing:** The available bandwidth updated by ECNs may get outdated. For example, a path's capacity may restore after failure or the competing flows may disappear. If we keep distributing traffic to this path using a relatively low weight, the path may never get ECN anymore since other paths may always be the traffic bottleneck. So it is necessary to re-probe the available bandwidth if ECNs have not be seen on a path for a relatively long time. Specifically, EMAN will proactively probe a path after a timeout (denoted as $T\_explore$) if not seeing ECNs. During the probing, the sender will force $C\_ratecalc$ successive packets going through the target path. Each probing round has a unique ID which is tagged into the header field Probe_ID. Then according to the feedback information (FB_Path_RRate), the sender updates its available bandwidth.

D. EMAN receiver

The receiver side is relatively easy. Specifically, when EMAN receiver receives a packet, it first decapsulates the packet and updates the path feedback table. The path feedback table records the ECN and probing signals and the number of bytes received on each path, which are used to calculate a path's receiving rate.

After that, the receiver hands over the packet to the host network stack if it is in-order, or otherwise, it puts the packet in the reorder buffer and hands it over to the host stack later when the former packets have arrived. If former packets have not arrived for a certain period of time, EMAN will hand over all packets in the reorder buffer and let the host stack to deal with the out-of-order packets.

When there is packet sending back from the receiver to the sender, EMAN will choose one path's information in the feedback table in a round-robin manner to feedback (as in [7], path information that recently changed can also be preferred as an optimization).

E. Parameter settings

One beautiful point of EMAN is that it has very few handcrafted parameters. Specifically, EMAN has only three groups of parameters: 1) $C\_ratecalc$. $C\_ratecalc$ is suggested to be just large enough to accurately monitor a path's transmission rate, which in our practice is $\sim$20 packets. 2) $T\_explore$, the timeout length for proactive probing. If path selection is proportional to all paths' available bandwidth, ECN will periodically appear on all paths since the TCP flow can saturate all parallel paths. So $T\_explore$ should be set longer than the time that TCP flows need to increase the throughput to saturate a path. According to the congestion control behavior, in the worst case (*i.e.*, there is a single flow and its cwnd linearly grows from 1), it needs BDP rounds of RTT to saturate a path (*e.g.*, $\sim$50 ms for 10 Gbps bandwidth and 250 us RTT). 3) L, the size of reorder buffer, and $T_{reorder}$, the maximum time for waiting out-of-order packets in the reorder buffer. L needs to be just large enough to cover the normal out-of-order degree across parallel paths, which is typically tens of MBs in production DCNs (*i.e.*, total buffer size on a network path [23]). Similarly, $T_{reorder}$ needs to be just large enough to cover the delay variance across paths, which is typically less than 10 ms in a 10 Gbps network.

IV. Implementation

We have implemented EMAN as a Linux kernel module which can be hot-plugged and hot-unplugged when the system is running. Our implementation has about 2K lines of code. Specifically, EMAN is built based on netfilter [18] framework. It intercepts the outbound and inbound packets using netfilter LOCAL_OUT hook (catch packets from L3 to L2 in the stack) and LOCAL_IN hook (from L2 to L3), respectively.

For the outbound packets, we first insert a UDP header (as the L4 tunnel header) and the EMAN header before the original packet. We choose UDP tunnel here since it is very simple and incurs small header length overhead. Since the original IP header is tunneled into the payload, we also copy the original IP header and insert it before the packet as the new IP header (with the length updated and checksum recalculated). Then, we fill the newly inserted UDP and EMAN header according to the path table, and send the packets out. The UDP source port is used as the identification of a network path, and we leverage the traceroute [8] to explicitly pre-map the source ports to physical paths.

For the inbound packets, we just decapuslate the packets and update the path feedback table, then remove the outer IP header, UDP tunnel header, and EMAN header. Note that the ECN signal (if any) is tagged in the outer IP header (TOS field), so we also copy the ECN field into the original IP header. Then, EMAN will reorder the out-of-order packets in a reorder buffer, and hand over the original packets to the stack.

V. Evaluation

Our evaluation mainly shows the following three points:

- Detailed simulation experiments show that EMAN can gracefully handle various asymmetric network conditions.
- Larger simulation experiments using real workload traces show that EMAN can significantly improve the load balancing performance in real DCNs.
- Testbed experiments show that our EMAN implementation has very small system overhead and can well balance traffic in practice.

A. Simulation experiments

Simulation setup: We implement EMAN as well as the latest edge-based schemes Clove [8] and Hermes[5] [9] on NS2 [19] simulation platform. We do not compare with network-based schemes such as CONGA [7], since they require network hardware modification which are not deployment friendly. The base RTT in our simulated network is 160 us. Correspondingly, the ECN threshold and the queue length are set to be 1 BDP and 3 BDPs, respectively [27]. TCP minRTO is 5 ms. Without explicitly specified, we use the following setups for each method: 1) For EMAN, $C_{ratecalc}$ is 10 packets. $T_{explore}$ is 50x network base RTT. L is 6Mb and $T_{reorder}$ is 5ms. 2) For Clove, the path weight decay ratio is 2/3, and the ECN hiding time length is 200x RTTs. 3) For Hermes, T_{ecn} is 40%. T_{RTT_low} and T_{RTT_high} are 20 us + RTT and 2 RTTs, respectively. S and R are 600 KB and 30%.

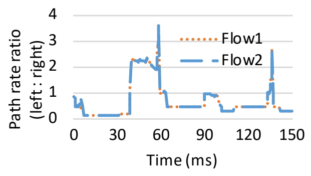
*1) **Simple asymmetric case:*** We evaluate EMAN using the same simple asymmetric case in Fig. 1(a). Results in Fig. 2 show that EMAN can quickly probe the bandwidth of the downgraded link and get the right traffic distribution ratio. Specifically, the traffic ratio is initialized to 1:1. Then the two paths will get ECN back successively and their available bandwidth is updated. After that, EMAN converges to the right traffic ratio of 1:2. As such, EMAN get almost the same throughput as the optimal bandwidth-based method in this case, which is ∼40% and ∼90% better than Clove and Hermes, respectively. Note that the slight ratio churn during transmission is caused by the small inaccuracy of active probing and receiving rate measurement.

*2) **Asymmetric case with flow competition:*** Next, we evaluate EMAN using the more complicated asymmetric case

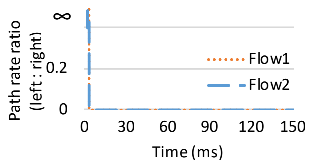
[5]Our NS2 implementation has largely referred to the NS3 [26] simulation code provided by the Hermes authors.

Method	Clove	Hermes	EMAN
Thoughput (Gbps)	32.5	39.5	56.7

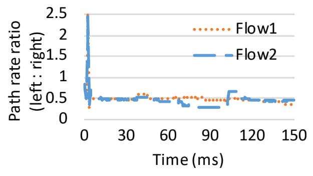
(a) Throughput.



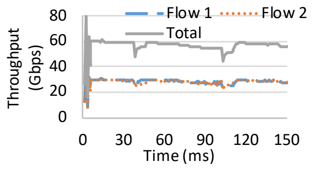
(b) Traffic on the two paths: Clove.



(c) Traffic on the two paths: Hermes.



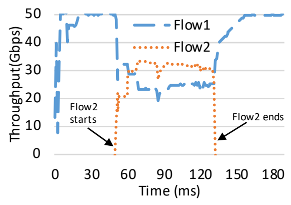
(d) Traffic on the two paths: EMAN.



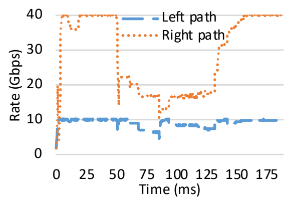
(e) Flow throughput of EMAN.

Fig. 4. Asymmetric case with flow competition (Fig. 1(b)): The aggregate throughput of the two flows and each flow's rate ratio on the two paths (left path rate : right path rate).

in Fig. 1(b). Fig. 4 shows the results. EMAN almost achieves the maximum throughput (57.4 out of 60 Gbps), which is ∼46% and ∼97% better than Hermes and Clove, respectively. EMAN distributes traffic just as described before (§II-C). It first sends traffic with ratio of 1:1 to the two paths, and converges to 1:2 after several rounds of ECN update and active probing. The big spike of traffic ratio at the initial state appears when one path gets ECN back but the other is still using the initial pre-configured available bandwidth. Hermes only uses the 40 Gbps path in this case, since it is considered much better than the other 10 Gbps link and the current rate is above the threshold (0.4×40 Gbps) for switching paths. Clove does not behave well since it always can not decay to the right weight for both paths. Fig. 4(e) shows the throughput of the two flows and their aggregate throughput as the time changes in EMAN.



(a) Throughput.



(b) Flow 1's rate on each path.

Fig. 5. Quick reaction to network dynamics.

*3) **Quick reaction to network dynamics:*** We now analyze how fast EMAN reacts to network dynamics. We use the same topology as before (Fig. 1(b)). Different from the former experiments, we let F1 keep running and let F2 join in the middle and leave the network after a while. Fig. 5 shows the throughput of the two flows and flow 1's (F1) rate on each path as the time grows. We can see that before F2 starts, F1 converges to the maximum throughput (*i.e.*, 50 Gbps). When F2 joins, F1 quickly adjusts the traffic distribution ratio (from

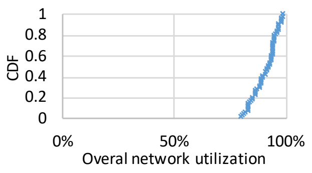


Fig. 6. CDF of the overall network utilization under various asymmetric cases.

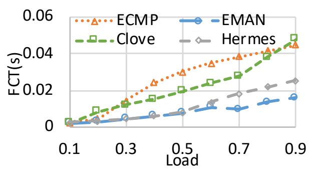


Fig. 7. Large-scale simulations: The average flow completion time.

1:4 to 1:2) and the two flows gradually converge to 30 Gbps. After F2 leaves, F1 restores its traffic distribution ratio back to 1:4 and achieves 50 Gbps again in about 30 ms.

*4) **Overall throughput under various asymmetric cases:*** We now enumerate all the link speed downgrade conditions. Specifically, we consider all the combinations of the total six links in Fig. 1(b) downgraded to 10Gbps. We evaluate how EMAN performs under various asymmetric cases. Fig. 6 shows the CDF of the overall network utilization under all the cases. EMAN achieves more than 85% utilization in more than 80% cases and achieves 80% utilization even under the most extreme cases.

*5) **Larger networks:*** We simulate a larger leaf-spine network, which have 4 leaf switches each with 16 servers connected through a 10G access link. Each leaf switch has 4 40G upward links connected with 4 spine switches, respectively. Two links (one between leaf 2↔spine 1 and the other leaf 2↔spine 2) downgrade to 10 Gbps due to failure. This is the largest scale for packet-level simulation that can be finished in acceptable time on our simulation servers. To evaluate the load balancing performance, we generate permutation traffic from each server in the left half of the network to the right half, with flow size sampled from two real data center workloads, web-search [23] and data-mining [2]. Flow arrives at the Poisson process and we adjust the flow inter arrival time to generate different load (from 0.1 to 0.9). Fig. 7 shows the average FCT under web-search workload for different loads. EMAN behaves the best at all the loads, improving the FCT by ∼5%-66% compared to Clove and Hermes, and ∼17%-75% compared to ECMP. We only show web-search results here due to space limitation, but EMAN also has similar better performance than Clove and Hermes under symmetric and asymmetric conditions for data-mining workload, respectively.

B. Testbed experiments

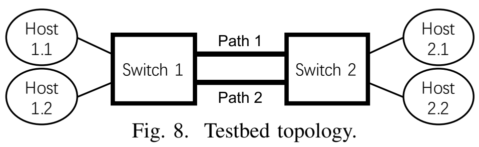


Fig. 8. Testbed topology.

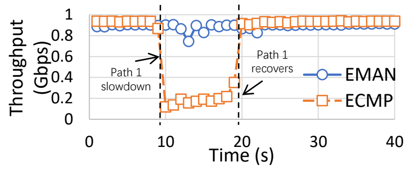
Testbed setup: We build a small testbed as shown in Fig. 8. It consists of two switches with two 1 Gbps parallel paths between them. The two switches are logical switches through configuring two VRFs within one physical Arista 7050S-64 switch. ECMP is enabled between the two parallel paths. There are two end hosts behind each switch. Each end host is a virtual machine (VM) running Ubuntu 18.04.1 system. Host 1.1/1.2 and 2.1/2.2 are located on two physical machines (Dell R720, E5-2620 2 GHz CPU, 4-port Intel Gigabit NIC), respectively. Each host VM is allocated with 1 dedicated CPU core, and 1 Gbps NIC port connected to the switch. The end-to-end RTT between hosts behind the two switches are ∼1 ms, so we configure the ECN marking threshold to be 128 KB on each path. We set EMAN's parameters $C\_ratecalc$ to be 10, $T\_explore$ to be 1 s, L to be 500 packets and $T\_reorder$ to be 100 ms. We generate TCP traffic using iPerf [28] tools, with MSS set to be 1350 (reserve space for EMAN headers). We do not compare Clove [8] and Hermes [9] in our testbed because their implementation are not available.

*1) **System overhead:*** To evaluate the overhead of our system implementation, we first install EMAN kernel module on host 1.1 and 2.1, and generate traffic with maximum speed between them. Compared to original TCP, results show that EMAN incurs almost zero CPU overhead on the sender side. EMAN receiver side needs to do more computation including rate calculation and packet reordering. However, on the receiver side, EMAN also only incurs ∼8% extra CPU utilization on a single CPU core (2 GHz) VM. As for the extra packet header, EMAN incurs ∼3% throughput overhead compared to original TCP (original TCP ∼930 Mbps and EMAN ∼900 Mbps)[6].

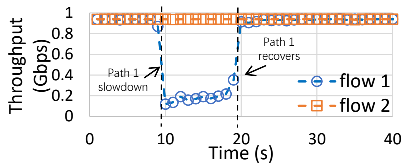
*2) **Load balancing according to path condition:*** During the TCP transmission, we generate a UDP flow with 500 Mbps constant rate on path 1 to emulate a link failure (last for 10 seconds), and examine whether EMAN can balance load according to the path condition.

- **Without flow competition.** We first consider the case that there is only one flow in the network. We generate one TCP flow from host 1.1 to 2.1 with maximum speed, and evaluate the flow throughput before, during, and after the UDP flow appears on path 1. Fig. 9(a) shows the results. The flow throughput in EMAN only drops temporarily when the UDP flow joins, since it encounters ECN. Then EMAN quickly switches most of the flow traffic to path 2 (1:2 on the two paths) and its throughput recovers. The flow throughput drops again just before the UDP flow ends, because the flow grows up to ∼900 Mbps and path 1 encounters ECN again (path 1 is saturated by the UDP plus TCP flow). We also evaluate the original ECMP's performance for comparison. Fig. 9(a) shows that, when the UDP flow occupies the path bandwidth, the TCP flow on this path drops to about 200 Mbps (note that TCP flow cannot fairly compete with a UDP flow).
- **With flow competition.** We now consider a more complex case that multiple flows competing within the network. Specifically, we generate two TCP flows from host 1.1/1.2 to 2.1/2.2 with maximum speed, respectively. Now the network does not only encounter path speed degradation (caused by outside UDP flow), but also will be congested by the two flows' competition. Fig. 9(b) and Fig. 9(c) show the two flows' throughput of ECMP and

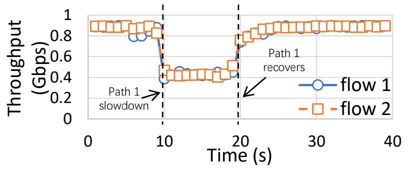
[6]This is the application throughput excluding all the packet headers.



(a) Without flow competition.



(b) With flow competition: ECMP.



(c) With flow competition: EMAN.

Fig. 9. Testbed experiment results.

EMAN, before, during, and after the UDP flow appears on path 1. In ECMP, we deliberately force the two flows going through two different paths. Results show that the throughput of flow 1 (on path 1) drops to about 200 Mbps when the UDP flow appears. However, for EMAN, The two flows evenly utilize the two paths before path 1's speed downgrades, and both switch to path 2 when the UDP flow appears on path 1. As such, both of the flow can remain about 460 Mbps during path 1's speed degradation. Note that since TCP can not fairly compete with UDP, EMAN will actively move traffic to the other path since path 1 frequently receives ECN signals. As such, the two flows only have a relatively low throughput on path 1, having the overall throughput (about 920 Mbps) a slightly higher than one path's maximum throughput (about 900 Mbps considering packet header overhead). However, in real cases when link speed slowdown due to failure instead of emulated by UDP, EMAN can gracefully utilize all available paths' bandwidth, as shown before (§V-A). We do not evaluate the real link-slowdown since our switch ports do not support such feature.

VI. Discussion

Per-packet processing overhead. EMAN requires per-packet processing to achieve fine-grained load balancing, which may raise concern of processing overhead. However, our evaluation shows that such overhead is negligible in 1Gbps environment (§V-B1). Moreover, previous work [9]–[11], [29] show that with careful implementation, per-packet processing can well support 10Gbps link speed. As EMAN works as a shim layer transparent to applications and network stacks, we can offload EMAN onto NICs. The deployment of high programmable NICs in DCNs [30]–[32] offers us a great chance to implement EMAN. We will study it in the future.

Paths with large delay/bandwidth difference. If failures lead to large delay/bandwidth difference among multiple paths, EMAN can adaptively trim those worst paths, thus to avoid using too many buffer spaces at the receiver side to reorder packets. Since EMAN always monitor each path's condition with feedback from the receiving end, it is easy to find out those extremely bad paths and wipe them out.

VII. Related Work

According to the capability of congestion awareness, load balancing works for DCNs can be classified as two types.

Static load balancing such as [11], [33], [34] try to split traffic in very fine-grained unit to balance the network load, but being oblivious to path congestion. They behave well in symmetric topology. However, real data center networks often encounter asymmetry, where their performance are inferior.

Congestion-aware load balancing distribute traffic according to paths' dynamic congestion condition, which can be further classified into two types. 1) Centralized congestion-aware scheme such as [6], [35], [36] use centralized way to monitor the whole network condition and then schedule traffic accordingly. However, the centralized monitoring and scheduling incurs long latency, which makes them hard to handle the fast network dynamics. 2) Distributed congestion-aware scheme such as [10], [12], [37] as well as the aforementioned [7]–[9] monitor network condition and schedule traffic distributively in each end host (edge-based) or switch (network-based). However, as introduced before, network-based works require non-trivial modifications to network hardware [7], [10], [12], and edge-based works can not achieve good performance due to the lack of in-network information [8], [9], [37].

Besides, multi-path transport [38]–[40] is another direction to balance traffic, which splits a flow to multiple sub-flows. However, it requires significant changes to transport stack, which is hard to deploy for public clouds hosting client VMs.

VIII. Conclusion

Load balancing for data center networks is an active research area with plenty of ongoing works. However, current works either require non-trivial network hardware modification, or can not achieve good load balancing performance only with endhost's software changes. Our proposed EMAN, an important complement to this area, is just a hot-pluggable endhost kernel module which is transparent to both applications and system stacks, meanwhile, can well balance traffic based on network paths' available bandwidth. Our testbed and simulation experiments show that EMAN greatly improves the performance compared to existing edge-based schemes.

Acknowledgements

We thank Zeyu Zhang and Kuan Cheng for helping us to improve the design and analyze existing works. We thank the anonymous reviewers of IFIP Networking for their comments. This research was partially funded by the National Natural Science Foundation of China (Grants No.6187060280), Tencent Rhino-Bird Open Research Fund, and the Fundamental Research Funds for the Central Universities.

References

[1] Chuanxiong Guo, Guohan Lu, Dan Li, Haitao Wu, Xuan Zhang, Yunfeng Shi, Chen Tian, Yongguang Zhang, and Songwu Lu. BCube: A high performance, server-centric network architecture for modular data centers. In *SIGCOMM*, 2009.

[2] Albert Greenberg, James R. Hamilton, Navendu Jain, Srikanth Kandula, Changhoon Kim, Parantap Lahiri, David A. Maltz, Parveen Patel, and Sudipta Sengupta. VL2: A scalable and flexible data center network. In *SIGCOMM*, 2009.

[3] Mohammad Al-Fares, Alexander Loukissas, and Amin Vahdat. A scalable, commodity data center network architecture. In *SIGCOMM*, 2008.

[4] Arjun Singh, Joon Ong, Amit Agarwal, Glen Anderson, Ashby Armistead, Roy Bannon, Seb Boving, Gaurav Desai, Bob Felderman, Paulie Germano, Anand Kanagala, Jeff Provost, Jason Simmons, Eiichi Tanda, Jim Wanderer, Urs Hoelzle, Stephen Stuart, and Amin Vahdat. Jupiter rising: A decade of clos topologies and centralized control in google's datacenter network. In *SIGCOMM*, 2015.

[5] Christian E Hopps. *Analysis of an equal-cost multi-path algorithm*. RFC 2992, 2000.

[6] Mohammad Al-Fares, Sivasankar Radhakrishnan, Barath Raghavan, Nelson Huang, and Amin Vahdat. Hedera: Dynamic flow scheduling for data center networks. In *USENIX NSDI*, 2010.

[7] Mohammad Alizadeh, Tom Edsall, Sarang Dharmapurikar, Ramanan Vaidyanathan, Kevin Chu, Andy Fingerhut, Vinh The Lam, Francis Matus, Rong Pan, Navindra Yadav, and George Varghese. CONGA: Distributed congestion-aware load balancing for datacenters. In *SIGCOMM*, 2014.

[8] Naga Katta, Aditi Ghag, Mukesh Hira, Isaac Keslassy, Aran Bergman, Changhoon Kim, and Jennifer Rexford. Clove: Congestion-Aware Load Balancing at the Virtual Edge. In *CoNEXT*, 2017.

[9] Hong Zhang, Junxue Zhang, Wei Bai, Kai Chen, and Mosharaf Chowdhury. Resilient Datacenter Load Balancing in the Wild. In *SIGCOMM*, 2017.

[10] Soudeh Ghorbani, Zibin Yang, P. Brighten Godfrey, Yashar Ganjali, and Amin Firoozshahian. DRILL: Micro Load Balancing for Low-latency Data Center Networks. In *SIGCOMM*, 2017.

[11] Keqiang He, Eric Rozner, Kanak Agarwal, Wes Felter, John Carter, and Aditya Akella. Presto: Edge-based load balancing for fast datacenter networks. In *the ACM SIGCOMM 2015 Conference*, pages 465–478. ACM, 2015.

[12] Peng Wang, Hong Xu, Zhixiong Niu, Dongsu Han, and Yongqiang Xiong. Expeditus: Congestion-aware load balancing in clos data center networks. In *Proceedings of the Seventh ACM Symposium on Cloud Computing*, SoCC '16, pages 442–455, New York, NY, USA, 2016. ACM.

[13] Kadangode Ramakrishnan and Sally Floyd. *A proposal to add explicit congestion notification (ECN) to IP*. RFC 2481, January 1999.

[14] Phillipa Gill, Navendu Jain, and Nachiappan Nagappan. Understanding network failures in data centers: Measurement, analysis, and implications. In *the ACM SIGCOMM 2011 Conference*, volume 41, pages 350–361. ACM, 2011.

[15] Chuanxiong Guo, Lihua Yuan, Dong Xiang, Yingnong Dang, Ray Huang, Dave Maltz, Zhaoyi Liu, Vin Wang, Bin Pang, Hua Chen, Zhi-Wei Lin, and Varugis Kurien. Pingmesh: A large-scale system for data center network latency measurement and analysis. In *the ACM SIGCOMM 2015 Conference*, SIGCOMM '15, pages 63–74, New York, NY, USA, 2015. ACM.

[16] Theophilus Benson, Ashok Anand, Aditya Akella, and Ming Zhang. Understanding data center traffic characteristics. *ACM SIGCOMM Computer Communication Review*, 40(1):92–99, 2010.

[17] Glenn Judd. Attaining the promise and avoiding the pitfalls of TCP in the datacenter. In *the USENIX NSDI 2015 Conference*, pages 145–157, Oakland, CA, May 2015. USENIX Association.

[18] The netfilter.org project. https://www.netfilter.org/.

[19] The Network Simulator - ns-2. *isi.edu/nsnam/ns/*.

[20] Guo Chen, Yuanwei Lu, Yuan Meng, Bojie Li, Kun Tan, Dan Pei, Peng Cheng, Layong Luo, Yongqiang Xiong, Xiaoliang Wang, et al. Fast and Cautious: Leveraging Multi-path Diversity for Transport Loss Recovery in Data Centers. In *USENIX Annual Technical Conference*, pages 29–42, 2016.

[21] Meg Walraed-Sullivan, Amin Vahdat, and Keith Marzullo. Aspen trees: Balancing data center fault tolerance, scalability and cost. In *the ACM CoNEXT 2013 Conference*, 2013.

[22] Shuihai Hu, Kai Chen, Haitao Wu, Wei Bai, Chang Lan, Hao Wang, Hongze Zhao, and Chuanxiong Guo. Explicit path control in commodity data centers: Design and applications. In *the USENIX NSDI 2015 Conference*, pages 15–28, Oakland, CA, May 2015. USENIX Association.

[23] Mohammad Alizadeh, Albert Greenberg, David A. Maltz, Jitendra Padhye, Parveen Patel, Balaji Prabhakar, Sudipta Sengupta, and Murari Sridharan. Data Center TCP (DCTCP). In *the ACM SIGCOMM 2010 Conference*, SIGCOMM '10, pages 63–74, New York, NY, USA, 2010. ACM.

[24] M Sridharan, K Duda, I Ganga, A Greenberg, G Lin, M Pearson, P Thaler, C Tumuluri, N Venkataramiah, and Y Wang. Nvgre: Network virtualization using generic routing encapsulation. *IETF draft*, 2011.

[25] Mallik Mahalingam, D Dutt, Kenneth Duda, Puneet Agarwal, Lawrence Kreeger, T Sridhar, Mike Bursell, and Chris Wright. Vxlan: A framework for overlaying virtualized layer 2 networks over layer 3 networks. *draftmahalingam-dutt-dcops-vxlan-01. txt*, 2012.

[26] ns-3. http://www.nsnam.org/.

[27] Haitao Wu, Jiabo Ju, Guohan Lu, Chuanxiong Guo, Yongqiang Xiong, and Yongguang Zhang. Tuning ECN for Data Center Networks. In *the ACM CoNEXT 2012 Conference*, CoNEXT '12, pages 25–36, New York, NY, USA, 2012. ACM.

[28] iPerf - The network bandwidth measurement tool. https://iperf.fr/.

[29] Bojie Li, Tianyi Cui, Zibo Wang, Wei Bai, and Lintao Zhang. SocksDirect: Datacenter Sockets can be Fast and Compatible. In *SIGCOMM*, 2019.

[30] Daniel Firestone, Andrew Putnam, Sambhrama Mundkur, Derek Chiou, Alireza Dabagh, Mike Andrewartha, Hari Angepat, Vivek Bhanu, Adrian Caulfield, Eric Chung, et al. Azure accelerated networking: SmartNICs in the public cloud. In *15th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 18)*, pages 51–66, 2018.

[31] Adrian M Caulfield, Eric S Chung, Andrew Putnam, Hari Angepat, Jeremy Fowers, Michael Haselman, Stephen Heil, Matt Humphrey, Puneet Kaur, Joo-Young Kim, et al. A Cloud-Scale Acceleration Architecture. In *Microarchitecture (MICRO), 2016 49th Annual IEEE/ACM International Symposium on*, pages 1–13. IEEE, 2016.

[32] Andrew Putnam, Adrian M Caulfield, Eric S Chung, Derek Chiou, Kypros Constantinides, John Demme, Hadi Esmaeilzadeh, Jeremy Fowers, Gopi Prashanth Gopal, Jan Gray, et al. A reconfigurable fabric for accelerating large-scale datacenter services. In *Computer Architecture (ISCA), 2014 ACM/IEEE 41st International Symposium on*, pages 13–24. IEEE, 2014.

[33] Jiaxin Cao, Rui Xia, Pengkun Yang, Chuanxiong Guo, Guohan Lu, Lihua Yuan, Yixin Zheng, Haitao Wu, Yongqiang Xiong, and Dave Maltz. Per-packet load-balanced, low-latency routing for clos-based data center networks. In *the ACM CoNEXT 2013 Conference*, CoNEXT '13, pages 49–60, New York, NY, USA, 2013. ACM.

[34] Abhishek Dixit, Pawan Prakash, Yu Charlie Hu, and Ramana Rao Kompella. On the impact of packet spraying in data center networks. In *the IEEE INFOCOM 2013 Conference*, pages 2130–2138. IEEE, 2013.

[35] Andrew R Curtis, Wonho Kim, and Praveen Yalagandula. Mahout: Low-overhead datacenter traffic management using end-host-based elephant detection. In *the IEEE INFOCOM 2011 Conference*, pages 1629–1637. IEEE, 2011.

[36] Theophilus Benson, Ashok Anand, Aditya Akella, and Ming Zhang. MicroTE: Fine grained traffic engineering for data centers. In *the ACM CoNEXT 2011 Conference*, page 8. ACM, 2011.

[37] Jonathan Perry, Hari Balakrishnan, and Devavrat Shah. Flowtune: Flowlet Control for Datacenter Networks. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*, pages 421–435, Boston, MA, 2017. USENIX Association.

[38] Costin Raiciu, Sebastien Barre, Christopher Pluntke, Adam Greenhalgh, Damon Wischik, and Mark Handley. Improving datacenter performance and robustness with multipath TCP. In *the ACM SIGCOMM 2011 Conference*, SIGCOMM '11, pages 266–277, New York, NY, USA, 2011. ACM.

[39] Yuanwei Lu, Guo Chen, Bojie Li, Kun Tan, Yongqiang Xiong, Peng Cheng, Jiansong Zhang, Enhong Chen, and Thomas Moscibroda. Multi-Path Transport for RDMA in Datacenters. In *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*, Renton, WA, 2018. USENIX Association.

[40] G. Chen, Y. Lu, B. Li, K. Tan, Y. Xiong, P. Cheng, J. Zhang, and T. Moscibroda. MP-RDMA: Enabling RDMA With Multi-Path Transport in Datacenters. *IEEE/ACM Transactions on Networking*, 27(6):2308–2323, 2019.