Resilient Datacenter Load Balancing in the Wild

Hong Zhang[1], Junxue Zhang[1], Wei Bai[1], Kai Chen[1], Mosharaf Chowdhury[2]
[1]SING Lab, Hong Kong University of Science and Technology
[2]University of Michigan

ABSTRACT

Production datacenters operate under various uncertainties such as traffic dynamics, topology asymmetry, and failures. Therefore, datacenter load balancing schemes must be resilient to these uncertainties; i.e., they should accurately sense path conditions and timely react to mitigate the fallouts. Despite significant efforts, prior solutions have important drawbacks. On the one hand, solutions such as Presto and DRB are oblivious to path conditions and blindly reroute at fixed granularity. On the other hand, solutions such as CONGA and CLOVE can sense congestion, but they can only reroute when flowlets emerge; thus, they cannot always react timely to uncertainties. To make things worse, these solutions fail to detect/handle failures such as blackholes and random packet drops, which greatly degrades their performance.

In this paper, we introduce Hermes, a datacenter load balancer that is resilient to the aforementioned uncertainties. At its heart, Hermes leverages comprehensive sensing to detect path conditions including failures unattended before, and it reacts using timely yet cautious rerouting. Hermes is a practical edge-based solution with no switch modification. We have implemented Hermes with commodity switches and evaluated it through both testbed experiments and large-scale simulations. Our results show that Hermes achieves comparable performance to CONGA and Presto in normal cases, and well handles uncertainties: under asymmetries, Hermes achieves up to 10% and 20% better flow completion time (FCT) than CONGA and CLOVE; under switch failures, it outperforms all other schemes by over 32%.

CCS CONCEPTS

• **Networks → Network architectures**; **End nodes**; **Data center networks**; Data path algorithms;

KEYWORDS

Datacenter fabric; Load balancing; Distributed

ACM Reference format:
Hong Zhang, Junxue Zhang, Wei Bai, Kai Chen, and Mosharaf Chowdhury. 2017. Resilient Datacenter Load Balancing in the Wild. In *Proceedings of SIGCOMM '17, Los Angeles, CA, USA, August 21-25, 2017,* 14 pages.
https://doi.org/10.1145/3098822.3098841

1 INTRODUCTION

Modern datacenter networks enable multiple paths between host pairs and balance traffic among them to deliver good performance to different bandwidth- and latency-sensitive datacenter applications [3, 4, 18]. Meanwhile, production datacenters operate under a multitude of uncertainties, such as congestion, asymmetry, and failures [5, 10, 14, 17]. Uncertainties arise due to a variety of reasons that include, among others, traffic dynamics, link cuts, device heterogeneity, and switch malfunctions [19]. Naturally, a datacenter load balancer must adapt to these uncertainties; i.e., they should (1) accurately sense path conditions, and (2) appropriately split traffic among parallel paths in reaction to path conditions.

However, the standard multi-path load balancing mechanism used in today's datacenters, ECMP (Equal Cost Multi-Path) [21], balances traffic poorly. ECMP randomly stripes flows across available paths using flow hashing. Because it accounts for neither network uncertainties nor flow sizes, it can waste over 50% of the bisection bandwidth [4].

As a result, many load balancing schemes have been proposed to address the problem. Despite significant efforts, prior solutions still have important drawbacks (see §2 and §7 for details). Some of them, such as DRB [12] and Presto [20], are insensitive or even oblivious to path conditions and blindly split traffic at fixed (e.g., packet or flowcell) granularity. These solutions suffer in the presence of asymmetry because the optimal traffic splitting across parallel paths depends on traffic demands and path conditions [5, 14], and schemes that lack visibility of path conditions are unable to make optimal decisions. Furthermore, blindly splitting flows onto different paths on a round-robin basis can adversely affect transport protocols. Besides the well-known packet reordering problem, we further unveil the relatively less understood congestion mismatch problem (§2.2.2).

In contrast, other solutions such as CONGA [5] and CLOVE [24] are designed to be congestion-aware. Although they can sense congestion, these solutions only reroute when flowlets emerge. While flowlet switching helps to minimize packet re-ordering [33], it is inflexible. As the formation of flowlets is decided by many factors such as applications and transport protocols, flowlet-based solutions are inherently passive and cannot always timely react to congestion by rerouting flows when needed. We show that this can easily result in ~50% performance loss (§2.2.2).

Furthermore, existing approaches for congestion sensing have key shortcomings. They are either impractical because they require non-trivial switch modifications [5, 25, 35], or inefficient because they provide limited visibility into network congestion [23, 24]. Moreover, none of them can detect switch failures such as packet blackholes and random packet drops, which are frequently witnessed in production datacenters [19]. Our experiments show that being unable to detect these failures may lead to unfinished flows and over 100× worse average FCT.

Given the above inefficiencies, we ask the following question: *can we design a resilient load balancing scheme that can gracefully handle all these uncertainties in a practical, readily-deployable fashion?* In this paper, we present Hermes to answer this question affirmatively.

At its heart, Hermes detects path conditions via comprehensive sensing (§3.1). It makes use of transport-level signals such as ECN and RTT to measure path congestion, while leveraging events such as retransmissions and timeouts to detect packet blackholes and random packet drops caused by malfunctioning switches. To further improve visibility, Hermes employs active probing – guided by the power of two choices technique [28] – that can effectively increase the scope of sensing at minimal probing cost.

Given path conditions, how to react to the perceived uncertainties is still non-trivial. Considering the passiveness of flowlets, a natural choice is to actively split flows at a finer granularity and always switching to the best available path instantly. However, our analysis reveals that such vigorous path changing, even coupled with comprehensive sensing, can interfere with transport protocols by causing frequent packet reordering and congestion mismatch problems.

Therefore, Hermes handles uncertainties timely yet cautiously (§3.2). On the one hand, rather than being constrained by a fixed or passive granularity, Hermes is capable of reacting timely once it senses congestion or failures. On the other hand, instead of vigorously changing paths, Hermes assesses flow status and path conditions, and makes deliberate rerouting decisions only if they bring performance gains. This enables Hermes to prune unnecessary reroutings and to reduce packet reordering and congestion mismatch, making it transport-friendly.

Hermes is a practical edge-based solution with no switch modifications. We implemented a Hermes prototype and deployed it in our small-scale testbed (§4). Testbed experiments as well as large-scale simulations with the realistic web-search [6] and data-mining [18] workloads show that (§5):

- Under symmetric topologies, Hermes achieves 10-38% better FCT than ECMP, outperforms CLOVE-ECN by up to 15% for both workloads, and achieves comparable performance to Presto and CONGA.
- Under asymmetric topologies, Hermes outperforms CONGA by up to 10% due to its timely rerouting when sensing congestion with the data-mining workload. Furthermore, with better visibility via active probing, it achieves up to 20% better FCT than solutions with limited visibility such as CLOVE-ECN and LetFlow [14].
- In case of switch failures, Hermes can effectively detect the failures, and it outperforms all other schemes by more than 32%.

2 BACKGROUND AND MOTIVATION

In this section, we first summarize different sources of uncertainties in datacenters (§2.1), and then elaborate the limitations of prior solutions in terms of both sensing and reacting to uncertainties (§2.2).

2.1 Uncertainties

Datacenters are filled with the following uncertainties:

- **Traffic dynamics:** As shown in previous studies [17], traffic in production datacenters can be highly dynamic. Congestion can quickly arise as a few high-rate flows start, and it can dissipate as they complete.
- **Asymmetries:** Asymmetries are the norm in datacenters. For example, topology asymmetry can arise as a datacenter evolves by adding racks and switches over time, which may also introduce coexistence of heterogenous devices (e.g., both 10G and 40G spine switches). Furthermore, it is reported that datacenters can experience frequent link cuts [17, 19], creating asymmetries.
- **Switch failures:** Besides link failures that directly cause topology asymmetry, production datacenters also suffer from a variety of switch failures or malfunctions that traditionally have received less attention. Notably, a recent study of Microsoft production datacenters [19] reveals two types of switch failures that adversely affect network performance: (1) *packet blackholes*: packets that meet certain 'patterns' (e.g., certain source-destination IP pairs, or port numbers) are dropped deterministically (i.e., 100%) by the switch; and (2) *silent random packet drops*: a switch drops packets silently and randomly at a high rate. The root causes of these switch failures include TCAM deficits in the switching ASIC, switching fabric CRC checksum errors, and linecards not being well seated, among others.

Load balancing traffic under these uncertainties is a challenge. To deal with traffic dynamics and asymmetries, an ideal solution needs to be congestion-aware. To handle failures, it needs to detect them quickly and accurately. Despite continuous efforts in recent years, prior solutions still have important drawbacks as we illustrate in the following.

2.2 Drawbacks of Prior Solutions in Handling Uncertainties

We follow Table 1 to discuss the drawbacks of prior solutions, motivating our design of Hermes.

2.2.1 *Limitations in Sensing Uncertainties.*

First of all, solutions such as DRB, Presto, and ECMP [12, 20, 21] are either oblivious to congestion or rely only on local traffic conditions. They perform poorly under asymmetry because the optimal traffic splitting across paths in asymmetric topologies depends primarily on traffic demands and path conditions. Schemes without global congestion-awareness cannot effectively balance the traffic [5, 14].

Second, existing solutions that are designed to be global congestion-aware [5, 23–25, 35] have drawbacks too. They are either impractical because they require non-trivial switch modifications [5, 25, 35] or inefficient because they only provide limited visibility into network congestion at the end hosts [23, 24].

To illustrate this, we quantify *network visibility* as the average number of concurrent flows observed on parallel paths. We run a trace-driven simulation based on web-search and data-mining workloads in a 8×8 leaf spine topology with 10 Gbps links and 128 servers. We measure the visibility of both ToR switches and end hosts for 2s. The results are shown in Table 2. We find that a source ToR switch can observe congestion status of several parallel paths to each destination ToR simultaneously, while the same is not the case for end host pairs. Overall, current end host based solutions [23, 24],

Schemes	Sensing Uncertainties		Reacting to Uncertainties		Advanced Hardware
	Congestion	Switch Failure	Minimum Switchable Unit	Switching Method and Frequency	
ECMP [21] Presto [20] DRB [12]	Oblivious	Oblivious	Flow Flowcell (small fixed-sized unit) Packet	Per-flow random hashing Per-flowcell round robin Per-packet round robin	No
LetFLow [14]	Oblivious	Oblivious	Flowlet	Per-flowlet random hashing	Yes[1]
DRILL [16]	Local awareness (Switch)	Oblivious	Packet	Per-packet rerouting (according to local congestion)	Yes
CONGA [5] HULA [25] CLOVE-INT [24]	Global awareness (Switch)	Oblivious	Flowlet	Per-flowlet rerouting (according to global congestion)	Yes
FlowBender [23] CLOVE-ECN [24]	Global awareness (End host)	Oblivious	Packet Flowlet	Reactive and random rerouting (when congested) Per-flowlet weighted round robing (according to global congestion)	No
Hermes	Global awareness (End host)	Aware	Packet	Timely yet cautious rerouting (based on global congestion and failure)	No

Table 1: Summary of prior work under uncertainties. Note that unlike link failures that directly cause topology asymmetry, here switch failure refers to malfunctions such as packet blackholes and silent random packet drops.

Workload	Data-mining 60% load	Data-mining 80% load	Web-search 60% load	Web-search 80% load
Switch pair	1.725	2.344	4.173	5.859
End host pair	0.007	0.009	0.016	0.022

Table 2: The average number of concurrent flows observed on parallel paths between ToR-to-ToR, host-to-host pairs.

which rely primarily on piggybacked information, lack sufficient visibility for making appropriate load balancing decisions.

Finally, none of the existing schemes detect switch failures such as packet blackholes and silent random packet drops. We note that some solutions such as Presto [20] and DRB [12] rely on a central controller to detect link failures; however, they do not detect switch failures. Moreover, current congestion-aware solutions such as CONGA [5] estimate congestion level by measuring link utilization; thus, they cannot effectively detect switch failures either. Consider a switch that experiences silent random packet drops; flows traversing this switch tend to have a low sending rate due to frequent packet drops. Then the corresponding paths experiencing switch failures will have even lower measured congestion levels compared to other parallel paths. As a consequence, existing congestion-aware solutions may shift more traffic to these undesirable paths. In our evaluation, this causes CONGA to perform worse than ECMP (§5.3.3).

2.2.2 *Problems with Reacting to Uncertainties.*

Flowlet switching cannot timely react to uncertainties: Many congestion-aware solutions [5, 24, 25] adopt flowlet switching. The benefit is that flowlets provide a finer-granularity alternative to flows for load balancing without causing much packet reordering. However, because flowlets are decided by many factors such as applications and transport protocols, solutions relying on flowlet switching are inherently passive and cannot always timely react to congestion by splitting traffic when needed.

Figure 1 shows such an example with CONGA [5]. We have two small flows (A, B) and two large flows (C, D) from L0 to L1 via parallel paths P_1 and P_2. We use DCTCP [6] as the underlying transport protocol. At the beginning, CONGA balances load by placing flow

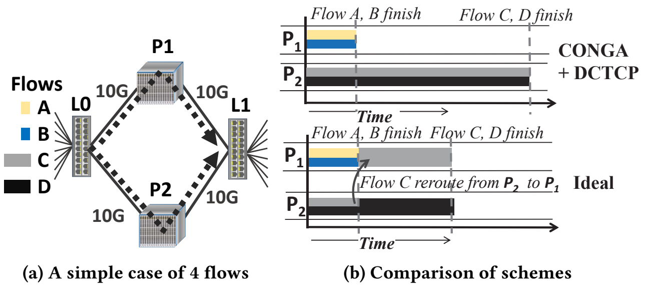


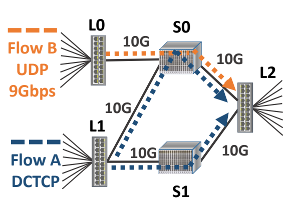
Figure 1: [Example 1] Flowlet switching cannot timely react to congestion by splitting flows under stable traffic pattern.

A, B on P_1 and C, D on P_2. After flow A and B complete, CONGA senses that P_1 is idle. However, when using a flowlet timeout of 100-500μs [5], CONGA cannot identify flowlets for rerouting, mainly because DCTCP is less bursty – this is because DCTCP adjusts its congestion window smoothly, and thus, it is less likely to generate sufficient inactivity gaps that form flowlets. On the other hand, a smaller flowlet timeout value (e.g., 50μs [14]) triggers frequent flipping, causing severe packet reordering in our simulation. In the ideal scenario, appropriate rerouting can almost halve the FCTs of the large flows.

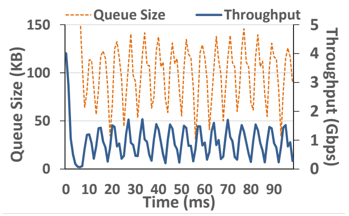
Vigorous rerouting is harmful: Considering the deficiency of flowlets, a natural choice is to split flows at a finer granularity and always switching to the best available path instantly. However, such vigorous path changing, even coupled with congestion awareness, can adversely interfere with transport protocols. Besides the well-known packet re-ordering problem, we further unveil the *congestion mismatch* problem (defined below) that has previously received little attention.

Basically, congestion control algorithms adjust the rate (window) of a flow based on the congestion state of the *current* path. Hence, each rerouting event can cause a mismatch between the sending rate and the state of the *new* path. With vigorous rerouting within a flow, the congestion states of different paths are mixed together, and congestion on one path may be mistakenly used to adjust the rate on another path. We refer to this phenomenon as congestion mismatch.

[1] LetFlow [14] has been implemented for Cisco's datacenter switch production line; however, currently it is not widely supported by commodity switches. For example, most Broadcom switching chipsets do not support flowlet switching.

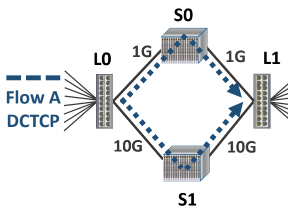


(a) Asymmetry due to broken link L0-S1

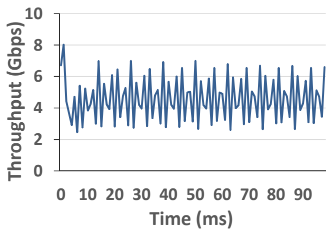


(b) Queue length of S0-L2 and throughput of flow A

Figure 2: [Example 2] Congestion mismatch results in severe throughput loss and queue oscillation under asymmetric topologies with Presto.



(a) A 1/10G hybrid network



(b) Throughput of flow A

Figure 3: [Example 3] Distributing loads according to link capacities does not solve the congestion mismatch problem.

In the following, we show three cases of congestion mismatch and their impacts. Although we use DCTCP as the transport protocol, the problems demonstrated are not tied to any specific transport protocol. To mask the throughput loss caused by packet reordering, we set `DupAckThreshold` to 500.

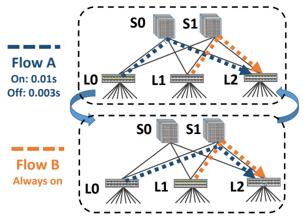
First, we show that for congestion-oblivious load balancing schemes with small granularity (e.g., Presto [20] and DRB [12]), congestion mismatch causes throughput loss and queue oscillations under asymmetries. Consider a simple 3×2 leaf-spine topology in Figure 2a[2] with a broken link from L0 to S1. Flow A is a DCTCP flow from L1 to L2, and flow B is a UDP flow from L0 to L2 with a limited rate of 9 Gbps. We adopt Presto with equal weights for different paths.

As shown in Figure 2b, flow A only achieves around 1 Gbps overall throughput, and the queue length of the output port of spine S0 to leaf L2 experiences large variations. Due to vigorous rerouting, the congestion feedback (i.e., ECN) of the upper path constrains the congestion window, resulting in throughput loss in the bottom path. Furthermore, when flow A with a larger window shifts from the bottom path to the upper path, the upper path cannot immediately absorb such a burst, causing queue length oscillations.

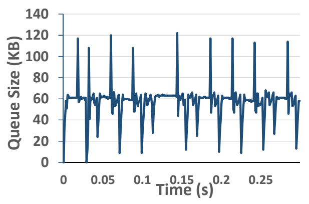
Second, we show that congestion mismatch remains harmful even if we distribute traffic proportionally to path capacity. To illustrate this, consider a heterogenous network with 1 and 10 Gbps paths shown in Figure 3a. We spread flowcells using 1:10 ratio to match path capacities and expect both paths to be fully utilized.

However, as shown in Figure 3b, flow A can only achieve an overall throughput of around 5 Gbps. To understand the reason, assume that the first 10 flowcells go through the 10 Gbps path. Because the

[2]We flip leaf L2 and omit the unused paths for clarity.



(a) Hidden terminal scenario



(b) Queue length of S1-L2

Figure 4: [Example 4] The hidden terminal scenario: (a) causes flow A to flip between spine S0 and S1 with stale information; (b) causes sudden queue build-up every time when flow A reroutes through S1.

path is not congested, DCTCP will increase the congestion window without realizing that the subsequent flowcell will go through the 1 Gbps path. With a large congestion window, the 11th flowcell is sent at a high rate on the 1 Gbps path, causing a rapid queue buildup on the output port of spine S0 to leaf L2. As the queue length exceeds the ECN marking threshold (i.e., 32KB for 1 Gbps link), DCTCP will reduce the window upon receiving ECN-marked ACKs without realizing that such a reduction will affect the following flowcells on the 10 Gbps path. As a result, such a congestion mismatch still causes throughput loss and queue oscillations.

Third, we show that for congestion-aware solutions, suboptimal rerouting also leads to severe congestion mismatch. Figure 4a shows such an example with CONGA. Flow A starts from L0 to L2, and we add a 3ms pause every 10ms to create a flowlet timeout. Flow B keeps sending from L1 to L2. We find that flow A keeps flipping between spine S0 and S1. This is because no matter which path flow A chooses, it does not have explicit feedback packets on the alternative path; thus, it always assumes the alternative path to be empty after an aging period (i.e., 10ms as suggested in [5]). As a result, flow A will aggressively reroute to the alternative path every time it observes a flowlet.

We note that imperfect congestion information is unavoidable in a distributed setting. However, current congestion-aware solutions [5, 25] do not take this into account, and their aggressive rerouting may lead to congestion mismatch. As shown in Figure 4b, each time flow A reroutes from spine S0 to S1, its sending rate is much higher than the desired rate at the new path; this causes an acute queue increase at the output port of S1-L2. These periodic spikes in queue occupancy can lead to high tail latencies for small flows or even packet drops when buffer-hungry protocols (e.g., TCP) are used. In large-scale simulations with production workloads (§5.3.2), we observe that CONGA performs 30% worse with a smaller flowlet timeout of 50μs compared to that of 150μs, even after we mask packet reordering. We believe that such performance degradations are due to congestion mismatch.

3 DESIGN

The limitations discussed in §2 highlight the following properties of an ideal load balancing solution:

- *Comprehensiveness*: it should effectively detect congestion and failures to guide load balancing decisions;

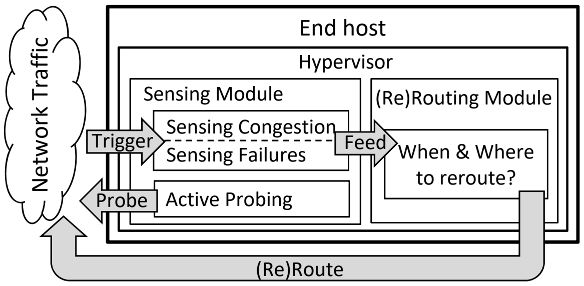


Figure 5: Hermes overview.

Flow-level Variable	
r_f	Sending rate of a flow
s_{sent}	Size sent of a flow, used to estimate the remaining size
$if_{timeout}$	Set if a flow experiences a timeout
Path-level Variable	
f_{ECN}	Fraction of ECN-marked packets of a path
t_{RTT}	RTT measurement of a path
$n_{timeout}$	Number of timeout events of a path
$f_{retransmission}$	Fraction of retransmission events of a path
r_p	Aggregate sending rate of all flows of a path ($\sum r_f$)
$type$	Characterization of path condition

Table 3: Variables in Hermes.

Parameter	Recommended Setting
T_{ECN}: threshold for fraction of ECN	40%
T_{RTT_low}: threshold for low RTT	$20-40\mu s$ + base RTT
T_{RTT_high}: threshold for high RTT	1.5×one hop delay + base RTT
Δ_{RTT}: threshold for notably better RTT	one hop delay
Δ_{ECN}: threshold for notably better ECN fraction	3-10%
R: the highest flow sending rate threshold for rerouting	20-40% of the link capacity
S: the smallest flow sent size threshold for rerouting	$100-800KB$
Probe interval	$100-500\mu s$

Table 4: Parameters in Hermes and recommended settings.

- *Timeliness*: it should quickly react to various uncertainties and make timely rerouting decisions;
- *Transport-friendliness*: it should limit its impact (i.e., packet reordering and congestion mismatch) on transport protocols;
- *Deployability*: it should be implementable with commodity hardware in current datacenter environments.

To this end, we propose Hermes, a resilient load balancing scheme to gracefully handle uncertainties. Figure 5 overviews the two main modules of Hermes: (1) the sensing module that is responsible for sensing path conditions; and (2) the rerouting module that is responsible for determining when and where to reroute the traffic.

Akin to previous work [9, 20, 24], we implement a Hermes instance running in the hypervisor of each end host, leveraging its programmability to enable comprehensive sensing (§3.1) and timely yet cautious rerouting (§3.2) to meet the aforementioned properties. Table 3 and 4 summarize the key variables and parameters used in the design and implementation of Hermes.

3.1 Comprehensive Sensing

3.1.1 Sensing Congestion.
Hermes leverages both RTT and ECN to detect path conditions:

Algorithm 1: Hermes Path Characterization

```
1 for each path p do
2     if $f_{ECN} < T_{ECN}$ and $t_{RTT} < T_{RTT\_low}$ then
3         $type = good$
4     else if $f_{ECN} > T_{ECN}$ and $t_{RTT} > T_{RTT\_high}$ then
5         $type = congested$
6     else
7         $type = gray$
8     if ($n_{timeout} > 3$ and no packet is ACKed) or
      ($f_{retransmission} > 1\%$ and $type \neq congested$) then
9         $type = failed$
```

ECN	RTT	Possible Cause	Characterization
High	High	Congested	Congested path
High	Moderate/Low	Not enough ECN samples or all delay is built up at one hop	Gray path
Low	High	Not enough ECN samples or the network stack incurs high RTT	
Low	Moderate	Moderately loaded	
Low	Low	Underutilized	Good path

Table 5: Outcome of path conditions using ECN and RTT.

- RTT directly signals the extent of end-to-end congestion. While it is informative, accurate RTT measurement is difficult in commodity datacenters without advanced NIC hardware [26]. Thus, in our implementation, we primarily use RTT to make a course-grained categorization of paths, i.e., to separate congested paths from uncongested ones. While a large RTT does not necessarily indicate a highly congested path (e.g., end host network stack delay can increase RTT), a small RTT is an indicator of an underutilized path.
- ECN captures congestion on individual hops. It is supported by commodity switches and widely used as congestion signal for many congestion control algorithms [6, 34]. Note that ECN is a binary signal that is marked when a local switch queue length exceeds a marking threshold, it can well capture the most congested hop along the path. However, in a highly loaded network where congestion may occur at multiple hops, ECN cannot reflect this. Furthermore, a low ECN marking rate does not necessarily indicate a vacant end-to-end path, especially when there are not enough samples.

Given that neither RTT nor ECN can accurately indicate the condition of an end-to-end path, to get the best of both, we combine these two signals using a set of simple guidelines in Algorithm 1. Specifically, we characterize a path to be *good* if both RTT measurement and ECN fraction are low. In contrast, if both RTT measurement and ECN fraction are high, we identify the path to be *congested* – this is because a large RTT value alone may be caused by the network stack latency at the end host, whereas a high ECN fraction alone from a small number of samples may be inaccurate as well. Otherwise, in all the other cases, we classify the path to be *gray*. Table 5 summarizes the outcome of the algorithm and reasons behind.

Scheme	Piggy-back ([23, 24])	Brute-force Probing	Power of two Choices	Hermes
Visibility	< 0.01	100	>3	>3
Overhead	NA	100×	3×	3%

Table 6: Comparison of different probing schemes in terms of visibility and corresponding overhead. Recall that visibility is quantified as the average number of concurrent flows a sender can observe on parallel paths to each end host (§2.2.1), and overhead is defined as the sending rate of the extra probe traffic introduced over the edge-leaf link capacity. Here we consider a 100×100 leaf-spine topology with 10^5 end hosts and 10Gbps link. A probe packet is typically 64 bytes and the probe interval is set to 500μs.

3.1.2 Sensing Failures.

Hermes leverages packet retransmission and timeout events to infer switch failures such as packet blackholes and random drops.[3]

Recall that a switch with blackholes will drop packets from certain source-destination IP pairs (or plus port numbers) deterministically [19]. To detect a blackhole of a certain source-destination pair, Hermes monitors flow timeouts on each path. Once it observes 3 timeouts on a path, it further checks if any of the packets on that path have been successfully ACKed. If none have been ACKed, Hermes concludes that all packets on this path are being dropped and identifies it as a failed path. Blackholes including port numbers can be detected in a similar way.

A switch experiencing silent random packet drops introduces high packet drop rates. To detect such failures, we observe that packet drops always trigger retransmissions, which can be captured by monitoring TCP sequence numbers. Based on this observation, Hermes records packet retransmissions on each path, and picks out paths experiencing high retransmission rates (e.g., 1% under DCTCP) for every τms. We set τ to be 10 by default. Congestion can cause frequent retransmissions as well. Therefore, Hermes further checks the congestion status, and identifies uncongested paths with high retransmission rates as failed paths (lines 8-9 in Algorithm 1).

3.1.3 Improving Visibility.

Hermes requires visibility into the network for good load balancing decisions. However, visibility comes at a cost. One can exhaustively probe all the paths to achieves full visibility, but it introduces high probing overhead – 100× the capacity of a 10Gbps link (Table 6). In contrast, piggybacking used in existing edge-based load balancing solutions [23, 24] incur little overhead, but it provides very limited visibility.

We seek a better tradeoff between visibility and probing overhead by leveraging the well-known power-of-two-choices technique [28]. Unlike DRILL [16] that applies this for packet routing within a switch, we adopt it at end hosts to probe path-wise information with the following considerations.

First, we find that there is no need to pursue congestion information for all the paths; instead, probing a small number of them can

[3]We note that existing diagnostic tools [19, 32] take at least 10s of seconds to detect and locate switch failures. As a result, they cannot be directly adopted by Hermes as load balancing requires fast reactions to these failures.

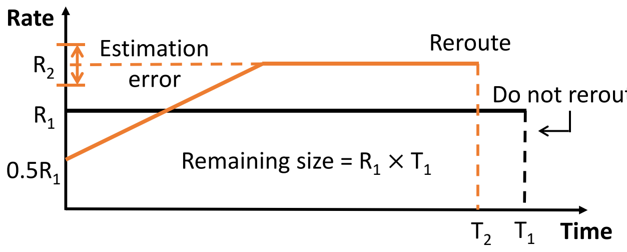


Figure 6: A simplified model to assess the cost and benefit of rerouting. Whether rerouting will shorten a flow's completion time depends on factors such as the rate difference between the new (R_2) and current (R_1) paths, the gradient of rate change, and the flow size.

effectively improve the load balancing performance with affordable probing cost. Second, all path-wise congestion signals have at least one RTT delay. Hence, probing only a small number of paths reduces the risk of many end hosts making identical choices and overwhelming one path while leaving others underutilized.[4] Third, in addition to the two random probes, we add an extra probe on the previously observed best path. This brings better stability [16, 28] and increases the chance of finding an underutilized path [29].

As shown in Table 6, with such a technique, Hermes ensures that each source can see the status of at least 3 parallel paths to its destination, which can effectively guide load balancing decisions. Meanwhile, it reduces the overhead by over 30× compared to the brute-force approach. To further reduce the overhead, Hermes picks one hypervisor under each rack as the *probe agent*. In each probe interval, these agents probe each other and share the probed information among all hypervisors under the same rack. This further reduces the overhead by 100×. Note that this approach can introduce inaccuracies when the last hop latencies to different hosts under the same rack are significantly different.

Overall, Hermes achieves over 300× better visibility than piggybacking. Its overhead is around 3%, which is over 3000× better than the brute-force approach.

3.2 Timely yet Cautious Rerouting

Even with comprehensive sensing, reacting to the perceived uncertainties is still non-trivial. As shown in §2.2.2, the challenges are two-fold: flowlet switching is passive and cannot always timely react to uncertainties, whereas vigorous rerouting adversely interferes with transport protocols. To address these challenges, Hermes employs timely yet cautious rerouting.

On the one hand, instead of passively waiting for flowlets, Hermes uses packet as the minimum *switchable* granularity so that it is capable of *timely* reacting to uncertainties. On the other hand, because vigorous rerouting can introduce congestion mismatch and packet reordering, Hermes tries to be transport-friendly by reducing the frequency of rerouting. Unlike prior solutions that perform random hashing [14, 21], round-robin [12, 20, 24], or always reroute to the best path [5], Hermes cautiously makes rerouting decisions based on both path conditions and flow status.

[4]This herd behavior caused by delayed information update is elaborated in [27].

Figure 6 illustrates a simplified cost-benefit assessment Hermes performs before making rerouting decisions. Consider a flow that has already sent some data, and due to changes in network conditions, we need to determine whether to reroute. Rerouting to a less congested path may result in packet reordering, which in turn can halve its sending rate ($R_1 \rightarrow \frac{1}{2}R_1$), assuming TCP fast recovery is triggered. In this case, whether a rerouting can improve the flow's completion time depends on many factors, such as the sending rate on the new path (R_2), the current path (R_1), and the remaining flow size. Note that it is a coarse-grained model because: (1) some metrics such as R_2 cannot be effectively measured; and (2) it does not accurately capture the cost of congestion mismatch caused by frequent rerouting. However, it highlights the need for caution and helps us identify the following heuristics for cautious rerouting.

First, considering the rate reduction caused by rerouting and the estimation error of R_2 shown in Figure 6, we note that rerouting is not always beneficial if R_2 is not significantly larger than R_1. As a result, Hermes reroutes a flow only when it finds a path with *notably* better condition (evaluated by an RTT threshold Δ_{RTT} and an ECN threshold Δ_{ECN} in our implementation) than the current path.

Second, it indicates that rerouting a flow with a small remaining size may bring limited benefit; this is because it may finish before its sending rate peaks. As a result, Hermes uses the size a flow already sent to estimate the remaining size [7, 30] and reroutes flows only when the size sent exceeds a threshold S.

Finally, although we cannot accurately measure R_2, we can measure R_1 by using CONGA's DRE algorithm [5]. If R_1 is already high, the potential benefit of rerouting is likely to be marginal; moreover, the cost would be high if we reroute to a wrong path (as shown in the example of Figure 4b). Therefore, we avoid rerouting a flow whose sending rate exceeds a threshold R.

To summarize, Algorithm 2 shows the rerouting logic of Hermes. It is timely triggered for every packet when (1) it belongs to a new flow or a flow experiencing failure/timeout; or (2) the current path is congested. In the former case (lines 3-12), Hermes first tries to select a good path with the least sending rate r_p in order to prevent local hotspots. If it fails, Hermes further checks gray paths. If it fails again, Hermes randomly chooses an available path with no failure. For the latter case (lines 13-23), Hermes first tries to evaluate the benefit of rerouting. As discussed above, Hermes checks the sending rate r_f and size sent S_{sent}, and decides to reroute only when both conditions for cautious rerouting are met (line 14). The following rerouting procedure is similar to that in the former case, except that the selected path should be notably better than the current path (lines 15 and 19). The flow stays on its original path if no better path is found.

Algorithm 2: Timely yet Cautious Rerouting

```
1  for every packet do
2      Assume its corresponding flow is $f$ and path is $p$
3      if $f$ is a new flow or $f.if_{timeout}$==true or $p.type == failed$
        then
4          {p'} = all good paths
5          if {p'} ≠ ∅ then
6              $p^* = Argmin_{p \in \{p'\}}(p.r_p)$
               /* Select a good path with the smallest
                  local sending rate                          */
7          else
8              {p''} = all gray paths
9              if {p''} ≠ ∅ then
10                 $p^* = Argmin_{p \in \{p''\}}(p.r_p)$
11             else
12                 $p^*$ = a randomly selected path with no failure
13     else if $p.type == congested$ then
14         if $f.s_{sent} > S$ and $f.r_f < R$ then
15             {p'} = all good paths notably better than $p$
               /* $\forall p' \in \{p'\}$, we have $p.t_{RTT} - p'.t_{RTT} > \Delta_{RTT}$
                  and $p.f_{ECN} - p'.f_{ECN} > \Delta_{ECN}$          */
16             if {p'} ≠ ∅ then
17                 $p^* = Argmin_{p \in \{p'\}}(p.r_p)$
18             else
19                 {p''} = all gray paths notably better than $p$
20                 if {p''} ≠ ∅ then
21                     $p^* = Argmin_{p \in \{p''\}}(p.r_p)$
22                 else
23                     $p^* = p$ /* Do not reroute                */
24     return $p^*$ /* The new routing path                      */
```

3.3 Parameter Settings

Hermes has a number of parameters as shown in Table 4. We now discuss how we set them up. Note that in this section we only provide several useful rules-of-thumb and leave (automatic) optimal parameter configuration as an important future work.

Recall that we use 3 parameters to infer congestion: T_{RTT_low}, T_{ECN} and T_{RTT_high} (§3.1). T_{RTT_low} is a threshold for good path; we set it to be 20-40μs (20μs by default) plus one-way base RTT to ensure that a good path is only lightly loaded. T_{ECN} is an indicator for congested path, we set it to be 40% to ensure the path is heavily loaded. To set T_{RTT_high}, we use per-hop delay as a guideline. Note that each fully loaded hop introduces a relatively stable delay. This value can be calculated by $\frac{ECN\ marking\ threshold}{Link\ capacity}$ with DCTCP [6]. We set T_{RTT_high} to be base RTT plus 1.5× of the one hop delay. Such a value suggests that the path is likely to be highly loaded at more than one hop. Due to different settings, T_{RTT_high} is configured to be 300μs in our testbed and 180μs in simulations. Sensitivity analysis in §5.4 shows that Hermes performs well with T_{RTT_high} between 140 to 280μs in simulations.

For probing, our evaluation suggests that an interval of 100-500μs brings good performance, and we set the default value as 500μs. For rerouting, the key idea is to ensure that each rerouting is indeed necessary. Δ_{RTT} is the RTT threshold to ensure a path is notably better than another; we set it to be the one hop delay (80μs in simulation and 120μs in testbed) to mask potential RTT measurement inaccuracies. Similarly, we set the ECN fraction threshold Δ_{ECN} to be 3-10% (5% by default). Moreover, we find that setting S to be 100-800KB can avoid rerouting small flows, and improve the FCT of small flows under high load. Also, setting R to be 20-40% of the link capacity can avoid rerouting flows with high sending rates, which effectively improves the FCT of large flows. The performance of Hermes is fairly stable with the above settings, and we set S to be 600KB and R to be 30% of the link capacity by default.

4 IMPLEMENTATION

We have implemented a Hermes prototype and deployed it in our testbed. We enforce the explicit routing path control at the end host using XPath [22]. Note that our implementation is compatible with legacy kernel network stacks and requires no modification to switch hardware.

End host module: The end host module is implemented as a Linux kernel module, residing between TCP/IP stacks and Linux `qdisc`. It can be installed and removed while the system is running without recompiling the OS kernel. The kernel module consists of four components: a `Netfilter` TX hook, a `Netfilter` RX hook, a hash-based flow table, and a path table. Note that it operates at both TX and RX paths.

The operations in the TX path are as follows: (1) The `Netfilter` TX hook intercepts all outgoing packets and forwards them to the flow table; (2) We identify the flow entry the packet belongs to and update its state (e.g., size sent, sending rate, sequence number, etc.); (3) Then we compute the packet's path using Hermes's rerouting algorithm; (4) Note that each path in XPath [22] framework can be identified by a unique IP address. So after obtaining the desired path, we add an IP header to the packet and write the desired path IP into the destination address field.

The operations in the RX path are as follows: (1) The `Netfilter` RX hook intercepts all incoming packets; (2) We identify the flow entry of this packet and update its flow state; (3) We identify the path this flow uses and update the path state; (4) After updating information, we remove the outer IP header and deliver this packet to the upper layer. In addition to the kernel module, we generate probe packets using a simple client-server application.

System overhead: To quantify the system overhead introduced by Hermes, we install it on a server with 4-core Intel E5-1410 2.8GHz CPU and a Broadcom BCM57810 NetXtreme II 10Gigabit Ethernet NIC. We generate more than 9Gbps of traffic with more than 5000 concurrent connections. The extra CPU overhead introduced is less than 2% compared to the case where the Hermes end host module is not running. The measured throughput remains the same in both cases.

Switch configuration: We configure L3 forwarding table, ECN/RED marking, and strict priority queueing (two priorities) at the switch. Inspired by previous work [26], we classify pure ACK packets in the reverse path into the high priority queue for more accurate RTT measurements.

5 EVALUATION

We evaluate Hermes using a combination of testbed experiments and large-scale ns-3 simulations. Our evaluation seeks to answer the following questions:

How does Hermes perform under a symmetric topology? Testbed experiments (§5.2) show that Hermes achieves 10-38% better FCT than ECMP, outperforms CLOVE-ECN by up to 15%, and achieves comparable performance to Presto. In a large simulated topology, we show that Hermes performs close to (and slightly better than) CONGA for the web-search (and data-mining) workload (§5.3.1).

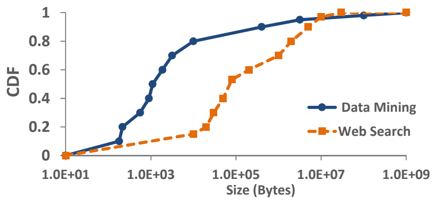


Figure 7: Traffic distributions used for evaluation.

How does Hermes perform under an asymmetric topology? Testbed experiments with a link cut show that Hermes performs 12-26% better than CLOVE-ECN (§5.2) and significantly outperforms Presto and ECMP at high loads. Moreover, we find that Hermes is very competitive in a large simulated topology with a high degree of asymmetry (§5.3.2): 1) For the web-search workload, Hermes is within 4-30% of CONGA. Compared to LetFlow and CLOVE-ECN, cautious rerouting in Hermes leads to up to 3.3× better FCTs for small flows at high loads; 2) For the data-mining workload, Hermes outperforms CONGA by up to 10% due to its more timely rerouting. With active probing, Hermes achieves up to 20% better FCT than prior solutions with limited visibility (e.g., CLOVE-ECN and LetFlow).

How effective is Hermes under switch failures? (§5.3.3) We simulate both silent random packet drop and packet blackhole scenarios, and our results show that Hermes can effectively sense and avoid both types of failures, achieving more than 32% better FCT compared to all other schemes.

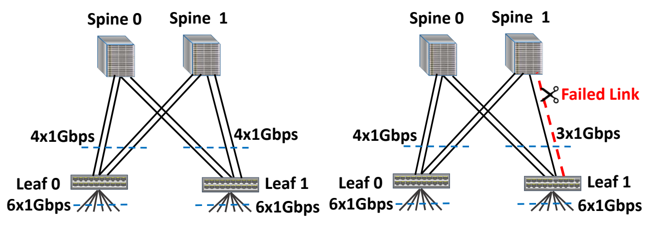
How does each design component contributes and how robust is Hermes under different parameter settings? (§5.4) We evaluate the benefits brought by good visibility and cautious yet timely rerouting separately. Results show that both components contribute to over 10% improvements in the overall outcomes. Moreover, we also find that Hermes' performance is stable under a variety of parameter settings.

5.1 Methodology

Transport: We use DCTCP as the default transport protocol. In our testbed, we use the DCTCP implementation in Linux kernel 3.18.11 [1]. In our simulator, we implement DCTCP on top of ns-3's TCP New Reno protocol, faithfully capturing its optimizations. The initial window is 10 packets. We set both initial and minimum value of TCP RTO to 10ms. We set other parameters as suggested in [6].

Schemes compared: Besides ECMP, we compare Hermes with the following solutions:

- **CONGA** [simulation] We simulate CONGA following the parameter settings in [5]. However, because DCTCP is less bursty than TCP, the default 500μs flowlet timeout value is too big. For a fair comparison against flowlet-based solutions, we tried different flowlet timeout values and adopted the best one (i.e., 150μs) in our simulations.
- **LetFlow** [simulation] We simulate LetFlow with a flowlet timeout value of 150μs (as we do for CONGA).



(a) The symmetric case (b) The asymmetric case

Figure 8: [Testbed] Topology used for testbed experiments.

- **Presto*** [testbed, simulation] Similar to [14], we implement a variant of Presto (Presto*) to isolate performance issues from those caused by packet reordering. We spray packets instead of flowcells, and implement a reordering buffer to mask packet reordering.
- **CLOVE-ECN** [testbed, simulation] We evaluated two versions of CLOVE: Edge-Flowlet and CLOVE-ECN via both testbed experiments and simulations. We only show the results of CLOVE-ECN since it slightly outperforms Edge-Flowlet in most cases. We do not simulate CLOVE-INT since it requires programmable switches and it is shown to be outperformed by CONGA [24]. We set a 150μs flowlet timeout value in simulations. For testbed, we pick the best flowlet timeout value (i.e., 800μs) after trying different values.

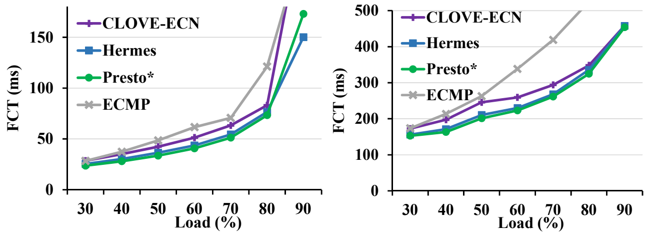
Remark: We do not compare against MPTCP [31] because there is a lack of a reliable ns-3 simulation package, and the latest publicly available MPTCP Linux release suffers from performance instability [20, 23]. Note that MPTCP suffers from incast issues and has been shown to be outperformed by CONGA in many cases similar to those we consider. Moreover, we have implemented FlowBender [23] on top of DCTCP in our testbed, and we follow the default settings in [23]. However, the performance is close to ECMP. One possible reason is that the workload or topology do not suit the default parameter settings. Hence, we omit the results of FlowBender to avoid unfair comparison.

Workloads: We use two widely-used realistic workloads observed from deployed datacenters: data-mining [18] and web-search [6]. As shown in Figure 7, both distributions are heavy-tailed. Particularly, the data-mining workload is more skewed with 95% of all data bytes belonging to about 3.6% of flows that are larger than 35MB, which makes it more challenging for load balancing [5]. We adopt the flow generator in [8], which generates flows between random senders and receivers under different leaf switches according to Poisson processes with varying traffic loads.

Metrics: We use flow completion time (FCT) as the primary performance metric. Besides the overall average FCT, we also break down the FCT for small flows (<100KB) and large flows (>10MB) in some cases to better understand the results. The results are the average of 5 runs.

5.2 Testbed Experiments

Testbed setup: Our testbed consists of 12 servers connected to 4 switches. As shown in Figure 8a, the servers are organized in two racks (6 servers each). We adopt the same topology as used in prior work [5, 14, 24], and all servers and switches are connected with 1Gbps links. Given that, there is a 3:2 oversubscription at the leaf level. We also consider an asymmetric case, where we cut one of the links between a leaf switch and a spine switch as indicated in Figure 8b.



(a) Web-search (b) Data-mining

Figure 9: [Testbed] Overall avg FCT in the symmetric case.

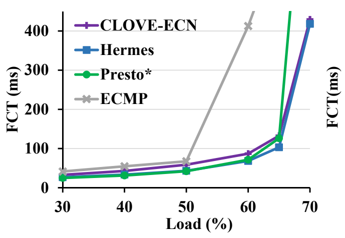
Our servers have 4-core Intel E5-1410 2.8GHz CPU and a Broadcom BCM5719 NetXtreme Gigabit Ethernet NIC. All the servers run Linux kernel 3.18.11. The switches are Pronto 3295 48-port Gigabit Ethernet switch. Since the base RTT of our testbed is around 100μs, we set standard ECN marking threshold to 30KB accordingly. We use a simple client-server application to generate traffic according to the two workloads discussed above and measure the FCT on the application layer. We compare Hermes with ECMP, CLOVE-ECN, and Presto*, all of which are implementable on our testbed.

The symmetric case: Figure 9 shows the results for both the web-search and data-mining workloads in the symmetric case. We see similar trends in both workloads. First, we find that Hermes performs increasingly better (10-38%) compared to ECMP as the load increases. This is as expected because ECMP suffers more from hash collisions at higher loads. Moreover, Hermes performs 9-15% better at 30-70% loads compared to CLOVE-ECN. This is because Hermes has better visibility and can react to congestion more timely than flowlet-based solutions, especially under low and medium loads. Also, we observe that Hermes performs closely to Presto*, which is shown to achieve near-optimal performance in symmetric topologies [20].

Another observation is that Hermes outperforms Presto* by ~13% at 90% load with the web-search workload. One possible reason is that our testbed is not perfectly symmetric because links may not have exactly the same capacity. As a result, Presto* may always be constrained by the bottlenecked link similar to the case shown in Example 2 of §2.2.2. Such congestion mismatch caused by imperfect symmetry may affect Presto* under very high load. In comparison, Presto* is more stable in the data-mining workload. We believe that this is due to the more bursty nature of the web-search workload, which has a smaller inter-flow arrival time and a larger number of small flows.

The asymmetric case: We repeat the above experiments for the asymmetric topology. Note that we only consider loads up to 70% relative to the symmetric case, because the bisection bandwidth is only 75% of the symmetric case.

First, we observe that the performance of ECMP deteriorates after the load exceeds 40-50%. This is because the link between


(a) Web-search

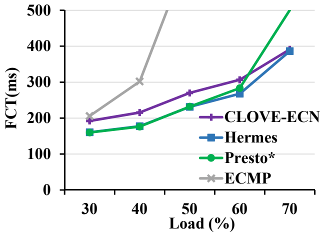
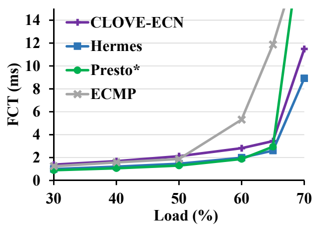

(b) Data-mining

Figure 10: [Testbed] Overall avg FCT in the asymmetric case.


(a) Small flow (<100KB) avg

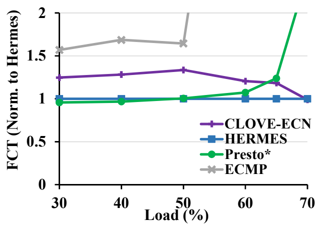

(b) Large flow (>10MB) avg

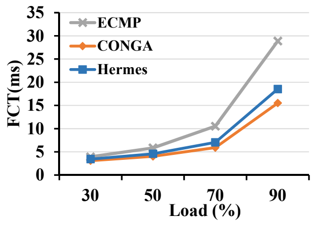
Figure 11: [Testbed] Web-search workload statistics in the asymmetric case. Note that for large flows we normalize the FCT to Hermes to better visualize the results.

spine 1 and leaf 1 becomes fully loaded. Moreover, we observe that Hermes performs 12-30% better than CLOVE-ECN at 30-65% loads — with both workloads and among different flow size groups (Figure 11). Similar to the symmetric case, this demonstrates Hermes's better visibility and more timely reaction to congestion compared with CLOVE-ECN. Note that CLOVE-ECN achieves comparable performance to Hermes at 70% load. One possible reason is that more flowlets are created at high loads, thus CLOVE-ECN can more timely converge to a balanced load.

For Presto*, we take the path asymmetry into account and assign weights for parallel paths statically to equalize the average load [14]. However, we find that even with topology-dependent weights, Presto* fails to match Hermes's performance. We believe that the root cause is congestion mismatch. As the load increases, different paths begin to experience different levels of congestion because the traffic matrix is not symmetric. As a result, the congestion window of Presto* is always constrained by the most congested path, and ECN signals are also mismatched among different paths. Such mismatch greatly affects the normal transport behavior (as shown in the Examples 2 and 3 of §2.2.2) and causes a dramatic increase in FCT after the load exceeds 60%.

5.3 Deep Dive with Large Simulations

Due to testbed limitations, our experiments contain 4 switches, 1 Gbps links, and 1 link failure. Using simulations, we further evaluate Hermes with a larger topology, different degrees of asymmetry, and multiple types of switch failures. Specially, we look into the performance of different schemes in detail, and many of our observations echo our design rationale of timely triggering, cautious rerouting, and enhanced visibility.


(a) Web-search

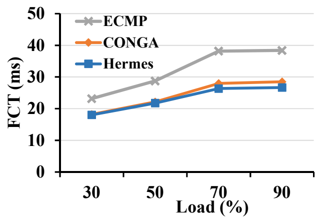

(b) Data-mining

Figure 12: [Simulation] Overall avg FCT (baseline topology).

5.3.1 *Baseline.*

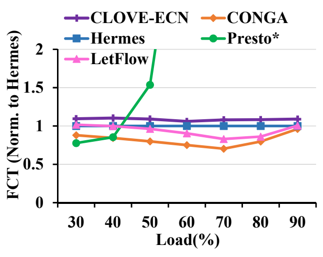
We first inspect the performance of Hermes under a symmetric 8×8 leaf-spine topology with 128 hosts connected by 10Gbps links. We simulate a 2:1 oversubscription at the leaf level, typical of today's datacenter deployments [5]. Figure 12 shows the overall average FCT for the web-search and data-mining workloads. For the web-search workload, Hermes outperforms ECMP by up to 55% as the traffic load increases. As a readily-deployable solution, we observe that Hermes is always within 17% of CONGA (requiring switch modifications) at all levels of loads. For the data-mining workload, Hermes is 29% better than ECMP at high load. Moreover, unlike the web-search workload, we find that Hermes can slightly outperform CONGA, by up to 4%.

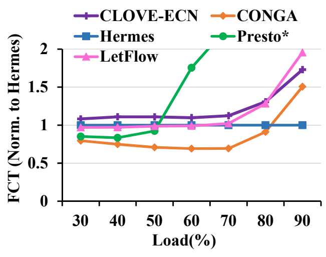
Analysis: The distinction between CONGA and Hermes under the data-mining and web-search workloads suggests the importance of visibility and timely reaction. On the one hand, CONGA outperforms Hermes for the web-search workload. One key reason is that CONGA achieves better visibility by keeping track of all the flows on a leaf switch, as shown in Table 2. On the other hand, Hermes performs slightly better for the data-mining workload because of its timely reaction to congestion. To explain this, note that the data-mining workload contains more large flows; hence, it is more challenging to deal with because multiple large flows may collide on one path [5]. Furthermore, the data-mining workload has a much bigger inter-flow arrival time. So when there are no bursty arrivals of new flows, the packet inter-arrival time of these large flows can be quite stable. Therefore, CONGA cannot always timely resolve such bottlenecks because there are not enough flowlet gaps, as shown in the Example 1 of §2.2.2. In comparison, Hermes can timely reroute these flows once congestion is sensed and a vacant path is found.

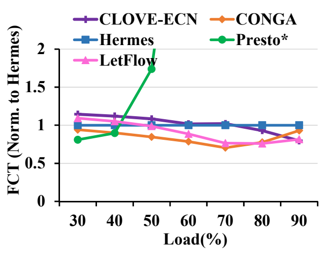
5.3.2 *Impact of Asymmetric Topology.*

We further compare Hermes with CONGA, LetFlow, Clove-ECN and Presto* under an asymmetric topology. We adopt the baseline topology, and reduce the capacity from 10Gbps to 2Gbps for 20% of randomly selected leaf-to-spine links. Note that we normalize the FCT to Hermes in order to better visualize the results.

Under the web-search workload: As shown in Figure 13, CONGA performs over 10% better than the other schemes in most cases. Hermes, CLOVE-ECN and LetFlow achieve similar performance. This is because the web-search workload contains many small flows and is also more bursty. Dynamics such as frequent flow arrival are likely to create flowlet gaps to break large flows into small sized flowlets. With enough flowlets, CLOVE-ECN and LetFlow can


(a) Overall avg FCT


(b) Small flow(<100KB) avg


(c) Large flow (>10MB) avg

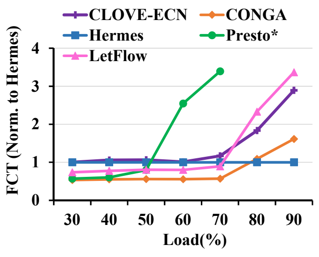
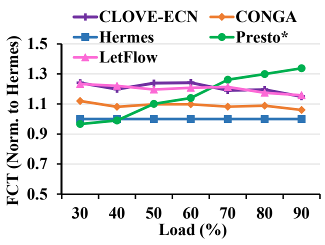
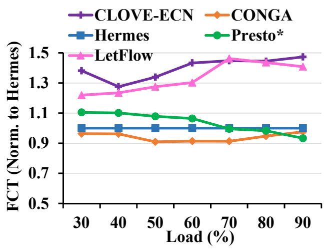
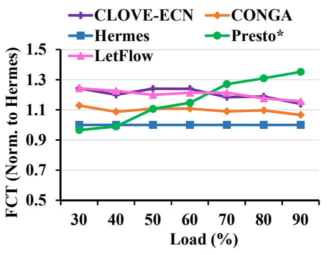

(d) Small flow 99th percentile

Figure 13: [Simulation] FCT statistics for the web-search workload in the asymmetric topology (normalized to Hermes).


(a) Overall avg FCT


(b) Small flow (<100KB) avg


(c) Large flow (>10MB) avg

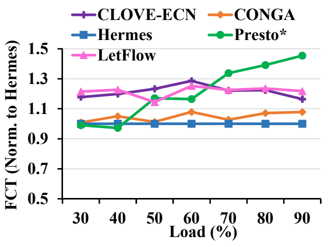

(d) Overall 99th percentile

Figure 14: [Simulation] FCT statistics for the data-mining workload in the asymmetric topology (normalized to Hermes).

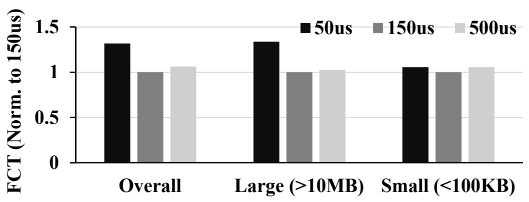


Figure 15: [Simulation] CONGA with different flowlet timeout values (web-search). To evaluate the impact of congestion mismatch, we try to mask the impact of packet reordering by implementing a reordering buffer [15].

quickly converge to a balanced load even without good visibility. In comparison, recall that Hermes outperforms CLOVE-ECN by 9-15% with the web-search workload in testbed experiments. One possible reason is that our 1Gbps testbed has a higher RTT, thus CLOVE-ECN converges more slowly.

However, excessive rerouting opportunities may negatively affect small flows without cautious rerouting. As we can see in Figure 13b and 13d, the average and the 99th percentile FCTs for small flows grow dramatically for flowlet-based solutions as the load increases. This is because small flows are broken into several flowlets under high loads; thus, they are heavily affected by packet reordering and congestion mismatch. In comparison, Hermes outperforms the competition by 1.5-3.3× at 90% load due to its cautious rerouting.

Under the data-mining workload: Figure 14 shows that Hermes outperforms CONGA by 5-10%. Compared to the baseline (Figure 12b), we observe that the timely reaction of Hermes brings more obvious performance gains. This is perhaps because Hermes can effectively resolve collisions of large flows on the 2Gbps links. We also observe that Hermes is 13-20% better than CLOVE-ECN and LetFlow. This is because the data-mining workload is significantly less bursty. When there are not enough rerouting opportunities (i.e., flowlets for CLOVE-ECN and LetFlow), solutions without good visibility can hardly balance the traffic.

Validating congestion mismatch: Similar to our testbed experiments, we observe that Presto* with topology-dependent weights fails to achieve comparable performance to others under the asymmetric topology. To further validate the effect of congestion mismatch caused by vigorous rerouting, we fix the traffic load at 80% and run CONGA with different flowlet timeout gaps. We mask packet reordering to rule out their impact. As shown in Figure 15, we find that reducing the flowlet timeout from 500μs to 150μs improves FCT (by ~6%) due to more rerouting opportunities. However, further reducing the timeout value to 50μs degrades FCT by ~30%. This indicates that even congestion-aware solutions suffer from congestion mismatch. With such a small flowlet gap, CONGA changes path vigorously. This, in turn, negatively affects the normal behavior of transport protocols as we showed in the Example 4 of §2.2.2.

5.3.3 *Impact of Switch Failures.*

We next evaluate Hermes under failure scenarios. We adopt the baseline topology and randomly select one core switch to simulate silent random packet drop and packet blackhole [19]. Note that because only 7 out of 8 core switches are working appropriately, we consider traffic loads up to 70%. We compare Hermes against CONGA, Presto*, LetFlow, and ECMP.

Silent random packet drop: To simulate the silent random packet drop scenario, we set the drop rate to 2% on a randomly selected core switch. Figure 16 shows the performance of different schemes.

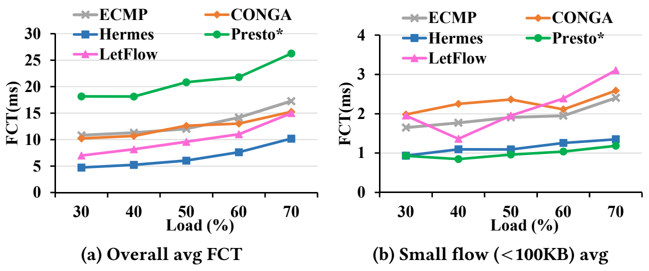


Figure 16: [Simulation] Performance with random packet drops (web-search).

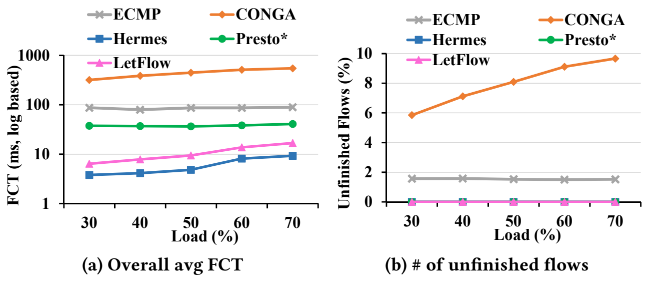


Figure 17: [Simulation] Performance with packet blackhole (web-search).

First of all, we observe that Hermes outperforms all other schemes by over 32%; this is because it can effectively sense the failure and avoid routing through the failed switch. ECMP has 1.7-2.3× higher FCT compared to Hermes even at low loads; this is because about 1/8th of the flows traverse the failed switch using ECMP and are affected by the high packet drop rate. We also observe that CONGA performs similarly to ECMP. To explain this, note that flows traversing the failed switch tend to have a low sending rate due to frequent packet drops. Therefore, CONGA paradoxically shifts more traffic to such undesirable paths because it senses and balances traffic based on network utilization. Presto* is more heavily affected because all the flows have to go through this failed switch and thus be affected.[5] LetFlow is comparatively less affected because random packet drops create more rerouting opportunities on the affected paths. However, without the ability to explicitly detect and avoid failures, LetFlow still performs ∼1.5× worse than Hermes.

Packet blackhole: To simulate the packet blackhole scenario, we drop packets for half of the source-destination IP pairs from Rack 1 to Rack 8 deterministically on one randomly selected switch. Figure 17 shows the performance of different schemes.

As expected, Hermes can effectively detect the blackhole after 3 timeouts; hence, all the flows can finish, and Hermes achieves over 1.6× better FCT than others. For ECMP, a fixed group of flows from Rack 1 to Rack 8 will be hashed to the failed switch, which leads to a ∼1.5% of unfinished flows (Figure 17b). The unfinished flows greatly enlarge the average FCT, which makes ECMP 9-22× worse than Hermes. Similar to the random drop scenario, CONGA paradoxically shifts more flows to the failed switch because it appears to be less congested. This results in a higher portion of unfinished flows and higher FCT compared to ECMP. Presto* works in a round-robin manner so all the flows can finish. However, the average FCT is still greatly enlarged because all the corresponding flows are affected. Finally, we see that LetFlow performs the second best because it can timely reroute the affected flows. However, it is still over 1.6× worse than Hermes because it cannot explicitly detect and avoid failures.

[5]Note that we observe good average FCT for small flows in case of Presto*. One possible reason is that Presto* has a lower network utilization on all the paths, as all large flows are heavily affected and cannot send at high rates.

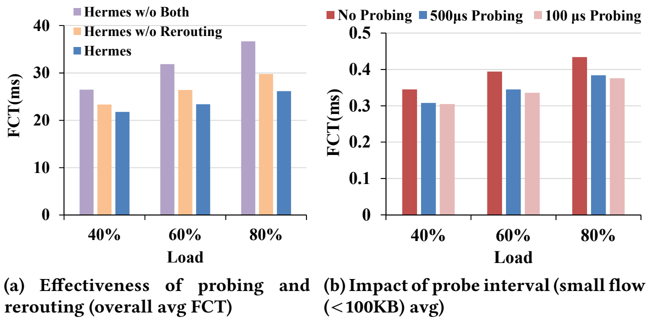


Figure 18: [Simulation] Hermes deep dive (data-mining). Here "without both" refers to Hermes without both probing and rerouting.

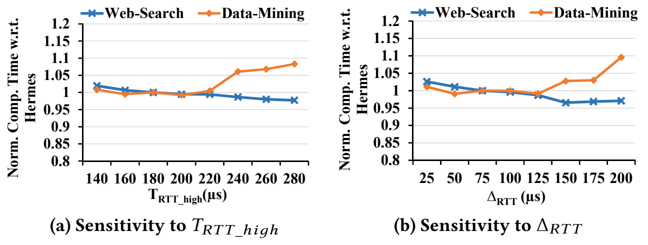


Figure 19: [Simulation] Sensitivity to T_{RTT_high} and Δ_{RTT}.

5.4 Performance Breakdown and Robustness

Effectiveness of probing and rerouting: We have already shown that good visibility and timely yet cautious rerouting together bring good performance. Now we further investigate the incremental benefits of some key design components, e.g., probing and rerouting, using the data-mining workload. Figure 18a shows that probing and rerouting bring around 20% and 10% improvement to the overall average FCT respectively. A similar trend is observed for the large and small flows as well. This observation validates that active probing can effectively increase the visibility of end hosts, and timely rerouting can effectively resolve hotspots caused by collisions among large flows.

Impact of probe intervals: Figure 18b shows that compared to Hermes without probing, a 500μs probe interval brings around 11-15% improvement, while reducing the probe interval to 100μs brings around 1-3% improvement.

Sensitivity of parameter settings: We next study how different parameter settings affect the performance of Hermes. Figure 19a and 19b show the sensitivity analysis for Δ_{RTT} and T_{RTT_high}. First, we observe that the FCT is relatively stable when these two parameters are set around the suggested values. Another observation is that the web-search and data-mining workloads experience different trends as T_{RTT_high} and Δ_{RTT} increase. To understand this, recall that the web-search workload is more bursty; so it tends to create frequent rerouting opportunities. Therefore, a conservative parameter setting (i.e., high T_{RTT_high} and Δ_{RTT}) brings better performance because it can prune excessive reroutings. However, because the data-mining is less bursty, an aggressive parameter setting leads to better performance. We have also tested other parameters and observed that Hermes performs well and relatively stable around the settings suggested in §3.3.

Different transport protocols: We finally check the performance of Hermes with TCP under the 8×8 topology in simulation. For Hermes, we rely only on RTT to sense congestion. Since TCP is more bursty and has a larger RTT, we set Δ_{RTT} and T_{RTT_high} 1.5× larger and keep all the other parameters unchanged. For CONGA, we set the flowlet timout to be 500μs. Under the web-search workload, Hermes is within 10-25% of CONGA at all loads in both baseline and asymmetric topology. Under the data-mining workload, Hermes performs almost identically to CONGA in most cases (figures omitted due to space limitation). We have observed a similar trend for DCTCP in §5.3, except that CONGA performs slightly better relative to Hermes. This is because TCP is more bursty, thus more likely to create sufficient flowlet gaps.

6 DISCUSSION

Hermes is not a panacea. Here, we discuss some design tradeoffs and potential deployment concerns.

End host based sensing: The choice of pushing congestion awareness to the network edge makes Hermes readily deployable and resilient to uncertainties at the same time. However, we note that visibility at end hosts, although enhanced by active probing, is still limited in comparison to switch-based solutions [5, 25, 35]. Better visibility often leads to better initial routing assignments, which can be especially important for small flows. For example, our evaluation shows that CONGA outperforms Hermes in the web-search workload, which has a relatively smaller average flow size.

Effectiveness of the rerouting design: Compared to flowlet-based solutions, the choice of timely yet cautious rerouting is the key for faster reaction to congestion, especially when traffic is too steady to create enough flowlet gaps. As a result, Hermes outperforms flowlet-based solutions under the relatively stable data-mining workload (Figure 12b and 14). However, when flows are small and bursty, dynamics such as frequent flow arrival are likely to create enough flowlet gaps, making flowlet-based solutions equally efficient (Figure 13).

Number of parameters: Hermes introduces a number of parameters to effectively sense path conditions and to make deliberate load balancing decisions. As a result, deploying Hermes requires more tuning compared to solutions with much simpler load balancing logics. At this point, we provide some rules of thumb for parameter tuning. A full exploration of the optimal parameter settings together with an automatic parameter tuning procedure would greatly simplify the deployment of Hermes. We consider it as an important future work.

Burst avoidance and stability: Hermes takes at least one RTT to sense and react to uncertainties, and thus, it does not directly handle microbursts [16]. As for stability, recent congestion-aware load balancing solutions [5, 24, 25] have demonstrated that stable performance can be achieved in practice as long as network state is collected at fine-grained timescales. Compared to these solutions, Hermes can more effectively prevent path oscillations and bursts caused by synchronized routing choices because: 1) Hermes leverages the power of two choices to avoid the herd behavior; and 2) Hermes does not reroute small flows and flows with a high sending rate.

7 RELATED WORK

The literature on datacenter load balancing is vast. Among them, we only discuss some representative ones close to this work.

Presto [20], DRB [12] and RPS [13] are per-packet/flowcell based, congestion-oblivious load balancing schemes. We have shown that they suffer from congestion mismatch under asymmetry.

CONGA [5] and Expeditus [35] employ congestion aware switching on specialized switch chipsets to load balance the network. HULA [25] and CLOVE-INT [24] leverage advanced programmable switches [2, 11] to achieve better visibility.

LetFlow [14] leverages flowlet switching to automatically converge to a balanced traffic share among parallel paths. However, we have shown that flowlets cannot always timely react to congestion under stable traffic patterns, which makes LetFlow converge slowly and achieve suboptimal performance.

CLOVE-ECN [24] adopts per-flowlet weighted round-robin at end hosts, where path weights are calculated based on piggybacked ECN signals. Due to its limited visibility and slow reaction to congestion, CLOVE-ECN performs up to 25% worse than Hermes under an asymmetric topology.

MPTCP [31] is a transport protocol that routes several subflows concurrently over multiple paths. Because subflows do not change paths, MPTCP does not suffer from the congestion mismatch problem. However, MPTCP has some important drawbacks [5, 23, 24]. First, it modifies the end host networking stacks, which makes it challenging to deploy in multi-tenant environments. Moreover, it performs poorly in incast scenarios because several connections are maintained simultaneously for each flow.

DRILL [16] is a switch-local, per-packet load balancing solution with a major goal of resolving micro bursts under heavy load. DRILL does not consider asymmetric topologies in their design, and it reroutes every packet vigorously with only local information. As a result, it also suffers from congestion mismatch under asymmetry.

FlowBender [23] reroutes flows blindly whenever congestion is detected by end hosts. Such random and vigorous rerouting brings sub-optimal performance under high loads.

Finally, all the aforementioned schemes lack the ability to timely react to switch malfunctions, which results in significant performance degradation as we showed in §5.3.3.

8 CONCLUSION

This paper has introduced Hermes, a datacenter load balancer resilient to uncertainties such as traffic dynamics, topology asymmetry, and failures. Hermes leverages comprehensive sensing to detect uncertainties (including switch failures unattended before), and reacts by timely yet cautious rerouting. We have implemented Hermes with commodity switches and evaluated it through testbed experiments and large simulations. We demonstrated that Hermes handles uncertainties well: under asymmetries, Hermes achieves up to 10% (20%) better FCT than CONGA (CLOVE); under switch failures, it outperforms all other schemes by over 32%.

ACKNOWLEDGMENTS

This work is supported in part by the Hong Kong RGC ECS-26200014, GRF-16203715, GRF-613113, CRF-C703615G, and the China 973 Program No.2014CB340303. Mosharaf Chowdhury was supported in part by NSF grants CNS-1617773, CCF-1629397, and CNS-1563095. We are thankful to Zhouwang Fu for his help in testbed evaluation. We would also like to thank our shepherd, Dina Papagiannaki, and the anonymous SIGCOMM reviewers for their valuable feedback.

REFERENCES

[1] DCTCP in Linux Kernel 3.18. "http://kernelnewbies.org/Linux3.18".
[2] In-band Network Telemetry (INT). "http://p4.org/wp-content/uploads/fixed/INT/INT-current-spec.pdf".
[3] Mohammad Al-Fares, Alexander Loukissas, and Amin Vahdat. A Scalable, Commodity Data Center Network Architecture. In *SIGCOMM 2008*.
[4] Mohammad Al-Fares, Sivasankar Radhakrishnan, Barath Raghavan, Nelson Huang, and Amin Vahdat. Hedera: Dynamic Flow Scheduling for Data Center Networks. In *NSDI 2010*.
[5] Mohammad Alizadeh, Tom Edsall, Sarang Dharmapurikar, Ramanan Vaidyanathan, Kevin Chu, Andy Fingerhut, Francis Matus, Rong Pan, Navindra Yadav, George Varghese, and others. CONGA: Distributed Congestion-Aware Load balancing for Datacenters. In *SIGCOMM 2014*.
[6] Mohammad Alizadeh, Albert Greenberg, David A Maltz, Jitendra Padhye, Parveen Patel, Balaji Prabhakar, Sudipta Sengupta, and Murari Sridharan. Data Center TCP (DCTCP). In *SIGCOMM 2010*.
[7] Wei Bai, Li Chen, Kai Chen, Dongsu Han, Chen Tian, and Hao Wang. Information-Agnostic Flow Scheduling for Commodity Data Centers. In *NSDI 2015*.
[8] Wei Bai, Li Chen, Kai Chen, and Haitao Wu. Enabling ECN in Multi-Service Multi-Queue Data Centers. In *NSDI 2016*.
[9] Hitesh Ballani, Paolo Costa, Christos Gkantsidis, Matthew P Grosvenor, Thomas Karagiannis, Lazaros Koromilas, and Greg O'Shea. Enabling End-host Network Functions. In *SIGCOMM 2015*.
[10] Peter Bodík, Ishai Menache, Mosharaf Chowdhury, Pradeepkumar Mani, David A Maltz, and Ion Stoica. Surviving Failures in Bandwidth-Constrained Datacenters. In *SIGCOMM 2012*.
[11] Pat Bosshart, Dan Daly, Glen Gibb, Martin Izzard, Nick McKeown, Jennifer Rexford, Cole Schlesinger, Dan Talayco, Amin Vahdat, George Varghese, and others. 2014. P4: Programming Protocol-Independent Packet Processors. *SIGCOMM CCR* 44, 3 (2014), 87–95.
[12] 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 *CoNEXT 2013*.
[13] Advait Dixit, Pawan Prakash, Y Charlie Hu, and Ramana Rao Kompella. On the Impact of Packet Spraying in Data Center Networks. In *INFOCOM 2013*.
[14] Vanini Erico, Pan Rong, Alizadeh Mohammad, Taheri Parvin, and Edsall Tom. Let it FLow: Resilient Asymmetric Load Balancing with Flowlet Switching. In *NSDI 2017*.
[15] Yilong Geng, Vimalkumar Jeyakumar, Abdul Kabbani, and Mohammad Alizadeh. JUGGLER: A Practical Reordering Resilient Network Stack for Datacenters. In *EuroSys 2016*.
[16] Soudeh Ghorbani, Brighten Godfrey, Yashar Ganjali, and Amin Firoozshahian. Micro Load Balancing in Data Centers with DRILL. In *HotNets 2015*.
[17] Phillipa Gill, Navendu Jain, and Nachiappan Nagappan. Understanding Network Failures in Data Centers: Measurement, Analysis, and Implications. In *SIGCOMM 2011*.
[18] 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 *ACM SIGCOMM 2009*.
[19] 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 *SIGCOMM 2015*.
[20] Keqiang He, Eric Rozner, Kanak Agarwal, Wes Felter, John Carter, and Aditya Akella. Presto: Edge-based Load Balancing for Fast Datacenter Networks. In *SIGCOMM 2015*.
[21] CE Hopps. Analysis of an Equal-Cost Multi-Path Algorithm. In *RFC 2992*.
[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 *NSDI 2015*.
[23] Abdul Kabbani, Balajee Vamanan, Jahangir Hasan, and Fabien Duchene. FlowBender: Flow-level Adaptive Routing for Improved Latency and Throughput in Datacenter Networks. In *CoNEXT 2014*.
[24] Naga Katta, Mukesh Hira, Aditi Ghag, Changhoon Kim, Isaac Keslassy, and Jennifer Rexford. CLOVE: How I Learned to Stop Worrying about the Core and Love the Edge. In *HotNets 2016*.
[25] Naga Katta, Mukesh Hira, Changhoon Kim, Anirudh Sivaraman, and Jennifer Rexford. HULA: Scalable Load Balancing Using Programmable Data Planes. In *SOSR 2016*.
[26] Radhika Mittal, Nandita Dukkipati, Emily Blem, Hassan Wassel, Monia Ghobadi, Amin Vahdat, Yaogong Wang, David Wetherall, David Zats, and others. TIMELY: RTT-based Congestion Control for the Datacenter. In *SIGCOMM 2015*.
[27] Michael Mitzenmacher. 2000. How Useful Is Old Information? *IEEE TPDS* 11, 1 (2000), 6–20.
[28] Michael Mitzenmacher. 2001. The Power of Two Choices in Randomized Load Balancing. *IEEE TPDS* 12, 10 (2001), 1094–1104.
[29] Michael Mitzenmacher, Balaji Prabhakar, and Devavrat Shah. Load Balancing with Memory. In *FOCS 2002*.
[30] Jayakrishnan Nair, Adam Wierman, and Bert Zwart. The Fundamentals of Heavy-tails: Properties, Emergence, and Identification. In *SIGMETRICS 2013*.
[31] Costin Raiciu, Sebastien Barre, Christopher Pluntke, Adam Greenhalgh, Damon Wischik, and Mark Handley. Improving Datacenter Performance and Robustness with Multipath TCP. In *SIGCOMM 2011*.
[32] Arjun Roy, Hongyi Zeng, Jasmeet Bagga, and Alex C Snoeren. Passive Realtime Datacenter Fault Detection and Localization. In *NSDI 2017*.
[33] Shan Sinha, Srikanth Kandula, and Dina Katabi. Harnessing TCP's Burstiness with Flowlet Switching. In *HotNets 2004*.
[34] Balajee Vamanan, Jahangir Hasan, and TN Vijaykumar. Deadline-Aware Datacenter TCP (D2TCP). In *SIGCOMM 2012*.
[35] Peng Wang, Hong Xu, Zhixiong Niu, Dongsu Han, and Yongqiang Xiong. Expeditus: Congestion-aware Load Balancing in Clos Data Center Networks. In *SoCC 2016*.