Efficient Datacenter Load Balancing with Microslices

Hafiza Ramzah Rehman*
Salesforce
hrehman@salesforce.com

Sana Mahmood*
Johns Hopkins University
smahmo12@jhu.edu

Syed Mohammad Irteza
NUCES, Lahore
m.irteza@nu.edu.pk

Fahad Rafique Dogar
Tufts University
fahad@cs.tufts.edu

Ihsan Ayyub Qazi
LUMS
ihsan.qazi@lums.edu.pk

***Abstract*—Efficient load balancing of traffic across multiple paths is crucial for maximizing bandwidth utilization in datacenters. While simple techniques like packet spraying achieve near-optimal performance in symmetric networks, we still lack efficient solutions for scenarios when there are asymmetries in the network (e.g., link failures and topology asymmetry). In this paper, we propose Wino, a new network load balancing scheme for datacenters that is explicitly designed with asymmetry in mind. Wino uses the abstraction of a microslice to construct efficient virtual symmetric topologies. By combining SDN-driven rate limiting with per-packet forwarding, Wino uniformly sprays traffic across these microslices, dynamically adapting to link capacities and failures. Our evaluation shows that Wino achieves robust performance in a range of traffic scenarios and application workloads.**

***Index Terms*—Load Balancing, Datacenters, Software-Defined Networking, Virtual Topologies, Rate limiting**

I. INTRODUCTION

Production datacenters (DCs) provide multiple paths between servers to deliver high throughput (e.g., full bisection bandwidth [1]) and reliability at low cost [2]. To effectively use these multiple paths, various load balancing solutions have been proposed in recent times [3]–[7]. These schemes differ in terms of the load-balancing granularity (packet vs. flowlet vs. flows) as well as the different kind of traffic scenarios they target (e.g., failures vs. no failures).

Per-packet load balancing schemes, such as packet spraying (PS) [8], offer a number of benefits: they are simple to implement, work well with existing transports (e.g., TCP), and can achieve near-optimal performance over symmetric networks. Unfortunately, they perform poorly under *asymmetry* due to increased packet reordering [8], [9]. We argue that network asymmetry in datacenters is the norm rather than the exception. There is mounting empirical evidence that DCs commonly operate under network asymmetry due to link failures [10], [11], switch malfunctions [6], [12], traffic volatility [4], and imbalanced striping [13]. The key to effectively using multiple paths in a DC is a fine-grained load balancing scheme that is robust to asymmetry and works well with TCP.

However, existing load balancing schemes that deal with network asymmetry either interact poorly with TCP or are too coarse-grained to efficiently utilize the available bandwidth under asymmetric networks. For example, weighted spraying approaches, such as Presto [3], improve performance over PS under asymmetric scenarios by sending less traffic over more congested paths but can interact adversely with transport protocols (TCP) in the presence of bandwidth asymmetry (e.g., due to partial failures—whereby a link's speed drops due to switch malfunctions [6], grey link failures [9], [14] or Ethernet's auto-negotiation [9], [15]). This occurs because besides packet reordering, flows face the *congestion mismatch problem*, whereby the growth of a flow's congestion window is limited by the bandwidth-delay product of the slowest link, leading to loss of throughput [6].

Given that asymmetries are a norm and application workloads can vary over time, we call for a load balancing solution that effectively deals with such scenarios. To this end, we ask, "*how can we make PS robust to network asymmetry and workload distributions while retaining its simplicity (e.g., without requiring any changes in TCP)?*"

To answer this question, we design, implement, and evaluate Wino, a simple network-layer load balancing scheme for DCs. Wino achieves fine-grained load balancing by using the abstraction of a *microslice*, which refers to a virtual topology of a given bandwidth realized by SDN rate limiting. It uses a logically centralized control plane to detect asymmetries and

*Joint first authors. This research was carried out during their appointments at LUMS.

ISBN 978-3-903176-72-0

adaptively construct microslices by using rate limiters in switches. Finally, packets are sprayed over microslices through suitable marking at the end-hosts.

To avoid the congestion mismatch problem and limit packet reordering, all microslices in Wino have the *same* bandwidth because doing so provides symmetric bandwidth and delay to flows, which enables per-packet load balancing to achieve high performance. To achieve these tasks, Wino actively makes use of various features provided by SDN switches: (i) detecting link failures (including partial failures) and notifying these failures to the controller, (ii) multiple forwarding tables that can be used to enforce symmetric virtual topologies through appropriate rate limiting, and (iii) mapping packets to an appropriate microslice based on marking in the packet header (e.g., DSCP value).

We implemented Wino using OpenFlow 1.3 [16] and deployed it on a small-scale 10 Gbps testbed. In addition, we also evaluate Wino using ns-3 simulations [17] and compare its performance with multiple load balancing schemes over realistic DC workloads (e.g., web search [18] and data mining [19]) and traffic scenarios. Our evaluation shows that Wino outperforms SAPS [5], Hermes [6] and CONGA [4], both in terms of average flow completion times (AFCT) of short flows as well as throughput of long flows.

We make the following contributions.

–We propose the **microslice** abstraction that allows PS to be robust to asymmetry and workload distributions.

–We designed **Wino**, a load balancing scheme that uses SDN-based rate limiting to adaptively construct microslices and then employs per-packet load balancing across microslices.

–We implemented **Wino** on a real testbed and the ns-3 simulator and compare it with other schemes.

–We evaluated **Wino** over a range of traffic scenarios and workloads and show that it performs well. For example, it improves the AFCT by up to 95.4%, 53.3%, and 50.1% over SAPS, Hermes, and CONGA respectively under the web search workload.

The paper is organized as follows. Section II motivates the need for DC load balancing schemes to be robust to asymmetry and failures. In Section III, we present the Wino design. Section IV presents real testbed & simulation results. Section V presents related work. In Section VI, we discuss Wino's limitations.

II. Motivation

In this section, we first motivate the need for asymmetry and failure-resilient schemes for load balancing (LB) in DCs and then quantify the performance of some existing LB schemes under failure scenarios.

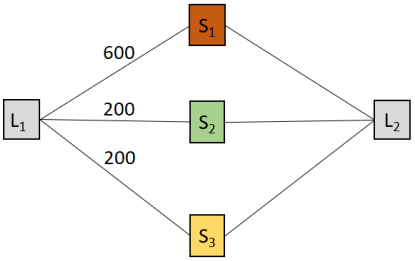


Fig. 1: 2×3 Clos topology with 1 Gbps links.

A. Types of Failures and Network Asymmetry

DCs are known to experience frequent link and switch failures [6] (e.g., large DCs see 40 link failures per day on average [10]). Moreover, there are several factors that affect the *symmetry* of DC network topologies, including:

Link failures: Partial and complete link failures routinely occur in DCs [20]. For instance, damaged wires or improperly seated RJ45 conductors in Cat-7 cables (10 Gbps Ethernet) can cause links to fall back to lower rates (e.g., 1 Gbps, 100 Mbps) [15].

North-South (N-S) traffic induced asymmetry: N-S traffic (i.e., traffic to/from endhosts outside the DC) can create bandwidth asymmetry for intra-DC traffic (also known as *East-West* or E-W traffic) when both types of traffic share links.

Imbalanced striping: Extra links can be added between leaf and spine switches by aggregating the links to enhance throughput beyond a single connection. This creates network asymmetry [13].

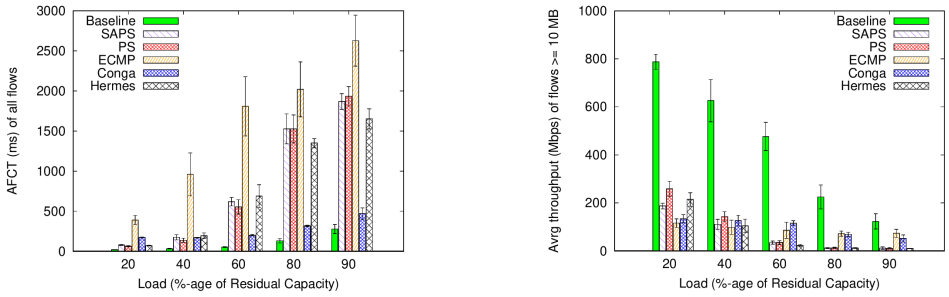
Silent packet drops and blackholes: Switches can silently drop packets due to CRC errors or TCAM deficits [6]. Packet blackholes occur when specific source/destination pairs experience drops due to TCAM entry corruption [21].

B. Limitations of Existing Approaches

State-of-the-art LB schemes such as Hermes [6], CONGA [4] and SAPS [5] aim to address the limitations of packet spraying schemes (e.g., PS) and ECMP in asymmetric scenarios. With the help of ns-3 simulations,[1] we now analyze the performance of these schemes in an asymmetric scenario and show that from the baseline (ideal) scheme (described later), they perform worse, leaving significant room for performance improvement. Other details of our evaluation setup are available in Section IV.

We consider a 2×3 Clos topology (see Figure 1) and compare different schemes over the web search workload [18]. To create bandwidth asymmetry, we introduce persistent N-S traffic from L_1 towards all

[1]We use the publicly available ns-3 implementations of Hermes and CONGA provided by the authors of Hermes [6].



(a) AFCT (b) Throughput of elephant flows ($\geq$ 10MB)

Fig. 2: SAPS, PS, ECMP, CONGA, and Hermes degrade in performance under network asymmetry, with increased AFCT and lower throughput

spine switches (i.e., S_1, S_2, and S_3), which leads to residual bandwidths of 600 Mbps, 200 Mbps and 200 Mbps for E-W traffic from L_1 to S_1, S_2, and S_3 respectively, as shown in Figure 1.

For this setup, we construct a baseline (ideal) scheme where each link is divided into virtual links of capacity 200 Mbps and then we use packet spraying to load balance traffic among those virtual links. This ensures that no flow faces asymmetry and that each flow can utilize all paths between L_1 and L_2.

Figure 2a shows the AFCT of all flows, achieved by different schemes as a function of network load. Observe that ECMP performs worst across all loads because it (a) uses random mapping of flows onto paths, leading to frequent collisions of large flows that degrade throughput and (b) always uses a *single path* for every flow, which may not be the path with the largest available bandwidth in cases of asymmetry. PS and SAPS perform better than ECMP but still result in high AFCTs due to asymmetry. Specifically, PS faces packet reordering and congestion mismatch, SAPS's size-based mapping limits the throughput achieved by long flows resulting in large AFCTs.

Hermes performs better than the above schemes by proactively detecting path conditions (e.g., ECN marking and RTT measurements at the transport layer). However, it performs worse than CONGA because it uses indirect congestion signals, which can lead to false positives–resulting in frequently switching flows–and thereby hurting the performance of large flows in particular, as shown in Figure reffig:simulation-motivation-Avg-Thrpt-elephant. CONGA performs better than other schemes because it explicitly measures the load on paths and performs flowlet switching on short time scales. However, it results in 1.7–8.1$\times$ larger AFCTs than the baseline for two reasons: (i) it also faces the congestion mismatch problem [6] and (ii) with CONGA, flows cannot utilize the bandwidth available on *all paths*, which limits the throughput of long flows (see Figure 2b).

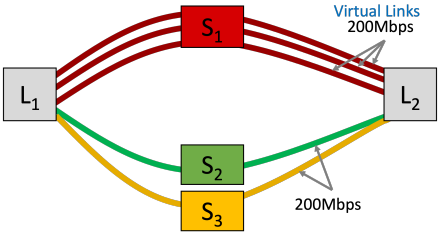


Fig. 3: Microslices created by Wino for the topology shown in Fig. 1, across which it sprays packets.

Summary. In asymmetric scenarios, state-of-the-art load balancing schemes (e.g., CONGA and Hermes) significantly improve performance over PS. However, they face the congestion mismatch problem and do not allow flows to efficiently aggregate bandwidth by using multiple paths. Thus, there is significant room for improvement, which is exploited in Wino.

III. DESIGN

We now discuss the design goals of Wino, the principles it applies to achieve these goals, and its key mechanisms to construct microslices.

A. Design Goals and Principles

Wino has the following design goals:

Efficient network utilization: It should achieve efficient utilization of the available bandwidth across various traffic scenarios and workloads (with the given traffic demand).

Ease of deployment: It should be easy to deploy without requiring custom hardware changes in network switches or applications (which may be proprietary).

Resilience to failures and asymmetry: It should be robust to different types of failures and network asymmetry.

To achieve the above goals, Wino relies on three key design principles: First, to achieve efficient network utilization, a scheme should perform fine-grained LB

(e.g., packet level) to limit load imbalances across paths, as evidenced by earlier works [5], [7], [8].

Second, to achieve ease of deployment, a scheme should rely on features available in commodity SDN switches, and any sender-side changes within the DC should be implemented outside the applications (e.g., at the network layer).

Third, to achieve resilience to failures, a scheme should be capable of detecting different types of failures and then adapt the network configuration to achieve high performance. For instance, to achieve efficient utilization with per-packet LB and limit packet reordering, flows should be provided with a symmetric network view comprising paths with symmetric bandwidth and delay characteristics.

Wino Overview. Wino applies these design principles through a logically centralized SDN controller that has a global view of network asymmetries. The Wino controller installs flow table rules and rate limiters on each switch to create microslices—which are symmetric virtual topologies of equal bandwidth—over which packets are sprayed. It employs *practical optimizations* when constructing microslices, ensuring that the overhead of rate limiting is taken into account. Finally, *end-host marking* ensures that packets receive appppropriate rate limiting treatment at switches. With homogeneous microslices, Wino eliminates the need for size-based flow mapping used in SAPS [5]).

We now describe Wino's three key design elements: (i) a controller module implementing the core functionality for creating, monitoring, and updating *microslices*, (ii) end-host marking that enables per-packet load balancing across slices, and (iii) optimizations accounting for rate limiter overheads in switches.

B. Wino Controller

The logically centralized Wino controller maintains a complete view of the network topology and is responsible for the following tasks: (i) monitoring of link failures (including partial as well as full link failures) by listening on asynchronous port status update messages[2] from OpenFlow switches, similar to [5] and other asymmetries (e.g., due to N-S traffic) by periodically measuring link utilizations [22], [23], and (ii) computing the bandwidth (or size) of microslices based on newly detected asymmetries and, (iii) setting up forwarding rules to realize microslices (via OpenFlow rate meters) and enable packet spraying based on packet header fields.

In case of asymmetry (e.g., link failure), the Wino controller detects the event and then computes the new microslice size using Algorithm 1. The algorithm takes as input the residual bandwidths of links on all paths affected by asymmetry and then computes their greatest common divisor (GCD), which is the microslice bandwidth.[3] The bandwidth is then enforced using OpenFlow rate meters. On leaf switches, packets are mapped to a microslice by matching against a packet header field.[4] On spine switches, packets are forwarded based on the destination leaf and the microslice on the downstream link.

Algorithm 1: Microslice size selection

Input: A set $C = \{c_{11}, c_{12}, \ldots, c_{ij}\}$ where c_{ij} represents the residual bandwidth of link i on a leaf-to-leaf path j affected by asymmetry.
Output: Size of microslice selected by Wino.
s_{lim}: minimum supported slice size due to switch constraints (e.g., rate limiter accuracy, TCAM size)
S_s: slice size selected for the network
WB: wasted bandwidth (if any)
$S_s \leftarrow max(GCD(c_{1j}, c_{2j}, \ldots, c_{rj}), s_{lim})$
$WB \leftarrow 0$
for *each link i on a leaf-to-leaf path j* **do**
  $WB \mathrel{+}= S_s \mod c_{ij}$
end
return S_s

We illustrate the working of Wino with the example shown in Figure 1. Given the residual bandwidths, the microslice bandwidth will be 200 Mbps (i.e., the GCD of 600, 200, 200). Thus, Wino will create five slices each of 200 Mbps bandwidth; one slice each on L_1-S_2-L_2 and L_1-S_3-L_2 paths and three slices on L_1-S_1-L_2 path. The slices constructed in this case are shown in Figure 3. To realize these slices, Wino will install five flow table rules at the L_1 switch to forward traffic destined towards L_2 and for each rule it adds a rate meter with a limit of 200 Mbps. Wino then sprays packets across microslices in a round robin fashion based on packet marking at the endhosts.[5]

C. End-host Packet Marking

To enable packet spraying across slices in a round-robin fashion; flows are marked at the endhosts to uniquely identify a slice. Marking occurs in the hypervisor; or in the shim layer below the transport layer, similar to PIAS [24], CLOVE [25], and SAPS [5]. For our experiments, we use the DSCP field for marking;

[2]These messages include the current downgraded speed of the port on which the link is currently operating as a result of the failure.

[3]If there is a minimum microslice size defined by the operator (e.g., due to rate limiter accuracy or to minimize TCAM usage), there may be some loss of bandwidth, which is calculated in Algorithm 1 using the *WB* variable.

[4]In our implementation, we use the DSCP header field.

[5]To install rate meters and flow table rules, the controller sends meter & flow modification messages to the switch.

and the mapping between the DSCP value and the slice is statically determined. To allow more microslices, we can also use VXLAN [26].

D. Practical Optimizations

The rate limiters in network switches can have varying levels of accuracy in meeting the target rate based on the amount of instantaneous bursts allowed [27], [28]. In general, allowance for larger bursts enables more efficient network usage but may violate rate limits by a greater margin, which can create delay asymmetry across microslices thereby leading to packet reordering. We now evaluate the accuracy of OpenFlow rate meters to inform microslice sizes.

Meter granularity and accuracy. We evaluate the accuracy of OpenFlow rate limiters in enforcing the target rate on our testbed comprising HPE ARUBA 2930F switches as well as ns-3 simulations (we used the `OFSwitch13` module available in ns-3). We observed similar behaviour in both settings so we omit results obtained from simulations for brevity.

We consider a two-hop network comprising a single OpenFlow switch, a sender and a receiver machine. We use OpenFlow rate limiting on the downstream link between the switch and the receiver. The round-trip time (RTT) of the network was measured to be ∼70 µs. The sender generates UDP traffic of fixed-size Ethernet frames of 1522 bytes to the receiver for 30 s. We use UDP instead of TCP to avoid the effect of TCP congestion control on meter accuracy measurements. To determine the granularity as well as time-window at which the meter rates can be effectively enforced, we vary the meter rate from 10 Kbps to 7 Gbps[6] and measure the throughput achieved by UDP traffic over time-windows ranging from 1 ms to 300 ms. The sending rate is set to 10 Gbps. To measure the accuracy of meter enforcement, we compute the meter error E over each time-window as follows:

$$E = \begin{cases} \dfrac{T - R}{R} \times 100 & T > R \\ 0 & T \leq R \end{cases}$$

where, R is the target meter rate and T is the throughput achieved. We then take the average of the meter error, $\bar{E}$, across all samples. Thus, $\bar{E}$ is positive if the average throughput exceeds the target rate across samples. The results are shown in Figure 4. The error bars show 95% confidence intervals.

[6]We tested the meters at up to 7 Gbps of rate instead of 10 Gbps because the OpenFlow pipeline of the hardware switch in our testbed couldn't support rates above 7 Gbps. In particular, it discarded packets at the ingress port beyond this rate. Note that several existing 40 Gbps and 100 Gbps OpenFlow switches can achieve line-rate performance so the 7 Gbps constraint we observe is likely due to the hardware implementation of our switch model only [29]–[31].

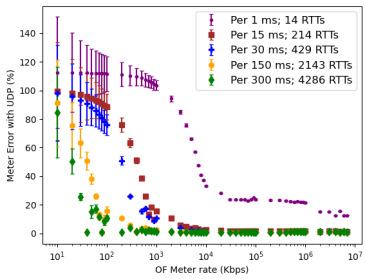


Fig. 4: OpenFlow meter accuracy in the HW switch. The OF meter is tested with UDP traffic over various sending rates (10 Kbps - 7 Gbps) and time-windows (1 ms - 300 ms). Error bars show 95% conf. intervals.

Challenges at target low rates. As shown in Figure 4, the error is large when the target rate is low. For example, when the target rate is less than 1 Mbps, $\bar{E}$ ranges from 18% to nearly 140% at 15 ms intervals. This occurs because the effective bandwidth-delay product in these regimes is even smaller than a single 1522 byte packet. Thus, sending a single packet exceeds the target rate thereby inducing an error.

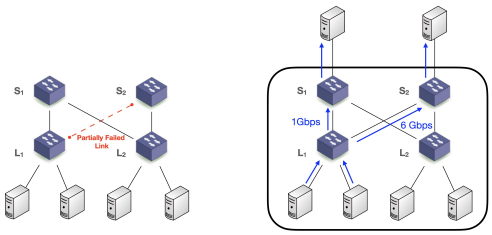
Challenges with bursty traffic. At high target rates, we still observe some error at short timescales (e.g., 1 ms). This occurs because rate limiters in switches typically allow the output rate to exceed the target rate within a short interval for efficiency reasons (e.g., by using the burst size parameter [32]). This allowance leads to a positive error.

The above results provide insight into the design decisions for Wino. In particular, by observing the meter errors, we determine: (i) the minimum size of a microslice in Wino without inducing high error; and (ii) the maximum number of microslices that can be constructed. We find that if we want to enforce rates of a microslice over 15 ms (or 214 RTTs) time-windows with a guarantee of less than 2% average error, a minimum slice size of 7 Mbps can be constructed.

IV. Evaluation

We evaluate Wino using experiments conducted on a small-scale 10 Gbps testbed as well as through ns-3 simulations [17]. The simulations allow us to compare Wino against state-of-the-art load balancing schemes while the testbed evaluation provides a proof-of-concept validation as well as practical insights related to realizing Wino using OpenFlow switches.

We compare Wino's performance with CONGA, Hermes, SAPS and PS. For our simulation evaluation, we use the publicly available ns-3 implementations of CONGA and Hermes [6], [33], and choose the default parameters for the schemes. We implemented SAPS and PS in ns-3 and found the results to be within 2% of



(a) Failure Case: link between L_1 and S_2 switches is partially failed

(b) N-S Traffic Case: 1 Gbps demand from L_1 to S_1, and 6 Gbps. & from L_1 to S_2

Fig. 5: 2×2 leaf-spine topology with 2 hosts per leaf.

the published results [5], [8]. For testbed experiments, we compare Wino with SAPS.[7]

Workloads. We consider two well-known DC traffic workloads: web search [18] and data mining [19]. The web search workload is based on ∼150 TB of data, which was collected from three clusters comprising nearly 150 racks and 6000 servers and includes real-time query traffic, short messages that coordinate activities, and background traffic [18]. The data mining workload [19] contains data from a heavily used cluster of 1500 nodes collected over 2 months. Both workloads exhibit heavy-tailed characteristics, but the data mining workload is more heavy-tailed with ∼98% of all the bytes contained in ∼5% of the flows greater than 2 MB, and 80% of the flows are less than 10 KB.

Communication Pattern. We use the *left-to-right* communication pattern, where all hosts in the left subtree generate traffic towards the hosts in the right subtree and the flows are generated in a round-robin manner. This is a common scenario in user-facing web services where the front-end servers and the back-end storage reside in separate racks [24], [34], [35].

Failures/Asymmetries. We evaluate Wino in case of partial failures as well as a scenario with N-S traffic [5], [36]. We consider two partial failure scenarios in which we reduce the bandwidth of the L_1-S_2 link to 1 Gbps and 5 Gbps, similar to earlier works [5], [6], [9], as shown in Figure 5a. Such partial failures are possible when a NIC supports Multi-Gigabit Ethernet like 1, 2.5, 5, and 10GbE [15]. For N-S traffic (Figure 5b), we consider traffic going from L_1 to a remote endhost through S_1 with a demand of 1 Gbps and from L_1 via S_2 with a demand of 6 Gbps, similar to [3], [5]. This traffic leaves 9 Gbps of residual bandwidth along path L_1 to S_1 and 4 Gbps of residual bandwidth along the path L_1 to S_2 for E-W traffic.

Metrics. To evaluate the performance of Wino compared to other schemes, we measure AFCT across all completed flows. In addition, we report the AFCT of short (mice) flows & the average throughput of large (elephant) flows.

[7]We do not perform comparison with CONGA on the testbed as it requires custom switch support whereas a real implementation of Hermes was not available.

A. Summary of Results

Our evaluation centers around four key questions:

How does Wino perform in comparison with other schemes? Our ns-3 evaluation shows that Wino outperforms SAPS, Hermes, and CONGA. On a 2×2 leaf-spine topology with 10 Gbps links, we find that Wino improves the AFCT by up to 44.3%, 53.3%, and 50.1% over SAPS, Hermes, and CONGA under the web search workload when there is one partially failed link with reduced capacity of 1 Gbps. On a larger 8×8 Clos topology, where 20% of the links between leaf and spine switches are downgraded from 10 Gbps to 2 Gbps. we find that Wino provides up to 11% improvement over CONGA.

How does Wino perform over different DC workloads? Wino's performs well across workloads. We find that Wino improves the AFCT by up to 65.5% over SAPS under data mining workload across loads.

How does Wino perform in the presence of asymmetry? We find that Wino offers AFCT improvements for different levels of asymmetry. For example, it improves AFCT by up to 95.4% over SAPS when the partially failed link has a degraded capacity of 5 Gbps.

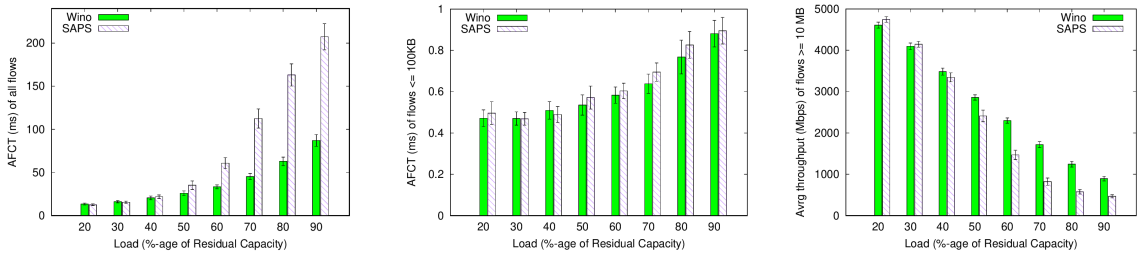
How does Wino perform on a real testbed? Wino offers significant improvement in the AFCTs over SAPS for 40% or higher loads under both traffic workloads and under partial failure and N-S scenarios.

B. Testbed Experiments

We first evaluate the performance of Wino in comparison with SAPS on a 10 Gbps testbed. We use SAPS [5] because it has been shown to outperform ECMP and PS. We use a 2×2 Clos topology comprising two hosts per rack as shown in Figure 5. The average round trip delay between two hosts in different racks was ∼80 µs. We use Aruba 2930F switches and Dell PowerEdge servers in our experiments. We implement the Wino controller as a Ryu SDN controller and use OpenFlow 1.3. We implement network slicing using OpenFlow rate meters. We use TCP CUBIC, which is the default transport protocol in Linux with an initial window of 10 packets similar to [24]. We use a simple client-server program to generate traffic. Each server requests flows according to a Poisson process from randomly chosen servers. The flow size is drawn from one of the workload distributions discussed earlier.

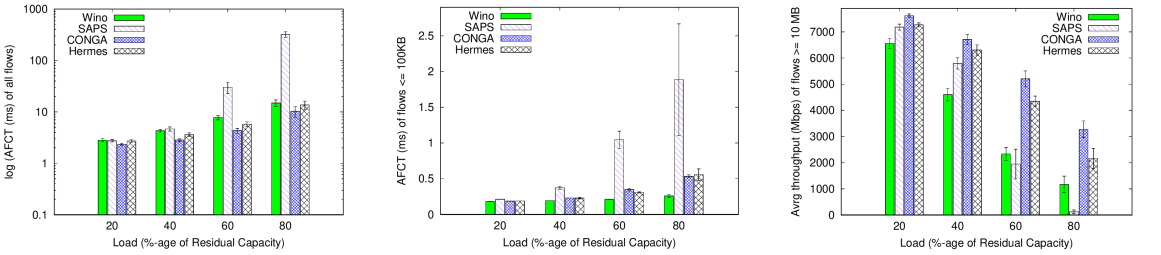
Data Mining Workload. We first evaluate Wino under the data mining workload.

Under Partial Link Failure. We first consider a scenario whereby we downgrade the capacity of a link



(a) AFCT (b) AFCT-Mice (≤100KB) (c) Avg. throughput of Elephant (≥10MB)

Fig. 6: (a) AFCT, (b) AFCT of mice flows & (c) avg. throughput of large flows with Wino and SAPS, with a partially failed link of 5 Gbps capacity, under the data mining workload. Error bars show 95% conf. intervals.



(a) AFCT (b) AFCT-Mice (≤100KB) (c) Avg. throughput of Elephant (≥10MB)

Fig. 7: AFCTs & avg. throughput for Wino, SAPS, CONGA, & Hermes with varying network load under partial failure (10G-5G), web search workload, & the left-to-right pattern. Error bars show 95% conf. intervals.

to 5 Gbps (thereby emulating a partially failed link) as shown in Figure 5a. We vary the network load from 20% to 90% according to the post-failure network capacities. Our results show that Wino achieves comparable AFCTs at low loads (i.e., under 40% load), and offers up to 61.5% improvement over SAPS at higher loads as shown in Figure 6a. In this case, SAPS creates two virtual topologies, one across the 10 Gbps path and the other across the 5 Gbps path; it sprays long flows on the former path. It sprays short flows to either the 10 Gbps path or the 5 Gbps path, based on a probability proportional to their bandwidths. On the other hand, Wino creates 15 microslices each of capacity 1 Gbps, thereby exposing 15 Gbps of bandwidth to the flows. At lower loads, Wino's performance is similar to SAPS because the network is not saturated and cannot benefit as much from the greater bandwidth offered by Wino. At higher loads, since SAPS maps all long and the majority of short and medium flows onto the 10 Gbps path, the 10 Gbps path gets saturated; whereas the 5 Gbps path is relatively less utilized. Wino reduces AFCTs significantly because it effectively uses the bandwidth available across both paths via equal distribution of traffic across microslices.

We then perform micro-benchmarking and analyze small and large flows separately. We find small improvements in the AFCTs of mice flows as shown in Figure 6b, i.e., 1.6% to 8.1% across most loads. This occurs because the data mining workload has mostly mice flows of sizes ≤ 10 KB, and SAPS forwards the majority of them to the 10 Gbps path and thus, these flows typically finish within one RTT. On the other hand, we find that as load increases, the average throughput gains of long flows also increase. In particular, we observe 16% to 54% improvement at network loads higher than 40% with highest gain occurring under 80% load (see Figure 6c).

Under North-South Traffic Scenario. Next, we emulate a N-S traffic scenario shown in Figure 5b by restricting the bandwidth of L_1-S_1 and L_1-S_2 links to 9 Gbps and 4 Gbps, respectively using OpenFlow rate meters. We find that Wino & SAPS achieve comparable AFCTs till 40% load but for higher loads, Wino improves the AFCTs by 16% to 65.6%. This occurs because SAPS maps long flows to the 9 Gbps path; whereas Wino equally distributes load across a virtual topology of 13 Gbps.

Web Search Workload. Considering the same partial failure scenario as before, we find a similar trend but significantly lower AFCTs (ranging from 6.1% to 84% improvement) over SAPS at 40% and higher loads. This happens because the web search workload is less skewed than the data mining workload and thus has a larger fraction of medium/elephant flows; these flows benefit from Wino's design. Since mice and elephant flows share paths, it also improves the AFCT of short flows. We also see gains in the average throughput of long flows. Similarly, we observe a substantially larger improvement in AFCT in the N-S scenario (compared to the data mining workload).

Parameter	Value
Packet size	1500 bytes
Initial window size	10 packets
Queue Size	1000 packets
Retransmission Timeout	5 ms
SAPS mice flow threshold (web search)	100 KB
SAPS mice flow threshold (data mining)	1000 KB

TABLE I: Simulation settings for ns-3 experiments.

C. Simulation Results

Next, we use ns-3 simulations to (i) expand our experiments to include other state-of-the-art load balancing schemes (i.e., CONGA and Hermes) and (ii) conduct evaluation on larger topologies with varying degrees of asymmetry. For this evaluation, we used the `OFSwitch13` OpenFlow module in ns-3 to emulate real OpenFlow-enabled switches.

1) Evaluation over a 2×2 Clos Topology: First, we use the same 2-tier Clos topology with two hosts per rack and 10 Gbps links as in the testbed setup. The round trip latency between two hosts in different racks is 80 μs. We use TCP CUBIC as the transport. The asymmetry is created using the OpenFlow rate meters. The experiment settings are mentioned in Table I.

Under Partial Link Failure (10G-5G). We now evaluate Wino under the the partial link failure scenario while using the web search workload. We find that Wino improves the AFCT by up to 95.4% against SAPS (see Figure 7a). As the web search workload contains many small to medium sized flows, the traffic tends to be bursty. In a saturated network, Wino is better at handling the bursts of traffic because it provides full available capacity to all of its flows whereas SAPS limits most of its flows to 10 Gbps capacity. The same trend is observed in the AFCT of short flows (of size smaller or equal to 100 KB) where the improvement over SAPS ranges from 15.8% to 82.9%; see Figure 7b. Compared to CONGA and Hermes, Wino achieves comparable AFCTs. We omit data mining results due to space limitations, which were found to be similar to the testbed results for Wino and SAPS.

V. Related Work

Previous work on DC load balancing primarily targets optimal utilization of available bisection bandwidth across multiple paths. Recent schemes focus on robustness to various asymmetries, whether due to network failures [10], or N-S flows [5]. Schemes like CONGA [4] are congestion-aware, while others, like ECMP and Presto [3], are not.

LB schemes can also be classified on the granularity of their LB decisions. Schemes that make *flow level* decisions are considered as the most coarse granularity, for example ECMP [37], the *de facto* LB scheme within commodity switches, as well as WCMP [13]. LB schemes that make *packet level* decisions are the most fine-grained granularity LB schemes; such as RPS [8], DRILL [7], SAPS [5], CAPS [38] and QDAPS [39].

Presto [3] introduced the concept of *flowcell* spraying, where a fixed size 64 KB chunk of data (i.e., the *flowcell*) is the level of granularity for LB decisions. While *flowcell* spraying reduces packet reordering, Presto's lack of congestion awareness leads to degraded performance under network asymmetry. Luopan [40] also uses *flowcell* granularity; however it periodically samples two randomly chosen paths & routes the *flowcell* to the least congested path.

Another level of granularity is the *flowlet*, a variable-sized burst of packets for which one specific path is chosen. Flare [41], CONGA [4], CLOVE-ECN [25], HULA [42], W-ECMP [43], and Halflife [44] all employ different versions of *flowlet switching*. Depending on the burstiness of traffic and the *flowlet* timer settings, *flowlets* can be as small as a single packet, and sometimes longer than a single flow. AG [45] adjusts the switching granularity based on the degree of asymmetry in the network.

QDAPS [39] is a packet-level LB scheme that aims to minimize packet reordering; it chooses the path for each packet based on the queuing delay faced in the output buffer. CAPS [38] sprays short flow packets across all paths, but limits long flows to a few paths via ECMP. Schemes such as FlowBender [46] and Hermes [6], while built on ECMP, enable flow re-routing based on signals like ECN and RTT. ConWeave [47] is a fine-grained LB scheme for RDMA traffic, that enables an in-network reordering mechanism for out-of-order packets.

VI. Limitations and Discussion

We present some limitations of Wino & discuss possible future work.

Optimization using demand estimation. Wino's current design does not account for asymmetries that may be *indirectly* induced (on other paths) due to paths that have been directly affected by asymmetry.

Beyond 2-tier topologies. Our evaluation focuses on 2-tier topologies that are common in DCs. Microslices can be extended to 3-tier Clos topologies, albeit with added complexity at the controller.

Evaluation under dynamic scenarios. Our evaluation concentrated on relatively static asymmetric scenarios. We plan to evaluate Wino with failures occurring at *short timescales* in future work.

VII. Conclusion

In this work, we presented Wino, a load balancing scheme for datacenters that makes per-packet load balancing schemes efficient by using the abstraction of microslices. Microslices are virtual topologies of

equal bandwidth that are realized through rate limiters available in SDN switches. Through experiments conducted over a 10 Gbps testbed and ns-3 simulations, we show that Wino outperforms state-of-the-art schemes including SAPS, CONGA and Hermes under the web search and data mining workloads across different network asymmetry scenarios.

REFERENCES

[1] M. Al-Fares, A. Loukissas, and A. Vahdat, "A scalable, commodity data center network architecture," in *SIGCOMM'08*.

[2] V. Liu, D. Halperin, A. Krishnamurthy, and T. Anderson, "F10: A fault-tolerant engineered network," in *NSDI'13*.

[3] K. He, E. Rozner, K. Agarwal, W. Felter, J. Carter, and A. Akella, "Presto: Edge-based load balancing for fast datacenter networks," in *SIGCOMM'15*.

[4] M. Alizadeh, T. Edsall, S. Dharmapurikar, R. Vaidyanathan, K. Chu, A. Fingerhut, V. T. Lam, F. Matus, R. Pan, N. Yadav, and G. Varghese, "Conga: Distributed congestion-aware load balancing for datacenters," in *SIGCOMM'14*.

[5] S. M. Irteza, H. M. Bashir, T. Anwar, I. A. Qazi, and F. R. Dogar, "Load balancing over symmetric virtual topologies," in *INFOCOM'17*.

[6] H. Zhang, J. Zhang, W. Bai, K. Chen, and M. Chowdhury, "Resilient datacenter load balancing in the wild," in *SIGCOMM'17*.

[7] S. Ghorbani, Z. Yang, P. B. Godfrey, Y. Ganjali, and A. Firoozshahian, "Drill: Micro load balancing for low-latency data center networks," in *SIGCOMM'17*.

[8] A. Dixit, P. Prakash, Y. C. Hu, and R. R. Kompella, "On the impact of packet spraying in data center networks," in *INFOCOM'13*.

[9] D. Wischik, C. Raiciu, A. Greenhalgh, and M. Handley, "Design, implementation and evaluation of congestion control for multipath tcp," in *NSDI'11*.

[10] N. Shelly, B. Tschaen, K.-T. Förster, M. Chang, T. Benson, and L. Vanbever, "Destroying networks for fun (and profit)," in *HotNets'15*.

[11] P. G. Kannan, N. Budhdev, R. Joshi, and M. C. Chan, "Debugging transient faults in data centers using synchronized network-wide packet histories," in *NSDI'21*.

[12] C. Tan, Z. Jin, C. Guo, T. Zhang, H. Wu, K. Deng, D. Bi, and D. Xiang, "NetBouncer: Active device and link failure localization in data center networks," in *NSDI'19*.

[13] J. Zhou, M. Tewari, M. Zhu, A. Kabbani, L. Poutievski, A. Singh, and A. Vahdat, "WCMP: Weighted cost multipathing for improved fairness in data centers," in *EuroSys'14*, 2014.

[14] A. Roy, H. Zeng, J. Bagga, and A. C. Snoeren, "Passive real-time datacenter fault detection and localization," in *NSDI'17*.

[15] E. Lynskey, "Auto-negotiation for 10GBASE-T," https://bit.ly/3fh1Da4, January 2004.

[16] "Openflow switch specification 1.3.3," http://bit.ly/1wuZZU0.

[17] "ns-3 Network Simulator," https://www.nsnam.org/.

[18] M. Alizadeh, A. Greenberg, D. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta, and M. Sridharan, "Data center TCP (DCTCP)," in *SIGCOMM'10*.

[19] A. Greenberg, J. R. Hamilton, N. Jain, S. Kandula, C. Kim, P. Lahiri, D. A. Maltz, P. Patel, and S. Sengupta, "VL2: a scalable and flexible data center network," in *SIGCOMM'09*.

[20] P. Gill, N. Jain, and N. Nagappan, "Understanding network failures in data centers: measurement, analysis, and implications," in *SIGCOMM'11*.

[21] C. Guo, L. Yuan, D. Xiang, Y. Dang, R. Huang, D. Maltz, Z. Liu, V. Wang, B. Pang, H. Chen, Z.-W. Lin, and V. Kurien, "Pingmesh: A large-scale system for data center network latency measurement and analysis," in *SIGCOMM'15*.

[22] T. Benson, A. An, A. Akella, and M. Zhang, "MicroTE: The case for fine-grained traffic engineering in data centers," in *In ACM CoNEXT'11*.

[23] D.-H. Luong, A. Outtagarts, and A. Hebbar, "Traffic monitoring in software defined networks using opendaylight controller," in *MSPN*. Springer, 2016, pp. 38–48.

[24] W. Bai, L. Chen, K. Chen, D. Han, C. Tian, and H. Wang, "Information-agnostic flow scheduling for commodity data centers," in *NSDI'15*.

[25] N. Katta, A. Ghag, M. Hira, I. Keslassy, A. Bergman, C. Kim, and J. Rexford, "Clove: Congestion-aware load balancing at the virtual edge," in *CoNEXT'17*.

[26] "VXLAN," http://en.wikipedia.org/wiki/Virtual_Extensible_LAN.

[27] P. P. Tang and T.-Y. Tai, "Network traffic characterization using token bucket model," in *IEEE INFOCOM'99*, pp. 51–62 vol.1.

[28] M. Karakus and A. Durresi, "Quality of service (qos) in software defined networking (sdn): A survey," *Journal of Network and Comp. Applications*, vol. 80, pp. 200–218, 2017.

[29] "Mx series universal routing platforms datasheet," https://rb.gy/mfygl.

[30] "Cisco Nexus 3000 series switches," https://rb.gy/5fglg.

[31] "HPE FlexFabric 12900 taa-compliant switch series," https://rb.gy/01dkv.

[32] "Qos: Policing and shaping configuration guide, cisco ios release 15m&t," https://bit.ly/379d66X.

[33] "Hermes," https://github.com/snowzjx/ns3-load-balance.

[34] D. Zats, T. Das, P. Mohan, D. Borthakur, and R. Katz, "DeTail: Reducing the Flow Completion Time Tail in Datacenter Networks," in *SIGCOMM'12*.

[35] A. Munir, G. Baig, S. Irteza, I. A. Qazi, F. Dogar, and A. Liu, "Friends, not foes - synthesizing existing data center transport strategies in pase," in *SIGCOMM'14*.

[36] A. Roy, H. Zeng, J. Bagga, G. Porter, and A. C. Snoeren, "Inside the social network's (datacenter) network," in *SIGCOMM'15*.

[37] C. Hopps, "RFC 2992: Analysis of an equal-cost multi-path algorithm," USA, 2000.

[38] J. Hu, J. Huang, W. Lv, Y. Zhou, J. Wang, and T. He, "CAPS: Coding-based adaptive packet spraying to reduce flow completion time in data center," in *IEEE/ACM Transactions on Networking, Dec 2019*.

[39] J. Huang, W. Lv, W. Li, , J. Wang, and T. He, "QDAPS: Queueing delay aware packet spraying for load balancing in data center," in *ICNP'18*.

[40] P. Wang, G. Trimponias, H. Xu, and Y. Geng, "Luopan: Sampling-based load balancing in data center networks," *IEEE Transactions on Parallel and Distributed Systems*, vol. 30, no. 1, pp. 133–145, 2019.

[41] S. Kandula, D. Katabi, S. Sinha, and A. Berger, "Dynamic load balancing without packet reordering," *SIGCOMM CCR*, vol. 37, no. 2, pp. 51–62, Mar. 2007.

[42] N. Katta, M. Hira, C. Kim, A. Sivaraman, and J. Rexford, "HULA: Scalable load balancing using programmable data planes," in *SOSR'16*.

[43] J.-L. Ye, C. Chen, and Y. Huang Chu, "A weighted ecmp load balancing scheme for data centers using p4 switches," in *CloudNet*, 2018, pp. 1–4.

[44] S. Liu, Y. Gao, Z. Chen, J. Ye, H. Xu, F. Liang, W. Yan, Z. Tian, Q. Sun, Z. Guo *et al.*, "Halflife: An adaptive flowlet-based load balancer with fading timeout in data center networks," in *EuroSys'24*.

[45] J. Liu, J. Huang, W. Li, and J. Wang, "AG: Adaptive switching granularity for load balancing with asymmetric topology in data center network," in *ICNP'19*, pp. 1–11.

[46] A. Kabbani, B. Vamanan, J. Hasan, and F. Duchene, "FlowBender: Flow-level adaptive routing for improved latency and throughput in datacenter networks," in *CoNEXT'14*.

[47] C. H. Song, X. Z. Khooi, R. Joshi, I. Choi, J. Li, and M. C. Chan, "Network load balancing with in-network reordering support for rdma," in *ACM SIGCOMM '23*.