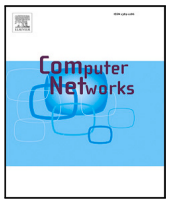
Computer Networks 257 (2025) 110980

Contents lists available at ScienceDirect

Computer Networks

journal homepage: www.elsevier.com/locate/comnet



DeFlow: Differential flowlet switching for load balancing in datacenter networks

Ying Wan [a], Haoyu Song [b], Yu Jia [a], Yi Wang [d], Ling Qian [a,*], Tian Pan [c,*]

[a] China Mobile (Suzhou) Software Technology, China
[b] Futurewei Technologies, USA
[c] Purple Mountain Laboratories, China
[d] Southern University of Science and Technology, China

ARTICLE INFO

Keywords:
Load balancing
Differential flowlet switching
Flow classification
Data center networks

ABSTRACT

Multipath enhances the reliability and bandwidth of datacenter networks, but it also necessitates effective load balancing. The dynamic nature of traffic and diverse flow characteristics present significant challenges in achieving optimal load distribution. Existing schemes either result in mediocre performance or rely on hard-acquired global information, leading to poor FCT or high implementation costs. This paper introduces DeFlow, a simple yet efficient flowlet-based load balancing scheme implementable on programmable switches with a low cost. DeFlow distinguishes flowlets of large and small flows based on packet size and interval. It employs distinct scheduling strategies for throughout-sensitive large flows and latency-sensitive small flows under congestion by prioritizing the performance of small flows. Extensive experiments on NS-3 demonstrate that DeFlow consistently outperforms competing schemes across various topologies and workloads, improving the FCT of small flows and throughput of large flows simultaneously.

1. Introduction

As a fundamental infrastructure of the intelligence era, Data Center Networks (DCNs) facilitate the storage, processing, and transmission of data generated by applications such as AI model training, edge computing [1], online gaming, and multimedia streaming [2]. To provide high bandwidth, reliability, and scalability, DCNs usually employ multi-rooted tree topologies like Fat-Tree [3], Spine-Leaf [4], and VL2 [5], which support tens of terabits per second of bidirectional bandwidth through multipath between end hosts across different racks. For instance, a Fat-Tree network with four pods shown in Fig. 1(a) allows four paths between servers in different pods, while there are two paths between servers connected to different leaf switches for the Spine-Leaf network in Fig. 1(b).

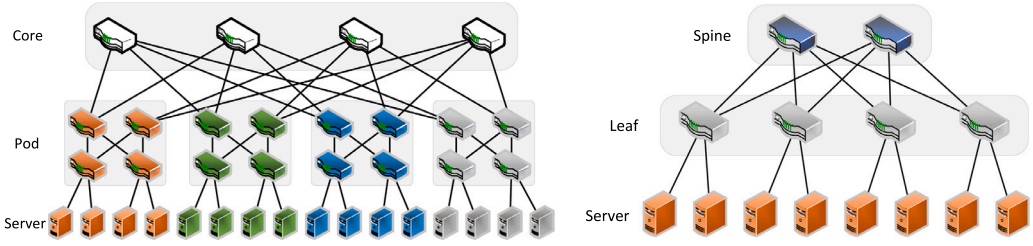
Despite the high aggregate bandwidth offered by these multipath architectures, certain traffic characteristics pose significant challenges in fully utilizing the bandwidth on all paths. (1) **Transiency**: Services such as web search and database query exhibit bursty traffic patterns [6–8], leading to transient congestion on certain paths. This underscores the necessity for real-time congestion detection and swift traffic redirection to suitable paths. (2) **Locality**: Traffic in DCN typically exhibits a skewed distribution where a small number of large flows account for the majority of the traffic, whereas numerous small flows contribute only a minor fraction [8,9]. Therefore, schemes such as ECMP [10], which randomly hashes flows to paths, may achieve balanced distribution in terms of flow quantities but can result in severe load imbalance due to flow size variance. (3) Differentiation: It is recognized that large flows and small flows usually have distinct transmission requirements. Specifically, large flows tend to be throughput-sensitive, prioritizing sustained high bandwidth, while small flows are more latency-sensitive [11,12], requiring low Flow Completion Time (FCT). This diversity necessitates differentiated path allocation and scheduling strategies to accommodate the needs of each type of flow.

Nowadays, poor load imbalance remains the culprit behind network congestion, even in scenarios where overall bandwidth is sufficient. Queuing delays and packet drops due to overflow on congested paths significantly degrade the flow transmission performance [13,14], impacting metrics such as FCT, overall throughput, and packet reordering. Therefore, it is essential to leverage the multipath capabilities of DCNs to distribute traffic effectively, thereby maximizing network bandwidth utilization and boosting application performance.

During the past two decades, numerous works have delved into the load balancing problem in DCN, which can be divided into three categories based on the assumed deployment locations. (1) **Switch**: Each switch distributes traffic to candidate ports based on various

* Corresponding authors (Tian Pan, Ling Qian).
E-mail address: 1003378739@qq.com (Y. Wan).

https://doi.org/10.1016/j.comnet.2024.110980
Received 1 August 2024; Received in revised form 25 October 2024; Accepted 3 December 2024
Available online 9 December 2024



(a) Fat-Tree. (b) Spine-Leaf.

Fig. 1. Two typical DCN architectures.

granularities, varying from packet [15] to flow [16], flowcell [17], and flowlet [18,19]. Finer granularity typically leads to a more balanced path load. Random Packet Spraying (RPS) [15] supported by many commercial switches (e.g., Cisco [20] and Broadcom [21]) is the simplest scheme yet can achieve the near-perfect load balancing and is listed as a key feature by Ultra Ethernet Consortium (UEC) for Ethernet-based AI and HPC workload optimization [22]. However, RPS increases the likelihood of packet reordering. This not only exerts extra processing pressure (e.g., sorting) on the receiver but also misleads the sender into perceiving congestion, leading to unnecessary reduction in transmission rate. Distributing flowlets, bursts of packets from a flow, by a silent time interval, achieves better load distribution than ECMP and reduces the possibility of packet reordering compared to RPS. However, it is congestion-agnostic, limiting its effectiveness in reacting to network congestion; furthermore, the tiny adjacent-packet interval (i.e., arriving time difference) in large flows poses a practical challenge in accurately identifying flowlets. (2) **Controller**: A central controller can designate a path for each flow according to the flow size and path states (e.g., bandwidth utilization) [23–28]. Although theoretically achieving optimal load balance through global state awareness, they often suffer from delays that can extend up to several seconds in production environments [29], preventing timely adjustments during congestion. Moreover, they require specialized protocols to communicate link status and install numerous rules frequently to implement routing decisions. (3) **Endhost**: Since switches determine the forwarding port for each flow through hashing specific packet header fields, endhosts can trigger rerouting at switches by modifying these fields when congestion is detected [30–32]. However, they require modifying the protocol stack and selecting the fields that do not affect forwarding or flow semantics. Additionally, hashing are unknown to the endhosts, there is no guarantee the newly selected path is different or better. Moreover, path status is acquired through explicit congestion notifications [33] or Round Trip Time (RTT) [34] measurements. This approach is less effective to small flows as most small flows conclude within one or a few RTTs [35].

An ideal load balancing scheme should maximize bandwidth utilization and promptly respond to congestion, optimizing the FCT of small flows and enhancing throughput for large flows simultaneously.

Our research identifies critical differences in size and duration between flowlets of large and small flows, which contribute to load imbalance and congestion. Although it is possible to distinguish large flows from small flows using custom header fields, this approach presents scalability and implementation challenges. Fortunately, we can leverage the notable flowlet disparity to accurately distinguish large and small flows at switches, without involving endhosts or controllers. Additionally, we find that redirecting numerous small flows from a congested path has minimal impact on alleviating congestion. Conversely, rerouting just a few large flows is enough to relieve congestion, thereby reducing FCT for the remaining flows and enabling large flows to access suitable paths for sustained high-speed transmission. Although this may introduce slight packet reordering in some cases, the overall enhancement in large flow throughput is substantial.

In light of these insights, we propose DeFlow, a novel load balancing scheme that effectively overcomes the limitations of existing methods. The principal methodology of DeFlow is twofold. (1) It distinguishes between large and small flows and distributes their flowlets to better paths when congestion is absent. (2) It employs differentiated scheduling strategies to handle flowlets of large and small flows during congestion, prioritizing small flows to enhance their FCT.

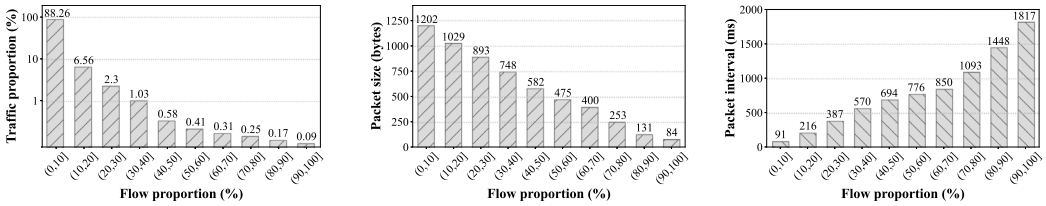
In this paper, we make the following major contributions:

- We demonstrate the notable disparities in average packet size and intervals between large and small flows, and propose a simple and efficient Flowlet Differentiating Algorithm (FDA) to promptly and accurately identify the flowlets of both large and small flows by considering the characteristics of the flow over a recent bout of time with a primary focus on its current characteristics. The computational and storage resources consumed by FDA are minimal, ensuring no impact on line-rate processing capabilities of hardware switches.
- We propose the Flowlet Switching Algorithm (FSA) to apply differential scheduling strategies for flowlets of large and small flows. On the one hand, FSA always chooses the better one between the original path and another randomly selected path to forward each flowlet when no congestion happens. On the other hand, when the original path endures congestion, FSA swiftly redirects the large flows, which not only finds suitable paths for sustained high-speed transmission for the throughput-sensitive large flows, but also alleviates congestion on the original path to keep low FCT and avoid packet reordering for the remaining latency-sensitive small flows.
- We implement a hardware prototype of DeFlow on the Tofino-based programmable switches at a low host and conduct extensive simulations on NS-3 [36] to verify its performance. Compared with RPS, ECMP, and LetFlow [18], DeFlow not only cuts down the FCT of all flows by 1.56×, 1.41×, and 1.35×, but also results in an improvement of 1.29×, 1.56×, and 1.15× for the throughput of large flows, respectively.

The remainder of the paper is organized as follows. Section 2 provides the background. Section 3 discusses the related works. Section 4 details the key algorithms of DeFlow: FDA and FSA. Section 5 presents the performance evaluation and hardware implementation. Finally, Section 6 concludes the paper.

2. Background

In this section, we first illustrate the traffic locality through statistical analysis of the real-world DCN traffic; then we discover the disparities in packet size and interval for large and small flows, which can be used for fast flow size estimation.



(a) Traffic locality. (b) Flow disparity in packet size. (c) Flow disparity in packet interval.

Fig. 2. Traffic characteristics of Equinix datacenters.

2.1. Traffic locality

Traffic locality refers to the phenomenon that, in a short period of time, a small number of large flows occupy the majority of the traffic. Fig. 2(a) shows the traffic statistics of the Equinix datacenter [37] at 13:15 on March 15, 2018, where the value $(a, b]$ on the X-axis corresponds to the 5-tuple flows ranked from a% to b% and the Y-axis represents the percentage of traffic contributed by these flows. It can be seen that, after sorting the 1110,342 flows by size in decreasing order, the top 10% of flows account for 88.26% of the traffic, and the bottom 50% of flows comprise less than 1.5%. Our analysis of the traffic collected from CERNET by IPTAS [38] at 12:56 CST on March 8, 2018 yields similar findings: the top 1.5% of flows contribute to approximately 80% of the traffic. The statistical results on traffic from other datacenters also validate the traffic locality [39–41].

The traffic locality tells us that load balancing based only on flow count without considering the flow size (e.g., ECMP) is set to yield sub-optimal results. When an excessive number of large flows are unfortunately allocated to the same path, the path will be congested while the other paths are underutilized.

2.2. Flowlet switching

On the other extreme, random packet spraying across multiple paths can achieve perfect load balancing. However, this approach can lead to excessive packet reordering. As depicted in Fig. 3(a), two consecutive packets of the same flow, p_1 and p_2, are assigned to two different paths, S0→S2→S3 and S0→S1→S3. Due to the different path delay, p_2 arrives at S3 before p_1. The packet reorder may be interpreted as packet loss and trigger sending rate reduction or even re-transmission, leading to poor network goodput and flow performance. To counter this problem, some designs apply a reorder buffer at the receiver which is expensive to implement [17].

In the above example, if the time interval between p_1 and p_2 is larger than the path delay difference, transmitting them via different paths will not cause packet reordering. As illustrated in Fig. 3(b), a flow is divided into two consecutive but non-overlapping flowlets fl_1 and fl_2. A flowlet is a collection of continuous packets, of which the adjacent packet interval does not exceed the timeout δ but the adjacent flowlet interval is larger than δ. As long as δ is large enough, transmitting flowlets over different paths will not cause packet reordering.

2.3. Flow differentiation

The simplicity and ease of implementation make flowlet a promising load balancing scheme in DCN. However, it grapples with two main challenges. (1) It is difficult to set a suitable δ. If it is too small, it may still incur excessive packet reordering; otherwise, too few flowlets can be detected and the scheme degrades to ECMP. (2) It does not differentiate between large and small flows so it fails to catch their different characteristics and cater for their unique needs.

After an in-depth analysis of real-world DCN traffic, we discover that large flows and small flows show significantly different characteristics in terms of packet size and interval. Figs. 2(b) and 2(c) present the average packet size and interval of all flows after sorting in the decreasing order of their sizes, where packet size and interval are normalized with respect to the corresponding values of the first and last 10% of the flows, respectively. It is evident that large flows have a larger average packet size and smaller average packet interval. The average packet interval of the last 10% of the flows is 19.97× larger than that of the first 10% of the flows, while their packet size is 14.31× smaller.

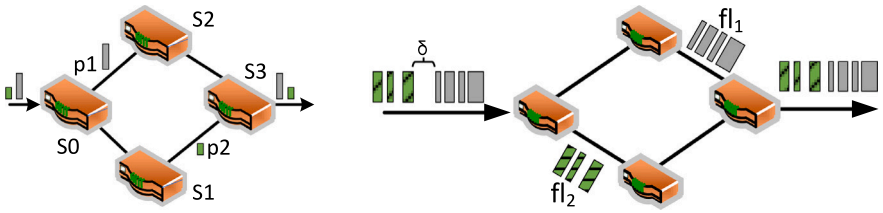
Considering the fact that a flowlet comprises consecutive packets of the same flow, the discrepancy in packet size and interval between large and small flows induces an even more pronounced difference (e.g., $\frac{avg\_interval}{avg\_size}$) at the granularity of flowlet. It is easy to get flowlets for small flows, but it does not help too much on load balancing; it is desirable to get more flowlets for large flows, but it is more difficult. In this paper, we distinguish large and small flows based on their characteristics and subsequently apply different path scheduling strategies to them.

3. Related work

The DCN load balancing schemes can be classified based on the main location they are implemented in, be it switches, controllers, or endhosts.

3.1. Switch based solutions

ECMP is the standard load balancing scheme in DCN due to its simplicity. It randomly assigns a path from multiple candidate paths for each flow. While ECMP can approximately equalize the number of flows assigned to each path, its ignorance of flow size leads to potential congestion. Weighted Cost Multiple Path (WCMP) [42] allows different number of flows to be assigned to each path based on its weight. DFFR [43] assumes that both path capacities and flow sizes are known in advance and directly allocates flows to the path with the largest residual capacity. Due to the significant variations in flow sizes, achieving load balancing between paths is challenging. Flare [44] divides a flow into flowlets and scatters flowlets randomly to multiple paths. It ensures a more balanced link utilization and reduces the probability of packet reordering. LocalFlow [45] constructs special matching rules to spatially split a flow into multiple subflows and assigns a path for each subflow. LetFlow reveals that flowlet can effectively deal with asymmetric DCN topologies. Burstbalancer [46] uses sophisticated sketch-based algorithms to detect the flow bursts and only switches the path for the substantially large flowlets. It sacrifices the performance of small flows which are more sensitive to delay and consumes too much resource in programmable switches. The aforementioned methods are all congestion-agnostic and do not react to congestion. Conga [47] and Hula [48] are deployed on edge switches. They obtain global path



(a) Packet reorder. (b) Flowlet switching.

Fig. 3. Illustration of packet reorder and flowlet switching.

status through probe packets and distribute the flowlet on the least congested link. They need specific communication protocols and unique data structures to exchange and record path information, and need to control the frequency of probe packets to prevent excessive bandwidth consumption. Drill [13] compares the queue length of each egress port on a switch and sends packets to the least congested port without considering the packet reorders. LBT [49] applies different packet spraying strategies based on the predetermined flow type. Specifically, it directs packets of small and large flows to the least and relatively congested ports, respectively, compromising the performance of large flows. FAMG [50] identifies a flow as a large flow if two consecutive packets entering a switch belong to the same flow and randomly distribute its flowlets across all available links. Meanwhile, FAMG sprays packets of small flows randomly on the less congested links, which significantly increases packet reordering. BBN [51] employs a binary neural network model for flow classification, which consumes a relatively high amount of hardware resources. Besides, BBN relies on the host to split flows into flowcells and distributes the flowcells of a flow to its private links, with a higher probability of selecting less congested links. Proteus [52] classifies paths into three priority levels based on signals such as RTT and link utilization, and selects the highest-priority available path for each flowlet at the source switch, while downstream switches rerouting the packets if the cumulative sojourn time is high. Note that Proteus, Conga, and Hula, which determine the complete path for packets at the source switch, have two main drawbacks. First, embedding path information in packet headers leads to extra packet overhead and bandwidth wastage. Second, the explosion of path numbers due to link combinations may exceed the resource limits of the switches, resulting in poor scalability.

3.2. Controller based solutions

Hedera [53] reads the rate of each flow from the underlay switch, then selects a path for each large flow to maximize the global path utilization, and finally issues rules to the switches to guide subsequent forwarding. Mahout [24] detects large flows through the socket buffer size in sender and then informs the controller to allocate these large flows to the least congested path according to the global path information obtained from the switches. In FDALB [54], the sender marks the packets of the large flows, the passing switch relies on the controller to decide how to forward the large flows. Freeway [23] categorizes paths into two types based on their utilization: high throughput and low latency. It always assigns the optimal high-throughput path for large flows reported by the switch, while the switch directly chooses the least-congested path for small flows. Fastpass [55] is more aggressive: the sender notifies the controller of its transmission requirements and the controller then determines the sending time and path for each packet. In DiffFlow [56], edge switches identify large flows based on the duration time and notify the controller, which then issues corresponding rules to guide the forwarding of subsequent packets of this flow. For small flows, DiffFlow employs the ECMP strategy. Although these solutions are good at preventing congestion, they rely on centralized controllers for decision-making [57], which leads to substantial computational and communication overheads, making deployment at scale challenging. Besides, the time required for the control loop may be too long to react to the dynamic DCN conditions.

3.3. Endhost based solutions

MPTCP [58] divides the single flow into multiple subflows at the sender by modifying certain header values. Flowbender [30] relies on the ECN flag in the ACK packet to perceive the path congestion and then modifies the header values of subsequent sent packets. Presto [17] divides the flow into segments of a fixed size (e.g., 64 KB) at the sender, and assigns different header values to each segment. Clove [31] learns the mapping between the packet's header values and the paths by sending multiple rounds of probes before sending normal packets. These solutions assume that the switches would hash on the packet header to select a path for each packet. The protocol modification at senders and the recovery at receivers bring additional computation and storage requirements. Moreover, the header field selection needs to ensure the switch can support it and the change of it will not affect the forwarding and the other network services.

Table 1 offers a concise summary of the above-mentioned schemes, covering their scheduling granularities, evaluation methods, deployment locations, utilized signals, and key limitations.

4. Algorithm description and implementation

Since large flows are the primary cause of load imbalance, and they are not as sensitive to latency as small flows, an effective load balancing scheme should distinguish between these two types of flows and employ differential path-scheduling strategies accordingly. We have shown in Fig. 2 that large flows exhibit distinct differences in packet size and adjacent-packet interval compared to small flows. In light of such insight, FDA in DeFlow can accurately identify the flowlet of large flows and FSA in DeFlow prioritizes the flowlets of small flows over that of large flows, thereby simultaneously accommodating the throughput sensitivity of large flows and the latency sensitivity of small flows.

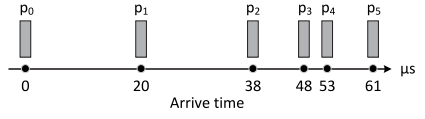
4.1. Flowlet differentiation algorithm

Fig. 2 reveals that packet interval Δt and packet size l can be used as hints to distinguish flowlets from large or small flows. The implementation of flowlet differentiation needs careful consideration. One can simply record the duration (T), the number of packets (C), and the total number of bytes (L) of each flowlet, calculate the average packet interval $g = \frac{T}{C}$, and the average packet size $s = \frac{L}{C}$, and mark flowlets with $g < \alpha$ and $s > \beta$ as flowlets of large flows, where α and β are the predefined thresholds based on the real-world traffic characteristics. However, such a scheme suffers a delay that makes it hard to capture real-time flowlet changes, and it is susceptible to the influence of the packets sent long time ago. As illustrated in Fig. 4(a), the large flow's flowlet fl_1 which contains six packets $p_0 \sim p_5$ never meets the criteria ($g < 12$) to be identified as one for large flow, with the changing average intervals $g_{0\sim5} = \langle \frac{20}{1}, \frac{38}{2}, \frac{48}{3}, \frac{53}{4}, \frac{61}{5} \rangle$. However, the flowlet fl_1 fulfills the criteria if only the last four packets are considered, during which $g_{2\sim5}$ is $\langle \frac{10}{1}, \frac{15}{2}, \frac{23}{3} \rangle$.

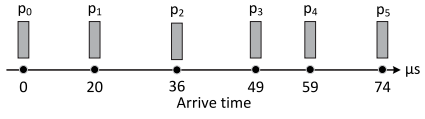
As an alternate approach, one can only consider the currently arriving packet and the last packet of the same flow when calculating

Table 1
Comparison of existing load balancing schemes.

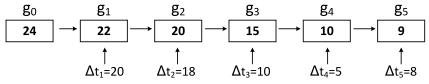
LB scheme	Deployed location	Utilized signals	Flow classification	Scheduing granularity	Evaluation method	Primary limitations
DFFR [43]	Edge switch	Path capacity, Flow size	Packet header	None	Custom simulator	Unbalancd load between paths
Flare [44] LetFlow [18]	Switch	None	None	Flowlet	NS-2 [59]	Congestion-agnostic
LocalFlow [45]	Switch	None	None	Subflow	Custom simulator	Excessive forwarding rules
Burstbalancer [46]	Switch	Flowlet size	None	Large flowlet	NS-2	Congestion-agnostic
Conga [47]	Edge switch	Path's queue length	None	Flowlet	OMNET++ [60], custom ASIC	Custom hardware
Hula [48]	Edge switch	Path's utilization	None	Flowlet	NS-2	Poor Scalability, bandwidth waste
Drill [13]	Switch	Ports' queue length	None	Packet	OMNET++	Increased reordering, custom hardware
LBT [49]	Switch	Ports' queue length	Packet header	Packet	NS-2	Degrade FCTs of large flows
FAMG [50]	Switch	Ports' queue length	Flowlet interval	Packet	NSL-3 [36]	Increased reordering
BBN [51]	Switch	Ports' queue length	Binary neural network model	Flowcell	NS-3	Endhost modification
Proteus [52]	Switch	Path priority, link utilization	None	Flowlet	NS-3	Poor Scalability, bandwidth waste
Hedera [53]	Controller	Flow rate, path utilization	None	Flow	NS-2	Poor Scalability, long control loop
Mahout [24]	Controller	Path's queue length	Socket buffer	Flow	Custom simulator	Endhost modification
FDALB [54]	Controller	Flow size	Packet header	Flow	Unknwon	Endhost modification
Freeway [23]	Controller	Path utilization, port's queue length	Flow size	Flow	htsim [58]	Frequent path partition
Fastpass [55]	Controller	Path utilization, flow size	None	Packet	Custom hardware	Low compatibility
DiffFlow [56]	Controller	None	Flow duration time	Packet	Custom simulator	Congestion-agnostic, long control loop
MPTCP [58]	Endhost	None	None	Subflow	htsim	Increased FCTs of small flows
Flowbender [30]	Endhost	Path's queue length	None	Flow	NS-3	Long control loop
Presto [17]	Endhost	None	None	Flowcell	OpenL vSwitch [61]	Increased FCTs of small flows
Clove [31]	Endhost	Path's queue length	None	Flow	NS-2	Excessive probe packets
DeFlow	Switch	Port's queue length	Packet interval, packet size	Flowlet	NS-3	Minimal reordering of large flows



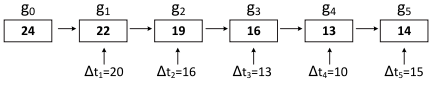
(a) Packet intervals in flowlet fl_1.



(b) Packet intervals in flowlet fl_2.



(c) The calculation process of g for fl_1.



(d) The calculation process of g for fl_2.

Fig. 4. Illustration of algorithm FDA ($\alpha = 12$).

Algorithm 1: Flowlet Differentiating Algorithm

Input: pkt: the arriving packet; fl: the flowlet pkt belongs to.
Output: $\mathcal{F}$: whether fl belongs to the large flow.

$l \leftarrow pkt.size$
$\Delta t \leftarrow pkt.t - fl.t$ ▷ $fl.t$ is the last packet's arriving time
if $fl.v \neq false$ **then** ▷ fl already exists
  $s_{pre} \leftarrow fl.s$
  $g_{pre} \leftarrow fl.g$
  $s_{now} \leftarrow l \times \frac{k-1}{k} + s_{pre} \times \frac{1}{k}$
  $g_{now} \leftarrow \Delta t \times \frac{k-1}{k} + g_{pre} \times \frac{1}{k}$
else ▷ fl does not exist
  $s_{now} \leftarrow \frac{1}{2} \times \beta$
  $g_{now} \leftarrow 2 \times \alpha$
$fl.s \leftarrow s_{now}$ ▷ update fl
$fl.g \leftarrow g_{now}$
if $g_{now} \leq \alpha$ **and** $s_{now} \geq \beta$ **then**
  $\mathcal{F} \leftarrow true$ ▷ fl belongs to the large flow
else
  $\mathcal{F} \leftarrow false$
return $\mathcal{F}$

average packet intervals and sizes to identify the large flow's flowlet fl_1 in Fig. 4(a). However, this tends to overreact to the situation illustrated in Fig. 4(b), where the small flow's flowlet fl_2 includes six packets and the adjacent-packet gaps are $g_{0\sim5} = \langle \frac{20}{1}, \frac{16}{1}, \frac{13}{1}, \frac{10}{1}, \frac{15}{1} \rangle$. Since $g_4 < \alpha = 12$, this scheme would misclassify the flowlet fl_2 as a large flow's flowlet according to the temporary condition fulfillment. When identifying the flow size, we work on the same flowlets. However, different algorithms may yield different results.

In order to promptly and accurately identify flowlets of large flows, FDA uses the weighted moving averages shown in Eqs. (1) and (2) to calculate the smoothed interval g_i and size s_i, respectively, at the time t_i when the $(i+1)$th packet p_i of flow f arrives with the size l_i, where k is a positive integer and Δt_i equals to $(t_i - t_{i-1})$. As a safeguard, before the first packet p_0 of the flow arrives, we configure $g_0 > \alpha$ and $s_0 < \beta$ (e.g., $g_0 = 2\alpha, s_0 = \frac{\beta}{2}$).

$$\begin{aligned} g_i &= \Delta t_i \times \frac{k-1}{k} + g_{i-1} \times \frac{1}{k} \\ &= \Delta t_i \times \frac{k-1}{k} + \Delta t_{i-1} \times \frac{k-1}{k^2} + g_{i-2} \times \frac{1}{k^2} \\ &= (k-1)\left(\frac{\Delta t_i}{k} + \frac{\Delta t_{i-1}}{k^2} + \cdots + \frac{\Delta t_1}{k^i}\right) + \frac{g_0}{k^i} \\ &= \left|\Delta t_i, \Delta t_{i-1}, \ldots, \Delta t_1, g_0\right| \cdot \begin{bmatrix} \frac{k-1}{k} \\ \frac{k-1}{k^2} \\ \vdots \\ \frac{k-1}{k^i} \\ \frac{1}{k^i} \end{bmatrix} \end{aligned} \tag{1}$$

$$\begin{aligned} s_i &= l_i \times \frac{k-1}{k} + s_{i-1} \times \frac{1}{k} \\ &= l_i \times \frac{k-1}{k} + l_{i-1} \times \frac{k-1}{k^2} + s_{i-2} \times \frac{1}{k^2} \\ &= (k-1)\left(\frac{l_i}{k} + \frac{l_{i-1}}{k^2} + \cdots + \frac{l_1}{k^i} + \frac{l_0}{k^{i+1}}\right) + \frac{s_{-1}}{k^{i+1}} \\ &= \left|l_i, l_{i-1}, \ldots, l_1, l_0, s_{-1}\right| \cdot \begin{bmatrix} \frac{k-1}{k} \\ \frac{k-1}{k^2} \\ \vdots \\ \frac{k-1}{k^i} \\ \frac{k-1}{k^{i+1}} \\ \frac{1}{k^{i+1}} \end{bmatrix} \end{aligned} \tag{2}$$

Eq. (1) shows that g_i can be expressed as a product of two matrices, $\mathbb{S} \times \mathbb{T}$, in which $\mathbb{S}[i] \cdot \mathbb{T}[i]$ represents the part of p_i and $\sum_{j=0}^{i-1} \mathbb{S}[j] \cdot \mathbb{T}[j]$ corresponds to the last $i-1$ packets. Meanwhile, we obtain the relationship shown in Eq. (3):

$$\forall i \in \mathbf{N}, k \geq 1 \quad \frac{1}{k^i} + \sum_{j=1}^{i} \frac{k-1}{k^j} = \frac{1}{k^{i+1}} + \sum_{j=1}^{i+1} \frac{k-1}{k^j} = 1 \tag{3}$$

Eq. (3) shows that FDA does not treat every packet in the lifespan of a flowlet equally. Instead, it assigns greater weight to recent packets, which better reflects the latest properties of the flowlet; older packets are assigned smaller weight values accordingly. By adjusting the effect weights different packets have on g, g becomes more accurate to reflect the current state of the flowlet. k can be viewed as the rate of packet aging, with larger values indicating a faster aging rate. g reflects the characteristics of the flowlet over a recent bout of time while primarily focusing on its current characteristics, which helps FDA avoid the delayed response and overreaction.

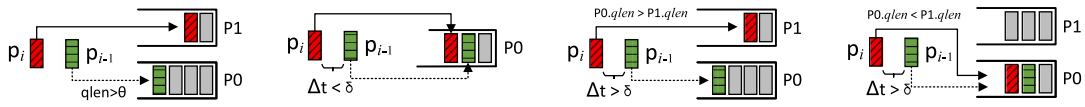
We use the example in Fig. 4(c) to illustrate the specific calculation process of g in FDA and list the corresponding pseudo code in Algorithm 1. In Fig. 4(c), we set the aging coefficient $k = 2$ and the flowlet differentiation threshold $\alpha = 12$. When the first packet p_0 arrives at time $t_0 = 0$, g_0 is set to $2\alpha = 24$. Next, p_1 arrives at time $t_1 = 20$, the interval between p_1 and p_0 is calculated as $\Delta t_1 = t_1 - t_0 = 20$. According to Eq. (1), $g_1 = \Delta t_1 \times \frac{2-1}{2} + g_0 \times \frac{1}{2}$. Following this process, when packets p_2, p_3, p_4, and p_5 arrive, the corresponding g_2, g_3, g_4, and g_5 are 20, 15, 10, and 9 respectively. Since $g_4 < \alpha$, the flowlet fl_1 is determined to belong to a large flow when p_4 arrives. As shown in Fig. 4(c), even though we set the initial value of g to a conservative value 2α and the intervals of the first three packets are large, g can still quickly reach the threshold when the packet arriving rate increases. Fig. 4(d) shows the g's calculation process of fl_2. Despite the transient increases in packet arriving rates at the time t_4 (i.e., $\Delta t_4 < \alpha$), FDA maintains $g_4 > \alpha$ due to its consideration of the currently arriving packet p_4 and the previous three packets.

As shown in Algorithm 1, FDA mainly involves three parameters to be configured in practical deployment: the packet aging rate k, the packet interval threshold β, and packet size threshold α for identifying flowlets of large flows. First, a larger k ensures that the average packet interval g and packet size s in FDA reflect the most recent characteristics of the flowlet. In extreme cases, as k approaches infinity, g and s are equal to the interval and size of the currently arriving packet, respectively. This is particularly beneficial during frequent micro-bursts, as a larger k is preferred to prevent delays in identifying large flows. Conversely, If minimizing packet reordering is the primary concern, lowering k will ensure that only flows with sustained large packets are classified as large flows. However, this may hinder the proactive alleviation of congestion. Second, the values of α and β should be aligned with the characteristics of the underlying physical network or the definition of large flows. For example, if a flow is classified as large flow when it occupies more than γ of a port's bandwidth B, then $\frac{\alpha}{\beta}$ should be set greater than $\gamma \times B$. In the future, we plan to explore AI-based methods [62,63] to more proactively and flexibly adjust the parameters in FDA.

4.2. Flowlet switching algorithm

After determining whether the flowlet fl belongs to a large or a small flow, DeFlow uses FSA to apply a differentiated strategy to it. If fl belongs to a large flow and the path it uses is experiencing congestion, FSA switches fl to another randomly selected candidate port, regardless of whether fl has reached the flowlet timeout. For the other scenarios, FSA considers both the state of previously and currently selected ports, as well as the characteristics of fl before deciding whether or not to change the port for fl.

We illustrate the differentiated path-switching strategy of FSA using the example in Fig. 5 and list the corresponding pseudo code in Algorithm 2. In Fig. 5, p_{i-1} and p_i are respectively the last and current packet



(a) Instant switching. (b) Instant keeping. (c) Conditional switching. (d) Conditional keeping.

Fig. 5. Illustration of algorithm FSA ($\theta = 2$).

Algorithm 2: Flowlet Switching Algorithm

Input: pkt: the arriving packet.
Output: $\mathcal{P}$: the egress port for pkt.

$fl \leftarrow$ FlowletLookup(pkt)
$\mathcal{F} \leftarrow$ FlowletDiffAlg(fl, pkt) ▷ classify the fl
$\mathcal{P}_{rnd} \leftarrow$ RoutingLookup(pkt) ▷ randomly select a port
if $fl.v \neq true$ **then** ▷ the first packet of a flow
  $fl.v \leftarrow true$
  $\mathcal{P} \leftarrow \mathcal{P}_{rnd}$
else
  $\mathcal{P}_{pre} \leftarrow fl.p$ ▷ $\mathcal{P}_{pre}$ is the last selected port
  $\Delta t = pkt.t - fl.t$
  if $\mathcal{P}_{pre}.qlen > \theta$ **then** ▷ $\mathcal{P}_{pre}$ is congested
    if $\mathcal{F} \neq false$ **then** ▷ fl belongs to large flow
      $\mathcal{P} \leftarrow \mathcal{P}_{rnd}$ ▷ instant switching
  else if $\Delta t \leq \delta$ **then**
    $\mathcal{P} \leftarrow \mathcal{P}_{pre}$ ▷ instant keeping
  else
    if $\mathcal{P}_{rnd}.qlen > \mathcal{P}_{pre}.qlen$ **then**
      $\mathcal{P} \leftarrow \mathcal{P}_{pre}$ ▷ conditional keeping
    else
      $\mathcal{P} \leftarrow \mathcal{P}_{rnd}$ ▷ conditional switching
$fl.t \leftarrow pkt.t$
$fl.p \leftarrow \mathcal{P}$
return $\mathcal{P}$

of the flowlet fl, $\mathcal{P}_0$ is the egress port to which p_{i-1} was forwarded and $\mathcal{P}_1$ is a randomly selected port for p_i. In determining whether p_i uses $\mathcal{P}_0$ or switches to $\mathcal{P}_1$, FSA divides the process into four cases:

- **Instant switching**: As shown in Fig. 5(a), if fl is determined to belong to a large flow and $\mathcal{P}_0$ is congested (i.e., the queue length of $\mathcal{P}_0$ exceeds the preset congestion threshold θ), FSA changes fl's egress port to $\mathcal{P}_1$, regardless of the interval between p_{i-1} and p_i.
- **Instant keeping**: As shown in Fig. 5(b), if fl is determined to not belong to a large flow and the interval between p_{i-1} and p_i is less than the flowlet timeout δ, FSA keeps fl's egress port $\mathcal{P}_0$ unchanged, without considering whether $\mathcal{P}_0$ is congested or not.
- **Conditional switching**: As shown in Fig. 5(c), if fl is determined to not belong to a large flow, and the interval between p_{i-1} and p_i is larger than δ, with the additional condition that $\mathcal{P}_1$ is less congested than $\mathcal{P}_0$ (i.e., $\mathcal{P}_1.qlen < \mathcal{P}_0.qlen$), FSA switches the egress port of fl to $\mathcal{P}_1$.
- **Conditional keeping**: As shown in Fig. 5(d), if fl is determined to not belong to a large flow and the interval between p_{i-1} and p_i is larger thanδ, with the additional condition that $\mathcal{P}_1$ is more congested than $\mathcal{P}_0$, FSA keeps fl egress port $\mathcal{P}_0$ unchanged.

The rationale for the approach is as follows. When congestion occurs at port $\mathcal{P}_0$, switching flowlet fl of large flows using it to other ports can rapidly alleviate its congestion. However, the packet interval of fl may prevent identifying the suitable switching opportunities in a short time, leaving the congestion episode unattended at $\mathcal{P}_0$. To counteract this, *instant-switching* does not wait for the flowlet switching opportunities and immediately alters the transmission path for fl. While this may result in temporary packet reordering for a few large flows, it can alleviate or even eliminate the congestion at $\mathcal{P}_0$, which not only allows lots of latency-sensitive small flows passing through $\mathcal{P}_0$ to complete their transmissions sooner, but also allows throughput-sensitive large flows to find another high-throughput path to transmit at high speeds over a long period of time. The negative effect of packet reordering on large flows is easily absorbed.

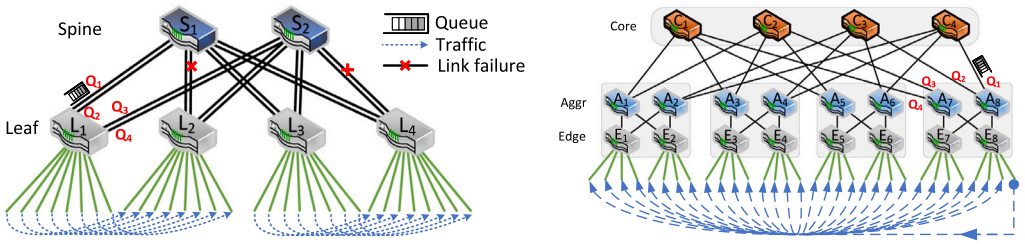
On the other hand, the traffic of small flows is minor as shown in Fig. 2(a). Hence, removing small flows from the congested port $\mathcal{P}$ or reducing its sending rate cannot help alleviate congestion at $\mathcal{P}$ much. Besides, the potential packet reordering caused by instantly switching a flow's egress port is much more detrimental to small flows by significantly increasing their completion time. Therefore, if and only if the packet interval of small flows reaches the flowlet timeout, will FSA consider to switch their paths. It takes into account both the queue lengths of the previously and newly selected ports. This approach prevents the packet reordering and allows the small flows to complete faster.

4.3. Implementation

DeFlow can be implemented on programmable switches widely deployed in the datacenter networks. From the algorithm description of FDA and FSA, the primary functionalities involve flow table lookups, register reads, and arithmetic operations, which are well supported by commercial programmable switches such as Tofino-1 and Tofino-2. Implementing DeFlow in programmable switches faces two challenges. (1) Some switches' data planes do not support the multiplication and division operations which are needed by the aging process in FDA; (2) FSA requires acquiring the queue length of the egress port during the ingress stage of the pipeline.

In response to the first challenge, we approximate the multiplication and division operations using bit shifting and addition/subtraction operations, which are supported by most programmable switches. For instance, $\Delta t \times \frac{1}{2}$ and $\Delta t \times \frac{1}{3}$ can be approximated as $\Delta t >> 1$ and $\Delta t >> 2 + \Delta t >> 4$, respectively. As for acquiring the queue length of the egress port at the ingress pipeline, many programmable switches such as Tofino-2 already support this feature by periodically writing the queue lengths of all egress ports into a set of registers in the ingress pipeline. For the programmable switches which can only obtain the queue length at the egress pipeline, we can realize this functionality using mechanisms such as packet looping/replication.

An alternative approach is to encapsulate the queue length at the egress stage of switch S into the packets sent to the neighboring switch S′ where the information is stored and piggybacked by the packets towards S through the reverse link. Given each port has a buffer size of 1024 KB and the average packet size is 1500 Bytes, adding a $(\log_2 1024)$-bit custom field to the original packet header would allow adjacent switches to exchange the queue lengths of their connected ports. This approach results in bandwidth overhead of only $\frac{\log_2 1024}{1500\times8}\times100\% = 0.08\%$. Additionally, each switch with n ports requires just $n \times \log_2 1024$ bits of SRAM to store the queue lengths of the connected ports of adjacent switches. Clearly, both the bandwidth and resource consumption of this approach are negligible.



(a) Spine-Leaf network with pattern Ring. (b) Fat-Tree network with pattern All2all.

Fig. 6. Two typical DCN network topologies with two typical communication patterns.

4.3.1. Analysis

Using LetFlow as a baseline, we analyze the additional overhead introduced by DeFlow's two algorithms FSA and FDA when deployed on switches. Note we exclude those shared components, such as the routing table for random port selection.

Both LetFlow and DeFlow utilize a flowlet lookup table to identify the flowlet associated with each packet. However, LetFlow only records the departure time and egress port of the last packet for each flowlet. In contrast, DeFlow requires additional information: the average packet size and interval for each flowlet to facilitate large flow detection in FDA. Consequently, DeFlow requires an additional $m \times w_s \times w_i$ bits of register resources, where m is the number of recorded flowlets, w_s and w_i represent the bit width for packet size and interval, respectively. On the other hand, LetFlow performs two comparisons to check flowlet existence and decide on egress port. In contrast, as shown in Algorithms 1 and 2, DeFlow needs additional 2 and 4 comparisons to identify large flows and egress port, respectively. Therefore, DeFlow's flowlet lookup table requires $\frac{2+4+2}{2}$ times more ALU resources to execute comparison action than LetFlow.

In terms of time complexity, DeFlow's operations including table lookups and register read/write can be completed in O(1) time, incurring no impact on line-rate processing capabilities of hardware switches.

5. Implementation and evaluation

We evaluate the effectiveness of DeFlow through extensive simulations using the network simulator NS-3, comparing its performance (e.g., FCT, throughput, and queue length) with other state-of-the-art load balancing schemes, such as RPS, ECMP, and LetFlow, which schedule paths at different granularities: packet, flow, and flowlet, respectively. Additionally, we implement the hardware prototypes on the Tofino-based EdgeCore Wedge 100BF-32X switch [64] to compare their resource consumption, demonstrating DeFlow's scalability in large-scale datacenter networks. The implementations of these existing load balancing schemes in both the software simulator and the hardware switches are based on the prior works [65,66].

5.1. Testbed

5.1.1. Topologies

We conduct simulations on two typical DCN topologies: Spine-Leaf and Fat-Tree, as depicted in Fig. 6. The Spine-Leaf network consists of two spine switches, four leaf switches, and 32 servers. As illustrated in Fig. 6(a), there are eight equal-cost paths for inter-leaf communications in the Spine-Leaf network. The Fat-Tree network in Fig. 6(b) comprises four pods with a total of 32 servers and 20 switches. The edge and aggregation switches within each pod are interconnected through four links to form a full mesh, while the aggregation and core switches across the pods are interconnected through 16 links to form another level of full mesh. Therefore, the intra-pod and inter-pod communication in Fat-Tree network has two and four paths, respectively. All links are configured with a capacity of 10 Gbps and the oversubscription ratio of both the Spine-Leaf and Fat-Tree networks is 1:2, a typical value in real-world datacenters. Additionally, we also test the performance of load balancing schemes under asymmetric topologies, where two links experience failures [67] as indicated by the red cross in Fig. 6(a).

Table 2
Two typical communication patterns in our testbed.

Communication pattern	Ring	All2all
Topology	Spine-Leaf	Fat-Tree
Sender→Receiver	s[i]→ s[(i+8)%32], i∈[1,32]	s[i]→ s[j], $i \neq j \in$[1,32]

5.1.2. Workloads

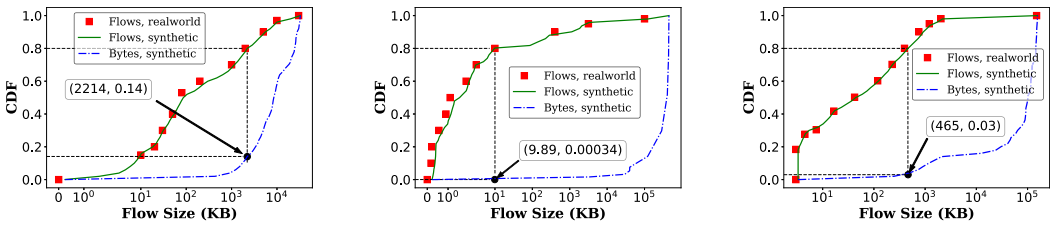
The performance evaluation is based on the workloads derived from three representative applications in datacenter environments: Web Search (WS) from a production cluster [14], Data Mining (DM) from a large enterprise datacenter [47], and Remote Procedure Call (RPC) from a laboratory cluster [68]. Their traffic characteristics are shown in Fig. 7, in which the rectangular red dots represent the real-world traffic distribution of the applications, while the green solid line and the blue dashed line depict the distribution of the synthetic traffic generated based on the real-world traffic. It can be observed from Fig. 7 that the characteristics of the synthetic traffic closely match that of the real-world traffic. Notably, all three types of traffic show strong locality, with just 20% of large flows contributing to 86%, 99.96%, and 97% of the total traffic volume for WS, DM and RPC, respectively.

5.1.3. Communication patterns

We select two communication patterns widely used in the distributed AI training: Ring for the Spine-Leaf network and All2all for the Fat-Tree network, as shown in Table 2. In the Ring pattern, the traffic is consistently sent from the ith server $s[i]$ connected to the leaf switch $L_{\frac{i}{8}}$ to the server $s[(i+8)\%32]$ connected to the leaf switch $L_{\frac{(i+8)\%32}{8}}$. In the All2all pattern, every server sends traffic to all other servers.

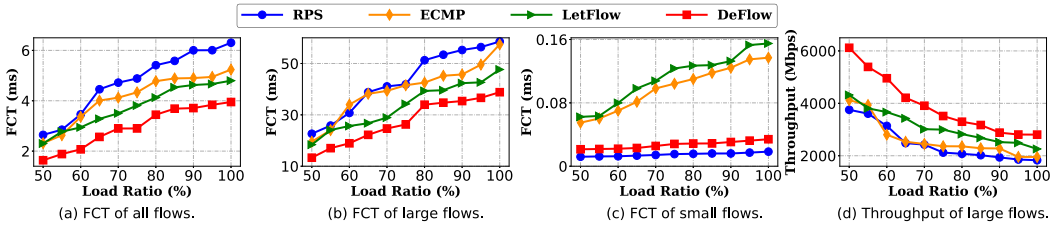
5.1.4. Parameters

Given that a large flowlet timeout δ can reduce the likelihood of packet reordering but increase the probability of load imbalance, we set the value of δ on the order of RTT (50 us) according to the suggestion in [18,46], which can effectively fragment bursts into flowlets, striking a desirable balance between packet reordering and load balancing. For DeFlow, based on the statistical analysis of traffic in Figs. 2 and 7, we set the thresholds α and β for identifying large flow's flowlet to 2 and 1000, respectively, i.e., if the average size s and adjacent-packet interval g of the mostly recent packets within the same flowlet exceeds 1000 bytes and less than 2 us, respectively, it is identified as belonging to a large flow. Furthermore, to ensure that newly arriving flows are not immediately classified as large flows, we configure the initial values for new flows (i.e., s_0 and g_0) to be 2α and $\frac{\beta}{2}$, respectively. Additionally, we set the rate of packet aging k to 2, allowing s and g to encompass the flowlet characteristics over a period but put more emphasis on the most current traits.



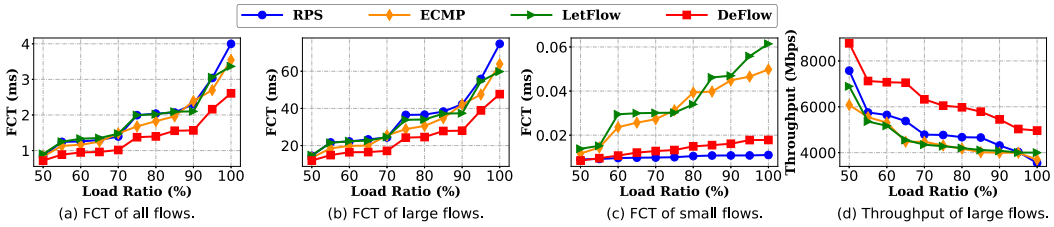
(a) Web search. (b) Data mining. (c) Remote procedure call.

Fig. 7. The characteristics of three real-world workloads used in our testbed.



(a) FCT of all flows. (b) FCT of large flows. (c) FCT of small flows. (d) Throughput of large flows.

Fig. 8. The FCT and throughput across different load ratios on [WS,Spine-Leaf,Ring].



(a) FCT of all flows. (b) FCT of large flows. (c) FCT of small flows. (d) Throughput of large flows.

Fig. 9. The FCT and throughput across different load ratios on [RPC,Spine-Leaf,Ring].

5.1.5. Metrics

We mainly compare DeFlow with competing schemes in terms of FCT under various network topologies, application workloads, communication patterns, and load pressures. Further, since large and small flows have different sensitivities to throughput and latency, we separately compare the FCT of large flows (top 5%) and small flows (bottom 50%), as well as the throughput of large flows. In addition, to reflect the reordering pressure at the receivers, we evaluate the degree of the packet reorder of DeFlow with RPS and LetFlow, which randomly spray packets and flowlets, respectively. Meanwhile, we measure the queue lengths of specific switch ports to gauge the effectiveness that each scheme manages congestion avoidance and mitigation. As shown in Fig. 6, we select the four ports of the first leaf switch and the last two aggregation switches for the Spine-Leaf and Fat-Tree network, respectively, which handle all the traffic sent from the eight downstream servers. Furthermore, we analyze the resource consumption on Tofino switch to illustrate the hardware cost associated with the different load balancing schemes.

5.2. Simulation results

5.2.1. Overall FCT

Figs. 8(a)~13(a) depict the average FCT of all flows in RPS, ECMP, LetFlow, and DeFlow as the load ratio varies from 50% to 100%, in which [X, Y, Z] indicates the combination of the workload X, the topology Y, and the communication pattern Z. As shown in Fig. 8(a), ECMP is better than RPS in the Spine-Leaf network under the Ring pattern but performs worse than RPS in the Fat-Tree network with the All2all pattern (Fig. 11(a)), indicating the limited adaptability of RPS and ECMP. In contrast, DeFlow consistently achieves the smallest overall FCT among all the schemes under all the combinations. At 90% load in Fig. 10, DeFlow reduces the overall FCT of RPS, ECMP and LetFlow by 1.56×, 1.41×, and 1.35×, respectively. Note that DeFlow prioritizes small flows over larger flows on FCT reduction when the congestion occurs. Even though, the above results demonstrate that DeFlow still decreases the overall FCT.

While RPS, ECMP and LetFlow achieve a good balance among different paths in terms of the number of packets, flows and flowlets, they do not take into consideration the significant size disparity between large and small flows and their performance preference. Moreover, they are oblivious to the path congestion status. DeFlow achieves superior performance through differential treatments to flowlets of small flows and large flows. It utilizes the FDA algorithm to identify whether each flowlet belongs to large flows or small flows, and employs the FSA algorithm to adopt different path-scheduling strategies for large and small flows based on the network state. Specifically, (1) When congestion occurs on a specific path, DeFlow swiftly redirects large flows on it to an alternative path, aiming for a rapid resolution of the congestion on this path. (2) If a small flow does not meet the flowlet switching threshold, DeFlow always keeps it on the original path to prevent packet reordering, irrespective of the path's state; otherwise, DeFlow will choose the less congested path for it to lower its FCT without causing packet reordering.

5.2.2. FCT of large flows

Figs. 8(b)~13(b) shows the average FCT of large flows. It is evident that, although DeFlow always moves large flows away from congested paths to ensure better FCT for small flows, the FCT of large flows in DeFlow still outperforms RPS, ECMP, and DeFlow in all six scenarios. As shown in Fig. 13(b), the FCT of large flows in DeFlow is

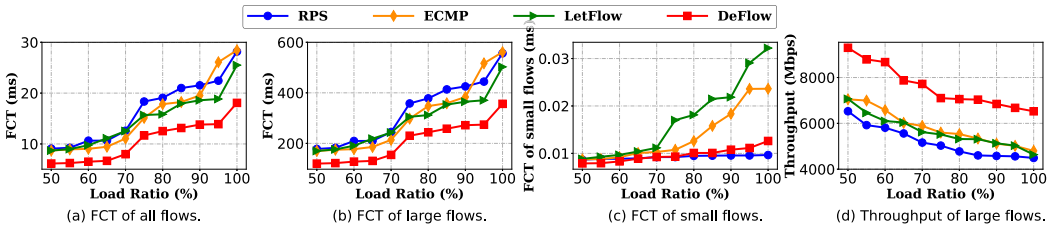


Fig. 10. The FCT and throughput across different load ratios on [DM,Spine-Leaf,Ring].

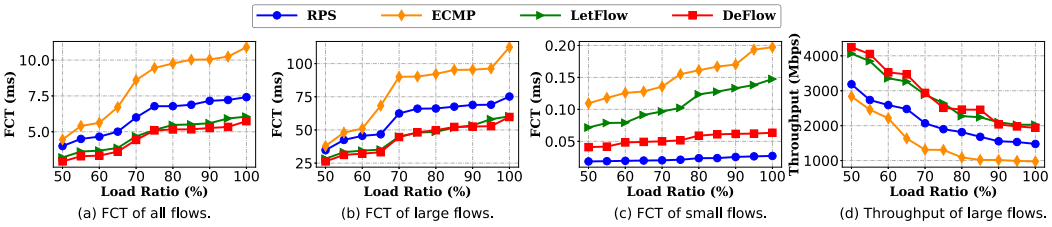


Fig. 11. The FCT and throughput across different load ratios on [WS,Fat-Tree,All2all].

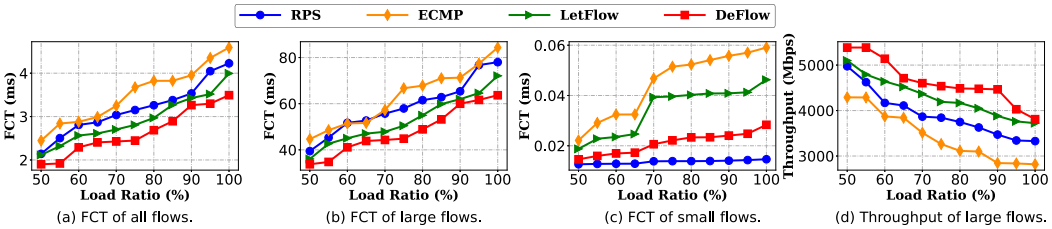


Fig. 12. The FCT and throughput across different load ratios on [RPC,Fat-Tree,All2all].

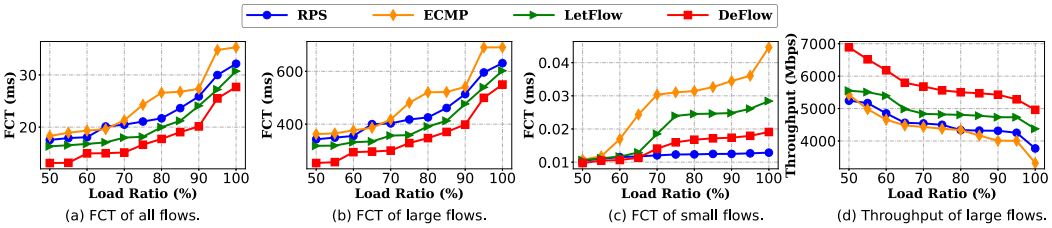


Fig. 13. The FCT and throughput across different load ratios on [DM,Fat-Tree,All2all].

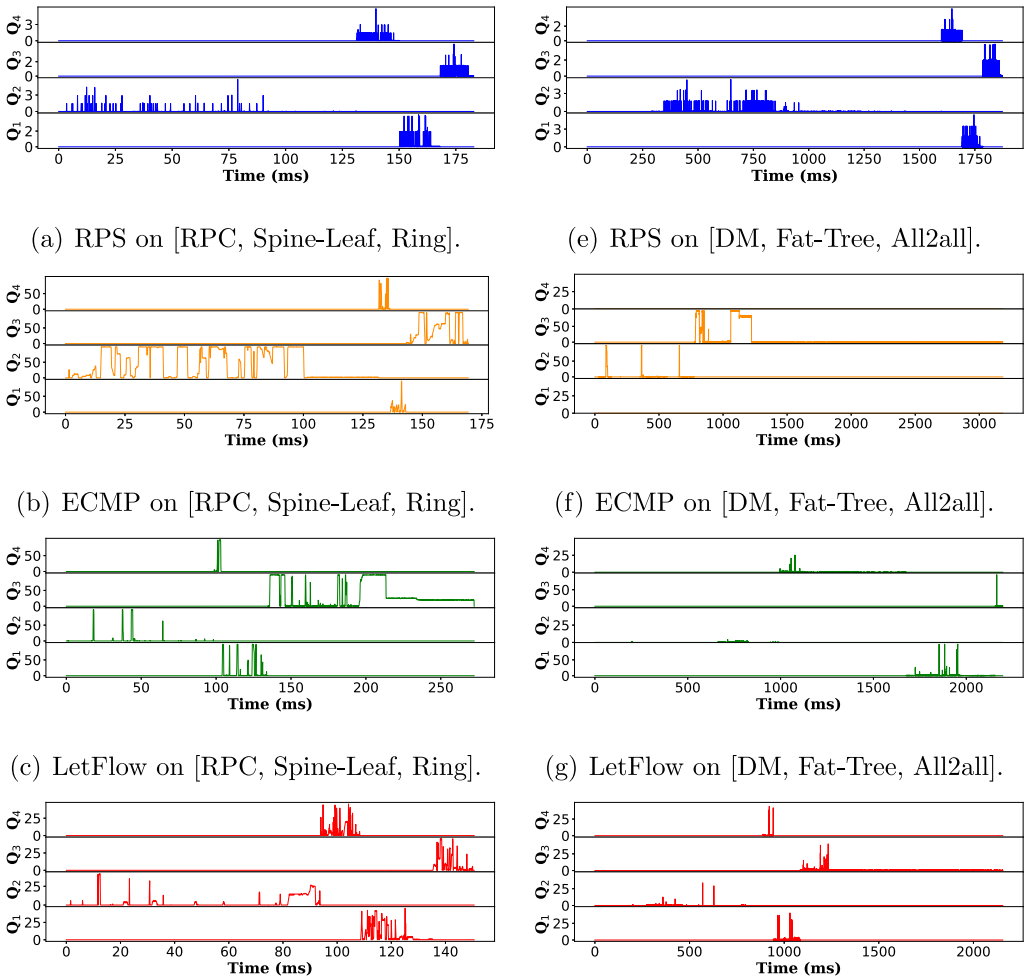
just 398.6 ms on the Fat-Tree network and All2all pattern, whereas RPS, ECMP, and LetFlow require 513.8 ms, 541.2 ms, and 476.6 ms, respectively. As shown in Fig. 2(c), the packet intervals of large flows are significantly shorter than that of small flows. Consequently, RPS causes severe packet reordering in large flows, which triggers frequent rate reductions and ultimately leads to much longer FCT for large flows. On the other hand, the very short packet intervals of large flows make it difficult for LetFlow to identify suitable opportunities to split flow into flowlets. Even during link congestion, LetFlow can only wait for proper chances to perform flowlet segmentation. As for ECMP, it does not take flow size into account when scheduling paths, making it likely for large flows to be assigned to the same path which leads to congestion when the combined traffic exceeds its capacity.

5.2.3. FCT of small flows

Figs. 8(c)~13(c) show the average FCT of small flows. It is evident that, across all six scenarios, the FCT of small flows of RPS and DeFlow is significantly lower than that of ECMP and LetFlow. As shown in Fig. 11(c), DeFlow improves the FCT of small flows by 2.76× and 2.16× than ECMP and LetFlow at 90% load, respectively. As revealed in Fig. 2(c), the packet intervals for small flows are quite large, meaning that randomly spraying packets across load-balanced paths rarely causes reordering. Therefore, RPS has a significant advantage in reducing FCT of small flows than ECMP and LetFlow. Note that LetFlow and DeFlow may encounter congestion due to difficulties in finding opportunities for flowlet segmentation. However, DeFlow addresses the issue by swiftly redirecting large flows during congestion to high-bandwidth paths, which effectively alleviates congestion without causing packet reordering for the numerous small flows.

5.2.4. Throughput of large flows

Figs. 8(d)~13(d) illustrate the average throughput of large flows. Clearly, as the load ratio increases, the throughput of large flows for all schemes drops significantly. Despite this, DeFlow consistently achieves the highest throughput for large flow across all six scenarios, maintaining a throughput of 4466 Mbps at 90% load in Fig. 12(d), while RPS, ECMP, and LetFlow achieve only 3471 Mbps, 2846 Mbps, and 3882 Mbps, respectively. The poor performance of RPS and ECMP can be attributed to packet reordering and large-flow collision, respectively. On the other hand, DeFlow's superiority over LetFlow demonstrates that proactively relocating large flows from congested links is



(a) RPS on [RPC, Spine-Leaf, Ring].

(e) RPS on [DM, Fat-Tree, All2all].

(b) ECMP on [RPC, Spine-Leaf, Ring].

(f) ECMP on [DM, Fat-Tree, All2all].

(c) LetFlow on [RPC, Spine-Leaf, Ring].

(g) LetFlow on [DM, Fat-Tree, All2all].

(d) DeFlow on [RPC, Spine-Leaf, Ring].

(h) DeFlow on [DM, Fat-Tree, All2all].

Fig. 14. The statistics of queue length of four ports marked in Figs. 6(a) and 6(b).

more beneficial for throughput, even if it causes occasional packet reordering.

5.2.5. Queue length of switch

To further investigate the reasons behind DeFlow's superior performance, we monitor four switch ports (Q_1~Q_4) marked in Figs. 6(a) and 6(b) and record their queue lengths in kilobytes (KB) every 100 microseconds. Figs. 14(a)~(d) and 14(b)~(h) illustrate the queue length statistics of different load balancing schemes at 80% load in the Spine-Leaf and Fat-Tree networks, respectively. Two aspects of queue length are crucial: the severity of queue congestion (i.e., the maximum value) and the duration of congestion (i.e., the average value), which reflect the capabilities of congestion avoidance and congestion mitigation, respectively. In RPS, congestion is mild and short-lived, with maximum queue lengths below 6 KB, as shown in Figs. 14(a) and 14(b). In contrast, ECMP and LetFlow experience significantly more severe congestion, with queue lengths peaking at 100 KB and persisting for longer duration. Once congestion occurs, ECMP relies on packet drops caused by queue overflows to trigger sender-side rate reduction, while LetFlow waits for flowlet timeouts to schedule traffic off the congested paths. DeFlow outperforms the others in both congestion severity and mitigation. By quickly shifting large flows away from congested paths, DeFlow is able to relieve congestion more effectively. For example, for the four ports [Q_1, Q_2, Q_3, Q_4] shown in Fig. 14(g), DeFlow's average queue length (KB) is [1.34, 1.61, 0.89, 0.70], with congestion duration exceeding 30 KB occurring [1.33, 0.60, 1.13, 0.66] of the time (%), compared to [0.53, 32.0, 6.7, 0.66] and [0.35, 34.5, 7.73, 0.83] for ECMP, and [1.59, 0.87, 15.7, 0.62] and [1.87, 0.88, 11.5, 0.62] for LetFlow.

5.2.6. FCT on asymmetric topology

We tested the performance of different load balancing schemes in asymmetric network topologies by creating two link failures shown in Fig. 6(a), and the results are shown in Fig. 15. It can be observed that, under the same conditions, the FCT of each load balancing scheme in an asymmetric network is longer than that in a symmetric network shown in Fig. 10, demonstrating the advantage of network symmetry. On the other hand, DeFlow demonstrates superior performance compared to RPS, ECMP and LetFlow. When the load ratio is 80%, it improves the FCT of all flows by about 1.23×, 1.14× and 1.03×, and improves the throughput of large flows by 1.40×, 1.16× and 1.13×, respectively, compared to RPS, ECMP and LetFlow. This once again fully demonstrates that the sacrifice of FCT of large flows made by DeFlow is minor and worthwhile. The results indicate that, under both symmetric and asymmetric networks, DeFlow outperforms RPS, ECMP and LetFlow.

5.2.7. Packet reorder degree

Fig. 16 provides the statistics on the packet reordering degree at the receiver for three spraying-based load balancing schemes, RPS, LetFlow, and DeFlow, under the 80% load in the Spine-Leaf and Fat-Tree

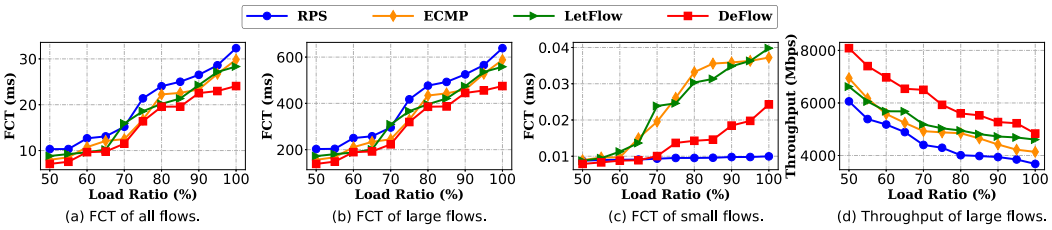
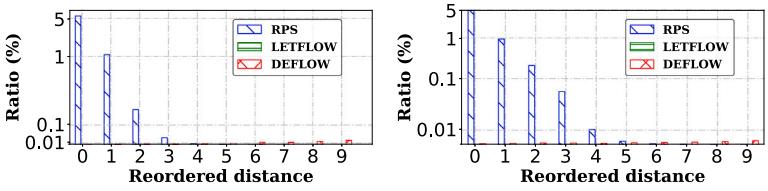


Fig. 15. The FCT and throughput on asymmetric network with DM workload.



(a) Reordering on [RPC, Spine-Leaf, Ring]. (b) Reordering on [DM, Fat-Tree, All2all].

Fig. 16. The reorder degree statistics of different load balancing schemes at 80% load.

networks. A received packet with the sequence number i is considered as reordered if the maximum sequence number of the previously received packets of the same flow is j and $i < j$. The reordering degree (i.e., distance) is calculated as $\frac{j-i}{s}$ where s is the packet size. Clearly, RPS results in a high likelihood of shallow reordering, where packets are frequently delivered slightly out of order. As illustrated in Figs. 16(a) and 16(b), the proportion of packets in RPS with a reordering degree of fewer than five is 6.5% in the Spine-Leaf network and 5.94% in the Fat-Tree network. LetFlow maintains sufficient gaps (100 us) between adjacent flowlets when spraying flowlets to accommodate latency differences across paths. Therefore, LetFlow experiences less packet reordering than RPS and DeFlow. DeFlow also has a low probability of packet reordering, with only 0.008% in the Spine-Leaf network and 0.003% in the Fat-Tree network. Note that the reorder degree in DeFlow is relatively higher. This is because DeFlow moves high-rate large flows from heavily congested path to uncongested path, which, although rare, can result in deep reordering. Nonetheless, the earlier experimental results on FCT, throughput, and queue length indicate that this trade-off is worthwhile.

5.3. Hardware prototype analysis

Using the network shown in Fig. 6(a) as the application scenario, we implement prototypes of RPS, ECMP, LetFlow, and DeFlow on a Tofino-based switch using 374, 374, 457, and 531 lines of P4 code, respectively.

Table 3 summarizes their hardware resource consumption. For RPS and ECMP, the destination IP as the key for looking up available ports, with RPS selecting one randomly and ECMP using a hash of the five-tuple (source IP, destination IP, source port, destination port, and protocol), respectively. In contrast, LetFlow and DeFlow use the five-tuple to identify and record every flowlet and assign it a randomly selected port. We evaluate the hardware resource consumption under 1024 and 65 536 concurrent flowlets. Note that the resource consumption of RPS and ECMP is independent of the number of flowlets and depends solely on the number of servers in the network.

As shown in Table 3, all the four load balancing schemes consume minimal hardware resources. For instance, for the most precious chip resource SRAM on Tofino chip, even with 65 536 concurrent flowlets, RPS, ECMP, LetFlow, and DeFlow only consume 0.7%, 0.7%, 3.6%, and 6.8% of the total SRAM resources, respectively. Since DeFlow needs to record the packet size and interval for each flowlet, it consumes more SRAM than RPS, ECMP, and LetFlow. Additionally, DeFlow performs multiple comparisons based on packet size and interval to decide whether to change the egress port, therefore it consumes 12.5% and 6.3% more ALU resources than RPS/ECMP and LetFlow, respectively. In summary, RPS, ECMP, LetFlow, and DeFlow can be fully deployed on programmable switches and consume very little hardware resources, leaving ample resources for other essential switch functions such as firewalls and traffic monitoring.

Table 3
Comparison of hardware resource consumption.

Resource type	1K flows			65K flows		
	RPS/ECMP	LetFlow	Deflow	RPS/ECMP	LetFlow	Deflow
Hash Bits	2.1%	3.0%	3.8%	2.1%	3.0%	3.8%
VLIW instruction	0.5%	1.6%	2.9%	0.5%	1.6%	2.9%
ALU	2.1%	8.3%	14.6%	2.1%	8.3%	14.6%
SRAM	0.7%	1.4%	2.2%	0.7%	3.6%	6.8%
Gateway	0.5%	2.1%	4.7%	0.5%	2.1%	4.7%
Match crossbar	1.5%	2.5%	3.7%	1.5%	2.5%	3.7%

6. Conclusion

We propose a new flowlet-based load-balancing algorithm DeFlow for datacenter networks. Unlike the previous solutions, DeFlow distinguishes large and small flows based on the realtime statistics on packet size and adjacent-packet interval, and adopts differential strategies for both types of flows which prioritize the transmission of small flows to meet their low latency requirement. Extensive experiments demonstrate that DeFlow improves the flow completion time of all and small flows by up to 1.56× and 2.76× respectively while improving the throughput of large flows by 1.57×. We implement DeFlow on programmable switches at low cost and demonstrate its outstanding performance gain over the other state-of-the-art schemes.

CRediT authorship contribution statement

Ying Wan: Writing – original draft, Methodology, Investigation, Funding acquisition, Formal analysis, Data curation, Conceptualization. **Haoyu Song:** Writing – review & editing, Project administration, Conceptualization. **Yu Jia:** Resources, Funding acquisition. **Yi Wang:** Writing – review & editing, Validation, Investigation. **Ling Qian:** Supervision, Resources. **Tian Pan:** Writing – review & editing, Supervision, Project administration, Conceptualization.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

The work of Ying Wan and Ling Qian was supported by the National Nature Science Foundation of China under Grant U21B2022, and the work of Tian Pan was supported by the National Nature Science Foundation of China under Grant 62372053.

Data availability

Data will be made available on request.

References

[1] M. Salimian, M. Ghobaei-Arani, A. Shahidinejad, An evolutionary multi-objective optimization technique to deploy the IoT services in fog-enabled networks: an autonomous approach, Appl. Artif. Intell. 36 (1) (2022) 2008149.
[2] A. Montazerolghaem, Efficient resource allocation for multimedia streaming in software-defined internet of vehicles, IEEE Trans. Intell. Transp. Syst. (2023).
[3] M. Al-Fares, A. Loukissas, A. Vahdat, A scalable, commodity datacenter network architecture, CCR (2008).
[4] M. Alizadeh, T. Edsall, On the data path performance of leaf-spine datacenter fabrics, in: HOTI, 2013, pp. 71–74.
[5] A. Greenberg, et al., VL2: A scalable and flexible data center network, in: SIGCOMM, 2009, pp. 51–62.
[6] H. Jiang, C. Dovrolis, Why is the internet traffic bursty in short time scales? in: SIGMETRICS, 2005, pp. 241–252.
[7] G.Y. Lazarou, J. Baca, V.S. Frost, J.B. Evans, Describing network traffic using the index of variability, TON (2009).
[8] S. Liu, X. Lin, Z. Guo, Y. Wang, M.A. Serhani, Y. Xu, Optimizing flow completion time via adaptive buffer management in data center networks, in: ICPP, 2021, pp. 1–10.
[9] R. Durner, W. Kellerer, Network function offloading through classification of elephant flows, TNSM (2020).
[10] C. Hopps, Analysis of an equal-cost multi-path algorithm, in: RFC 2992, 2000, http://dx.doi.org/10.17487/RFC2992, URL https://www.rfc-editor.org/info/rfc2992.
[11] F. Jazayeri, A. Shahidinejad, M. Ghobaei-Arani, A latency-aware and energy-efficient computation offloading in mobile fog computing: a hidden Markov model-based approach, J. Supercomput. 77 (2021) 4887–4916.
[12] A. Shahidinejad, F. Farahbakhsh, M. Ghobaei-Arani, M.H. Malik, T. Anwar, Context-aware multi-user offloading in mobile edge computing: a federated learning-based approach, J. Grid Comput. 19 (2) (2021) 18.
[13] S. Ghorbani, et al., Drill: Micro load balancing for low-latency data center networks, in: SIGCOMM, 2017, pp. 225–238.
[14] M. Alizadeh, A. Greenberg, D.A. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta, M. Sridharan, Data center tcp (DCTCP), in: SIGCOMM, 2010, pp. 63–74.
[15] A. Dixit, et al., On the impact of packet spraying in data center networks, in: INFOCOM, IEEE, 2013, pp. 2130–2138.
[16] L. Giraudo, R. Birke, A. Bianco, S. Atalla, et al., A load balancer for a multi-stage router architecture, IJCA (2014).
[17] K. He, et al., Presto: Edge-based load balancing for fast datacenter networks, CCR (2015).
[18] E. Vanini, et al., Let it flow: Resilient asymmetric load balancing with flowlet switching, in: NSDI, 2017, pp. 407–420.
[19] C.K. Song, H. Song, C. Qian, Dynamic and load-aware flowlet for load-balancing in data center networks, IPCCC (2023).
[20] Cisco, Congestion-awarw packet spraying, 2023.
[21] Broadcom, Broadcom unveils industry's highest performance fabric for AI networks, 2023, https://www.broadcom.com/company/news/product-releases/61156.
[22] UEC, UEC progresses towards v1.0 set of specifications, 2024, https://ultraethernet.org/uec-progresses-towards-v1-0-set-of-specifications/.
[23] W. Wang, et al., Freeway: Adaptively isolating the elephant and mice flows on different transmission paths, in: ICNP, IEEE, 2014, pp. 362–367.
[24] A.R. Curtis, W. Kim, P. Yalagandula, Mahout: Low-overhead datacenter traffic management using end-host-based elephant detection, in: INFOCOM, 2011, pp. 1629–1637.
[25] R. Tu, X. Wang, J. Zhao, Y. Yang, L. Shi, T. Wolf, Design of a load-balancing middlebox based on SDN for data centers, in: INFOCOM WKSHPS, 2015, pp. 480–485.
[26] Z. Guo, M. Su, Y. Xu, Z. Duan, L. Wang, S. Hui, H.J. Chao, Improving the performance of load balancing in software-defined networks through load variance-based synchronization, Comput. Netw. (2014).
[27] Z. Guo, S. Hui, Y. Xu, H.J. Chao, Dynamic flow scheduling for power-efficient data center networks, in: IWQoS, 2016, pp. 1–10.
[28] Z. Guo, Y. Xu, Y.-F. Liu, S. Liu, H.J. Chao, Z.-L. Zhang, Y. Xia, AggreFlow: Achieving power efficiency, load balancing, and quality of service in data center networks, TON (2020).
[29] J. Zhang, F.R. Yu, S. Wang, T. Huang, Z. Liu, Y. Liu, Load balancing in data center networks: A survey, COMST (2018).
[30] A. Kabbani, B. Vamanan, J. Hasan, F. Duchene, Flowbender: Flow-level adaptive routing for improved latency and throughput in datacenter networks, in: CoNEXT, 2014, pp. 149–160.
[31] N. Katta, A. Ghag, M. Hira, I. Keslassy, A. Bergman, C. Kim, J. Rexford, Clove: Congestion-aware load balancing at the virtual edge, in: CoNEXT, 2017, pp. 323–335.
[32] M.A. Qureshi, et al., PLB: Congestion signals are simple and effective for network load balancing, in: SIGCOMM, 2022, pp. 207–218.
[33] Y. Zhu, et al., Congestion control for large-scale RDMA deployments, CCR (2015).
[34] R. Mittal, et al., TIMELY: RTT-based congestion control for the datacenter, CCR (2015).
[35] P. Goyal, P. Shah, N.K. Sharma, M. Alizadeh, T.E. Anderson, Backpressure flow control, in: BC, 2019, pp. 1–3.
[36] T.R. Henderson, et al., Network simulations with the NS-3 simulator, SIGCOMM Demonstr. (2008).
[37] The CAIDA UCSD anonymized internet traces[20180315], 2018, www.caida.org/data/passive/passive_dataset.xml. (Accessed March 2018).
[38] H. Wang, W. Ding, Z. Xia, A cloud-pattern based network traffic analysis platform for passive measurement, in: CSC, 2012, pp. 1–7.
[39] W. mao, et al., Facilitating network functions virtualization by exploring locality in network traffic: A proposal, in: CSAI, 2018, pp. 495–499.
[40] B. Yan, Y. Xu, H. Xing, K. Xi, H.J. Chao, Cab: A reactive wildcard rule caching system for software-defined networks, in: HotSDN, 2014, pp. 163–168.
[41] J. Wallerich, A. Feldmann, Capturing the variability of internet flows across time, in: INFOCOM, 2006, pp. 1–6.
[42] J. Zhou, M. Tewari, M. Zhu, A. Kabbani, L. Poutievski, A. Singh, A. Vahdat, WCMP: Weighted cost multipathing for improved fairness in data centers, in: EuroSys, 2014, pp. 1–14.
[43] C.-M. Cheung, K.-C. Leung, DFFR: a flow-based approach for distributed load balancing in data center networks, Comput. Commun. 116 (2018) 1–8.
[44] S. Kandula, D. Katabi, S. Sinha, A. Berger, Dynamic load balancing without packet reordering, CCR (2007).
[45] S. Sen, D. Shue, S. Ihm, M.J. Freedman, Scalable, optimal flow routing in datacenters via local link balancing, in: CoNEXT, 2013, pp. 151–162.
[46] Z. Liu, et al., Burstbalancer: Do less, better balance for large-scale data center traffic, TPDS (2023).
[47] M. Alizadeh, et al., CONGA: Distributed congestion-aware load balancing for datacenters, in: SIGCOMM, 2014, pp. 503–514.
[48] N. Katta, M. Hira, C. Kim, A. Sivaraman, J. Rexford, Hula: Scalable load balancing using programmable data planes, in: SOSR, 2016, pp. 1–12.
[49] J. Wang, S. Rao, Y. Liu, P.K. Sharma, J. Hu, Load balancing for heterogeneous traffic in datacenter networks, J. Netw. Comput. Appl. 217 (2023) 103692.
[50] Y. Lu, Z. Xu, X. Ma, FAMG: A flow-aware and mixed granularity method for load-balancing in data center networks, Comput. Commun. 209 (2023) 415–428.
[51] J. Zhang, W. Liu, C. Zheng, Z. Guo, Load balancing based on flow classification with private link in programmable switch, in: 2024 9th International Conference on Computer and Communication Systems, ICCCS, IEEE, 2024, pp. 646–651.
[52] J. Hu, C. Zeng, Z. Wang, J. Zhang, K. Guo, H. Xu, J. Huang, K. Chen, Load balancing with multi-level signals for lossless datacenter networks, IEEE/ACM Trans. Netw. (2024).
[53] M. Al-Fares, et al., Hedera: dynamic flow scheduling for data center networks, in: NSDI, 2010, pp. 89–92.
[54] S. Wang, J. Zhang, T. Huang, T. Pan, J. Liu, Y. Liu, FDALB: Flow distribution aware load balancing for datacenter networks, in: IWQoS, 2016, pp. 1–2.
[55] J. Perry, et al., Fastpass: A centralized "zero-queue" datacenter network, in: SIGCOMM, 2014, pp. 307–318.
[56] F. Carpio, A. Engelmann, A. Jukan, DiffFlow: Differentiating short and long flows for load balancing in data center networks, in: 2016 IEEE Global Communications Conference, GLOBECOM, IEEE, 2016, pp. 1–6.
[57] S. Imanpour, M. Kazemiesfeh, A. Montazerolghaem, Multi-level threshold SDN controller dynamic load balancing, in: 2024 8th International Conference on Smart Cities, Internet of Things and Applications (SCIoT), IEEE, 2024, pp. 88–93.
[58] C. Raiciu, C. Pluntke, S. Barre, A. Greenhalgh, D. Wischik, M. Handley, Data center networking with multipath TCP, in: Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks, 2010, pp. 1–6.
[59] The network simulator-NS-2, 2009, https://www.isi.edu/websites/nsnam/ns/. (Accessed September 2024).

[60] A. Varga, The omnet++ discrete event simulation system, in: Proc. of the European Simulation Multiconference, ESM'2001, 2001, pp. 1–7.
[61] B. Pfaff, J. Pettit, T. Koponen, E. Jackson, A. Zhou, J. Rajahalme, J. Gross, A. Wang, J. Stringer, P. Shelar, et al., The design and implementation of open vswitch, in: NSDI, 2015, pp. 117–130.
[62] S. Imanpour, A. Montazerolghaem, S. Afshari, Load balancing of servers in software-defined internet of multimedia things using the long short-term memory prediction algorithm, in: 2024 10th International Conference on Web Research, ICWR, IEEE, 2024, pp. 291–296.
[63] A.H. Alhilali, A. Montazerolghaem, Artificial intelligence based load balancing in SDN: A comprehensive survey, Internet Things 22 (2023) 100814.
[64] EdgeCore wedge 100BF-32X series switches, 2024, [Online]. Available: https://www.epsglobal.com/products/switches/data-center-switches/100g-data-center-switches/dcs800-32x-100g-qsfp28-data-center-switch. (Accessed March 2024).
[65] H. Zhang, J. Zhang, W. Bai, K. Chen, Mosharaf, Resilient datacenter load balancing in the wild, in: SIGCOMM, 2017, pp. 253–266.
[66] LetFlow on github, 2024, https://github.com/BurstBalancer. (Accessed March 2024).
[67] P. Gill, N. Jain, N. Nagappan, Understanding network failures in data centers: measurement, analysis, and implications, in: SIGCOMM, 2011, pp. 350–361.
[68] B. Montazeri, et al., Homa: A receiver-driven low-latency transport protocol using network priorities, in: SIGCOMM, 2018, pp. 221–235.

Ying Wan received the B.S. degree in communication engineering from Northwestern Polytechnical University in 2016 and the Ph.D. degree in computer science and technology from Tsinghua University, China, in 2022. He is currently a Senior Network Technology researcher with China Mobile (Suzhou) Software Technology. His research interests include bloom filter design, high performance network algorithm, software defined networking and multipath load balancing in datacenter networks.

Haoyu Song received the BE degree in electronics engineering from Tsinghua University, in 1997, and the MS and DSc degrees in computer engineering from Washington University in St. Louis in 2003 and 2006, respectively. He is a senior principal network architect with Futurewei Technologies, USA. His research interests include software defined network, network virtualization and cloud computing, high performance networked systems, algorithms for network packet processing and intrusion detection. He is a senior member of the IEEE.

Yu Jia received the BE degree in Sichuan University in 2012. He is a senior principal network architect with China Mobile (Suzhou) Software Technology Co., Ltd, China. His research interests include Future Network Architectures, Software-defined Networks, and Quantum Computing.

Yi Wang is a Research Associate Professor in the Sustech Institute of Future Networks, Southern University of Science and Technology. He received the PhD degree in Computer Science and Technology from Tsinghua University in July 2013. His research interests include Future Network Architectures, Information Centric Networking, Software-defined Networks, and the design and implementation of high-performance network devices.

Ling Qian received the BE, MS and DSc degree in computer science and technology from Tsinghua University, in 1995, 1997 and 2001, respectively. He is a chief scientist with China Mobile (Suzhou) Software Technology Co., Ltd, China. His research interests include Software Engineering, Cloud Computing, Big Data, and Quantum Computing.

Tian Pan received the Ph.D. degree from the Department of Computer Science and Technology, Tsinghua University, in 2015. He is currently an associate professor with Beijing University of Posts and Telecommunications. His primary research interests include cloud data center networks, programmable data plane, and satellite networks.