QALL: Distributed Queue-Behavior-Aware Load Balancing Using Programmable Data Planes

Wai-Xi Liu, *Member, IEEE*, Jun Cai, Sen Ling, Jian-Yu Zhang, and Qingchun Chen, *Senior Member, IEEE*

***Abstract*—Existing load-balancing methods used in data center networks involve some shortcomings such as excessively large decision delays during reactions to microbursts and large overheads involved in active probing. Programmable data planes have provided new opportunities for local decision-making on switches to address these issues. We observe that queue behavior (i.e., queue occupancy, queuing trend, and dequeue time interval) in switches can reflect the current or future congestion degree on a network. Furthermore, following data-driven experiments, we found an accurate fitting function of congestion degree to queue behavior. Thus, we propose an in-network load-balancing scheme based on a programmable switch, called queue-behavior-aware localized load balancing (QALL). In QALL, each switch independently selects egress ports probabilistically according to fine-grained-measured local queue behavior. The key concept of QALL is to take account the evolutionary process of reaching the current queue state into its decision basis for load balancing. Experimental results under actual DCN workloads (including Web search and data mining workloads) demonstrate the effectiveness of QALL. In terms of flow completion time, decision delay, network shock, load sharing accuracy, and packet reordering, QALL outperformed recent per-packet (DRILL), per-flowlet (LetFlow and CONGA), and per-flow (ECMP) load balancers, particularly under heavy load. For example, under asymmetrical topology with 90% load level, the flow completion time of QALL was lower than that of ECMP, LetFlow, CONGA, and DRILL by up to 54.7%, 46.5%, 38.9%, and 18.9%, respectively.**

***Index Terms*—Data center networks, distributed, load balancing, programmable data plane, queue behavior.**

I. Introduction

DATA center networks (DCNs) provide infrastructure for many online services, such as machine learning, on-demand video delivery, Web search, cloud computing, and interactive online tools [16].

Specifically, the DCN topology plays a significant role in determining the level of failure resiliency, ease of incremental expansion, communication bandwidth and latency. Based on a CLOS architecture [16], existing DCN topologies often involve a large degree of path redundancy, which allows for increased fault tolerance. Properly distributing traffic loads across these paths reduces contention among flows while increasing overall resource utilization. Effective load balancing aims to avoid situations in which many links may fall idle while others continue to experience congestion.

Although most DCN topologies are symmetrical, in practice, DCNs turn out to be often asymmetrical because of frequent failures of network elements (e.g., switches, links, and ports); for example, up to 40 link failures per day [15], [16]. However, the performance of some load-balancing schemes depending on symmetrical characteristic of topology deteriorates significantly under asymmetrical topologies (e.g., equal-cost multi-path (ECMP) [3] and Presto [8]).

Static load-balancing approaches such as ECMP [3] are not suitable in DCNs because of the highly dynamic and bursty nature of typical traffic. Alternative adaptive load-balancing approaches can dynamically select paths for traffic loads to minimize hotspots. Thus, the decision delay of adaptive load-balancing approaches becomes critical owing to the frequent decision-making required. However, decision delays in load-balancing methods based on controllers (e.g., Hedera [6], DeepRLB [28], Shafiee and Ghaderi [19], and Oddlab [33]) or end hosts (e.g., HPCC [45], CLOVE [7], Presto [8], Zhang et al. [20], and NDP [21]) are generally quite large. The basic concept behind these methods is to collect and react to global or nearly global congestion information. However, they typically have control loops that are several orders of magnitude slower. For example, in terms of controller-based methods, the interaction latency between switches and the controller may be orders of magnitude slower than the speed at which typical datacenter congestion events occur. They also react slowly to microbursts [4]. However, microbursts have been identified as the main culprit of packet loss in DCNs, which leads to retransmissions that impose significant latency and degrade application performance [4], [46].

In summary, these methods move network functions out of the network fabric, striving to delegate load balancing to centralized controllers [6], [19], [28] or end hosts [7], [20], [21]. These entities serve as convenient locations to collect global or end-to-end congestion information.

Manuscript received 9 March 2023; revised 28 September 2023 and 11 December 2023; accepted 19 December 2023. Date of publication 22 December 2023; date of current version 15 April 2024. This work was supported by the National Natural Science Foundation of China (62272113, 61972104), Guangzhou Key Laboratory of Software-Defined Low Latency Network (202102100006), Guangzhou Basic Research Program (2024A03J0398), Key Disciplines of Guangzhou Education Bureau (202255467), Key Laboratory of On-Chip Communication and Sensor Chip of Guangdong Higher Education Institutes (2023KSYS002), China. The associate editor coordinating the review of this article and approving it for publication was P. Papadimitriou. *(Corresponding authors: Wai-Xi Liu; Jun Cai.)*

Wai-Xi Liu is with the Department of Electronic and Communication Engineering, Guangzhou University, Guangzhou 510006, China (e-mail: liuwaixi@sina.com).

Jun Cai is with the School of Cyber Security, Guangdong Polytechnic Normal University, Guangzhou 510665, China (e-mail: caijun@gpnu.edu.cn).

Sen Ling, Jian-Yu Zhang, and Qingchun Chen are with the Department of Electronic and Communication Engineering, Guangzhou University, Guangzhou 510006, China.

Digital Object Identifier 10.1109/TNSM.2023.3345862

From a different direction, some methods (e.g., CONGA [1] and HULA [2]) strive to delegate load balancing to the core of a network, where switches make decisions for load balancing. However, these methods require coordination among multiple switches, leading to a considerable delay in making decisions. For example, although CONGA adds customized hardware mechanisms to leaf and spine switches, its control loop nonetheless typically requires several RTTs, by which time a typical congestion event is likely to have ended [4].

Furthermore, these methods (i.e., controller-based, end-host-based, and multiple-switch coordination) not only increase delays in making decisions but also result in additional overhead. For example, Shafiee and Ghaderi [19] collected link utilization data from switches at a controller. Zhang et al. [20] periodically sends small probe packets between end-host pairs to monitor path conditions. HULA [2] regularly sends probe packets transmitted between switches to sense the global link utilization. Thus, designing load-balancing methods with short decision delays suitable for asymmetric topologies and operating in a distributed manner with low overhead is a considerable challenge.

Recently, programmable data planes (PDPs) [23] such as programmable network interface cards (i.e., smartNICs) and programmable switches have attracted increasing attention. In this study, PDPs refers to programmable switches. Due to their programmability, PDPs have provided new opportunities to drive unprecedented innovation in network protocols and architectures. Switches located at the core of the network can directly and accurately observe the network behavior of all processed flows over short timescales. Furthermore, when the switch becomes programmable, it is possible to perform flexible load-balancing strategies directly inside PDPs (i.e., in-network load balancing), as in, for example, HULA [2], DASH [14], and Contra [17]. Clearly, in-network load-balancing schemes are more effective at scale and more responsive to network dynamics. However, they all use active probing to collect network state. Of note, probing adds communication overhead that can lead to performance degradation. Although it does not depend on probing, DRILL [4] can suffer from *network shock* because it allows each programmable switch to select the "best" option among a set of randomly selected multiple possible egress ports for each packet based only on local queue occupancy.

To address dynamic networks, traffic control schemes (e.g., load balancing and routing/flow scheduling) use a strategy to change the path (i.e., they actually also change the switches passed by a given traffic flow in the network) by which traffic is transmitted according to an optimal goal. Clearly, this is eventually reflected in changes in the queue behavior of the switches. In this article, we refer to *queue occupancy*, *dequeue time interval*, and *queuing trend* in the egress ports of switches as the queue behavior of switches. However, collecting fine-grained statistics on queue behavior in real time is challenging. Fortunately, in contrast to traditional switches that infer queue behavior based on back-to-back methods, emerging programmable switches can measure their own queue behavior independently and in a fine-grained manner.

In short, load-balancing strategies actually involve choosing ports to distribute traffic loads. Choosing a port determines the egress traffic of the given egress port, and then the egress traffic largely determines the congestion degree (or load) of a switch to which this egress port connects. We observed some interesting relations between egress queue behavior and egress traffic, including (*i*) egress traffic being positively correlated with egress *queue occupancy* and (*ii*) egress traffic being negatively correlated with egress *dequeue time interval*. Moreover, switches use a port to connect to other switches, and thus the queue behavior of an egress port actually reflects the congestion degree (or load) of the network connected to the port. That is, the queue behavior of switches can reflect the state of the corresponding network. PrintQueue [42] also observed that queuing is both a result of historical effects and the current state of the network.

Therefore, we propose a distributed in-network load-balancing method on programmable data planes, called queue-behavior-aware localized load balancing (QALL). In QALL, each switch probabilistically selects an egress port according to fine-grained-measured local queue behavior to achieve per-packet/per-flowlet load balancing without any coordination among switches or any controllers or probing. The main contributions of this study are summarized as follows.

1) We propose a distributed queue-behavior-aware load-balancing method on PDPs. The key concept is that QALL creatively takes account the evolutionary process of reaching the current queue state into its decision basis for load balancing: how to arrive (i.e., *queuing trend*) and how long to arrive (i.e., *dequeue time interval*) the current queue state (i.e., *queue occupancy*). Furthermore, QALL includes a probabilistic forwarding strategy designed to evenly distribute traffic to each available port, rather than only selecting the best port as in other schemes.
2) We propose a data-driven load-balancing method. Following a data-driven approach, we found an accurate function to fit the congestion degree to queue behavior and applied it to improve the performance of QALL.
3) We used Mininet+BMv2 to test QALL on actual DCN traffic workloads. The experimental results demonstrate that QALL performed better than several existing schemes in terms of lower flow completion time (FCT), shorter decision delay, and smaller load-sharing accuracy. Moreover, QALL does not depend on the symmetrical characteristics of the network topology.

The remainder of this article is organized as follows. In Section II, we review some relevant background and related studies. In Section III, we present some observations and describe the motivation of this work. In Section IV, we present system design of QALL along with a description of the problem it is designed to solve. In Section V, we present a data-driven version of QALL. In Section VI, we provide the experimental results. In Section VII, we discuss some practical issues and suggest several challenging directions for future research. We conclude in Section VIII with a summary of our findings.

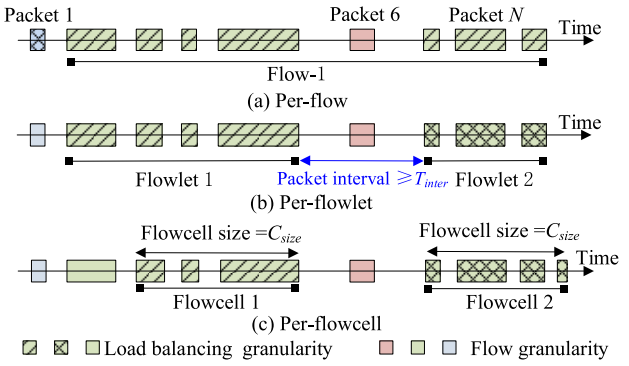

Fig. 1. Load balancing granularity.

II. Background and Related Work

As shown in Fig. 1, load balancing can be performed per-packet, per-flowlet, per-flowcell, and per-flow granularity. A "flow" is a packet stream with the same 5-tuples header. In one flow, a flowlet is a burst of packets that is separated in time from other bursts by a sufficient gap — called the "flowlet timeout" (T_{inter} in Fig. 1); a "flowcell" is a group of packets with a fixed size (C_{size} in Fig. 1). Generally, despite suffering the packet reordering in a flow under network asymmetry, per-packet balancing can obtain high throughput owing to its fine-grained scheduling. Although per-flowlet, per-flowcell, and per-flow load balancing can avoid packet reordering, link utilization cannot be maximized due to the inflexibility and coarseness of these methods. In addition, they are all stateful schemes that must record a flow state (e.g., 5-tuples); that is, some memory is occupied.

Furthermore, decision location of load balancing (in this article, referred as decision-maker) can be at the host (at the end of the network), at a switch (in the core of the network), or at a controller (at the top of the network). However, different decision locations have different capabilities and views of the network.

A. Load Balancing at Switch

1) Load Balancing at Programmable Switch: In terms of per-flowlet load balancing, based on lazy evaluation, CONGA [1] employed a customized leaf switch which has a table to hold the link utilization seen along its outgoing paths. Such link utilization is collected by receiving switches and then piggybacked on traffic. However, its control loop typically requires a few RTTs, and required customized switches. To decrease the decision delay, HULA [2] periodically send probing packets to proactively disseminate link utilization information to all switches in network. However, such probing adds some communication overhead.

In terms of per-packet, DRILL [4] determines the forwarding path of every packet of a flow independently by considering per port local queuing at the switches. In DRILL, each forwarding engine randomly chooses d out of N possible output ports, and finds the one with the current minimum queue occupancy between these d samples and m least loaded samples from previous time slots, and routes its packet to that port. To avoid packet reordering under per-packet granularity, QDAPS [13] selects paths for packets according to the queueing delay of output buffer, and lets the packet arriving earlier be forwarded before the later packets. Moreover, using the "power-of-n-choices" paradigm, QDAPS alleviate the impact of herd behavior under multiple forwarding engines. However, the complexity of QDAPS is a challenge to switches, for example, QDAPS's CPU utilization and memory utilization increase 33% and 64% than ECMP's respectively.

Contra [17] enforces performance-aware routing policies, where a compiler analyzes a desired policy in conjunction with the network topology, and decomposes them into switch-local Programming Protocol-independent Packet Processors (P4) programs. These programs generate probes to collect path metrics, and dynamically choose the best paths along which to forward traffic.

However, one common limitation of these solutions (CONGA, HULA, Contra, DRILL, etc.) on programmable switch is that they only consider use a single "best" path at any given time, and this leads to the "best" path to be quickly congested. The benefits of using multiple paths have been demonstrated by many works on the controller (e.g., HALO [41]). W-ECMP [10], DASH [14], Closer [29], and CLB [27] aim to balance load dynamically across multiple paths in the data plane.

In terms of per-flowlet, as a weighted-cost multipath mechanism (WCMP), W-ECMP [10] uses the path's utilization as the probability of choosing a path other than the best path, thus it is not as sensitive to the frequency of state update compared with CONGA and HULA. However, W-ECMP take a long time to converge to new weights (i.e., large decision delay). For this reason, DASH [14] presents a hash-based data structure that quickly achieves adaptive traffic splitting in programmable data planes to balance traffic across multiple paths. Where, DASH uses the utilization of bottleneck link as its decision basis. Closer [29] leverages in-band network telemetry (INT) to obtain precise link state, and employs WCMP at the network edge to proactively map the flows to the appropriate paths and avoid the excessive congestion of a single link. CLB [27] uses WCMP for traffic-aware load balancing over many paths at a coarse-grained precision.

To adapt to different levels of burst in DCN, IntFlow [30] integrates end-host based per-packet flow state monitoring with flowlet switching in programmable switches. IntFlow's core idea is that proactively rerouting flows experiencing network congestion or failures, while performing cautious flowlet switching for small flows with high sending rate.

Different from other works based on programmable switch, QALL uses probabilistic selection algorithm to distribute load across multiple paths. Besides, QALL's key idea is that selecting egress port according to the *queue occupancy* in conjunction with *dequeue time interval* and *queuing trend* instead of a single network state metric (e.g., queue occupancy or link utilization or FCT).

2) Load Balancing at Traditional Switch: In terms of per-flow, ECMP [3] is widely used in DCN and spreads traffic uniformly across multiple paths. However, because of congestion-oblivious, it is well-known that ECMP performs

poorly when there is asymmetry either in the network topology or the flow sizes [43]. In remote direct memory access (RDMA) supported DCN, Dart [35] isolates the common case of receiver congestion, and further subdivides the remaining in-network congestion into the simpler spatially-localized and the harder spatially-dispersed cases. And then quickly alleviating congestion with the idea of divide-and-specialize. Where Dart uses the local congestion information as its decision basis.

In terms of per-flowlet, LetFlow [5] is a simple congestion-oblivious approach. LetFlow relies on the natural property of flowlets which allows them to shrink or expand (in size) according to available capacity over paths. However, due to the randomness of LetFlow scheduling, the optimal load balancing performance cannot be achieved. In summary, most per-flowlet load-balancing schemes depend on a proper static setting of the flowlet gap, which decides when new flowlets are detected. While a too small gap may result in reordering, a too large gap leads to missed load-balancing opportunities [31]. FlowDyn [31] and Flex [9] can dynamically adapt the flowlet gap. Under a switch-host collaborative paradigm, Flex [9] split the flow into flowlets at the host based on the adaptive timeout., and then tell the flowlet results to switches by marking the adjacent flowlets of the same flow.

In terms of per-packet, some methods are proposed to mitigate packet reordering. For example, (*i*) RMC [12] based on network coding can effectively solve the reordering problem, but it also introduces too many redundant coding packets, which leads to too much extra traffic overhead, long queuing delay and even packet loss. (*ii*)To address this problem, OPER [26] uses opportunistic redundant packets which are replaceable by the data packets in the switches under heavy congestion.

Lots of load balancing schemes, from ECMP [3] to LetFlow [5], to Presto [8], avoid packet reordering under asymmetric topology by balancing coarser units of traffic, but easily lead to under-utilization of multiple paths. AG [18] adaptively adjusts switching granularity according to the asymmetric degree of multiple paths, to alleviate packet reordering.

B. Load Balancing at Controller

Considering the control overhead and decision delay, most controller-based load balancing methods are per-flow granularity. By using a central controller to monitor the network, Hedera [6] detects long flows and reschedules them on a lightly loaded path, but it is not friendly to short flows. Shafiee and Ghaderi [19] dynamically adjusts the weight of the link according to the link utilization, and assigns every arriving traffic to the minimum weight path. However, due to depending on frequently updating link weight, Shafiee's performance is greatly affected by the speed of updating weight and calculating minimum weight path.

DeepRLB [28] and DRL-PLink [36] deploy the deep deterministic policy gradient (DDPG) algorithm on software defined networking (SDN) controller to achieve load balancing. DRL-PLink [36] establishes some corresponding private-links for different types of flows to isolate them such that the competition among different types of flows can decrease accordingly. Where DDPG is used to adaptively and intelligently allocate bandwidth resources for these private-links, by observing FCT.

For SDN-enabled hybrid optical/electrical DCN, DDMP [22] dynamically adjusts traffic distribution according to the inverse ratio of the buffer occupancy. Where the SDN controller guarantees the capacity of the scheduling buffers and reconfiguring the switch fabric.

C. Load Balancing at End-Host

More easily being deployed on end-hosts by a software update, the multipath TCP (MPTCP) leverages multiple sub-flows for data transmission. However, in practice, using multiple sub-flows is efficient only under inter-rack. DCMPTCP [32] aims to improve the efficiency of MPTCP, for example, preventing MPTCP from establishing multiple sub-flows for rack-local traffic; estimating flow size, with which inter-rack flows can leverage multipath in a smarter way. HPCC [45] uses INT to collect queue information in switches to achieve high precision congestion control in DCN.

In terms of per-packet, NDP [21] is a DCN transport protocol which limits the aggregate transmission rate of all incast senders by maintaining a PULL queue at the receiver. In terms of per-flow, to an asymmetric DCN topology, FlowFurl [34] reroutes the flows by combining link failure and congestion information.

In terms of per-packet/per-flow, Zhang et al. [20] monitor path conditions at the end-hosts by sending probe packets between end-host pairs periodically, and reroute flows affected by failures or congestion caused by asymmetries. Specially, in Zhang et al. [20], short flows and long flows use per-flow granularity and per-packet granularity, respectively.

In terms of per-flowlet, Clove [7] employs Paris traceroute [11] to obtain all paths conditions traversing the network. Where, each source leaf node collects the path conditions information to the destination leaf by sending a probe packet, and this information is brought back to the source leaf node by ECN or custom packet headers. Clove relies on ECMP in physical switches.

Besides, few works achieve per-flowcell granularity. For example, without needing sensing congestion, Presto [8] breaks flow into small near-uniform units of data (called flowcell), where, in end-host, flowcells are assigned over multiple paths very evenly by iterating over paths in a round-robin. However, Presto has the difficulty with asymmetric scenarios, and cannot interact well with unbalanced legacy traffic.

III. Some Observations and Motivations

From this overview, we can identify some challenges and make some notable observations regarding existing methods.

A. Challenges

Challenge 1: Which network state serves as a better decision basis. In general, all approaches to load balancing require the state of the network as the basis for their

decision-making, regardless of whether they are controller-based, end-host-based, or switch-based, except for a few methods that do not need to perform sensing. Briefly, network states can be classified into two types: device states (e.g., the queue occupancy of switches, link utilization, and physical bandwidth of links) and traffic states (e.g., flow-level FCT, packet-level delay, jitter, and loss rate). These can provide different types of useful information for decision-making, and their associated cost of measurement differs.

However, most previous studies have used a single metric (e.g., queue occupancy, link utilization, or FCT) as their decision basis. In fact, these metrics reflect only the current state of the network. That is, these metrics do not fully reflect the degree of network congestion. Because queuing is a result of both historical effects and the current state of the network [42], we argue that the evolutionary process by which the current network state was reached (i.e., how the network reached its current state and how long this took) should be considered as the decision basis. However, to the best of our knowledge, no previous studies have considered this evolutionary process.

Challenge 2: How to avoid network shock. Previous load-balancing schemes (e.g., CONGA, HULA, Contra, and DRILL) typically adopt a coarse-grained port selection strategy that tends to directly schedule all traffic along a single "best" path (the port with the smallest queue occupancy or the path with the lowest link utilization) at any given time. Such strategies may easily lead to the "best" path quickly becoming excessively congested, and may also result in frequent rerouting, which leads to the network state to fluctuate widely and change over frequently. For example, under per-flowlet granularity, link utilizations vary over time and from one another by up to 2× [43]. In addition, frequent rerouting within a flow can mix ACKs belonging to different paths in congestion control protocols, which adversely affects the flow rate control [30]. This study refers to the frequent rerouting of traffic as *network shock*, which can be evaluated by the *Variance* of the queue occupancy.

Challenge 3: How to decrease the decision delay to meet high bandwidth and microbursts requirements. In DCNs, to meet line-speed forwarding and ultra-low end-to-end latency requirements (∼10's of μs), the processing time within a switch is required to be smaller and smaller when bandwidth become higher and higher (>10 Gbps). Moreover, microbursts (short-lived traffic spikes that last for less than a millisecond) quickly cause queues of switches to become fully utilized, leading to immediate packet loss and subsequent periods of unexpectedly high packet delay [24]. Measuring and managing microbursts is challenging because of their short lifespans, frequent occurrence at irregular intervals, and diverse and ever-changing root causes (e.g., applications and TCP artifacts such as ACK compression) [46]. For example, at Facebook, more than 70% of microbursts last for less than a few tens of microseconds, which is significantly shorter than the frequency of most deployed measurement frameworks [47]. The two main methods in current use to manage microbursts include absorbing the microbursts by adding sufficient buffer space at switches [48] and load balancing [4]. The former may incur high costs and fail under load and at scale. However, most existing load-balancing methods [1], [2], [6], [7], [8], [19], [20], [21] that are performed on large timescales react slowly to microbursts. A few methods have aimed to achieve microburst tolerance on switches at short timescales, such as DRILL [4] and Vertigo [46]. DRILL performs micro load balancing to distribute a load as evenly as possible on a microsecond timescale. Thus, we argue that switches (the network core) should take corrective action in response to microbursts in situ and in real-time before a situation worsens.

Challenge 4: How to address constraints of programmable switches. To achieve packet processing with a high line-speed, programmable switches have many constraints on the algorithms that can be implemented [23]. Some of these constraints are highlighted below to clarify the design challenges and decisions involved in QALL.

(*i*) Programmable switches can perform only limited operations (e.g., missing division and floating-point arithmetic operations [44]) and programming models (e.g., missing loops). In this study, we used shift and addition/subtraction operations to replace division equivalently. We also use a random function to avoid floating-point operations (more details are presented in Sections IV-C and IV-D, respectively).

(*ii*) Programmable switches provide relatively limited computational and memory resources to support application-specific tasks. For example, in a typical programmable switch (e.g., with an Intel Tofino), each stage can access only ∼10 MB of stateful memory (e.g., registers) [24]. To save the memory, we use a hash operation to replace storing data.

(*iii*) In most commodity programmable switches, queue behaviors (i.e., the decision basis in QALL) are generally available in the egress pipeline. However, the load-balancing decision location must be within the ingress pipeline. The decision basis must be transmitted to the load-balancing decision location using a P4 clone operation [25]. Thus, a space-time mismatch obtains between the load-balancing decision location and its decision basis. That is, the network state reflected by the decision basis is slightly out of date compared to the time at which decisions are made. This naturally reduces the accuracy of decision-making processes. To address this space-time mismatch, we apply an updating period factor (T_b) to adjust the freshness of the decision basis as discussed in Section IV-E.

B. Observations

In essence, by constantly adjusting the transmission path for traffic, the load-balancing scheme aims to achieve a reasonable space-time distribution of traffic in the network. Because switches are forwarding nodes on the transmission path, variations in the traffic distribution in a network eventually lead to changes in the queue behavior of the switches. Thus, we aim to find the relationship between egress traffic and queue behavior.

As shown in Fig. 2, *queue occupancy* (denoted by L) is defined as the proportion of the queue depth of the egress port to its total queue length when packets enter the queue of the egress port. The *dequeue time interval* (denoted by T) is

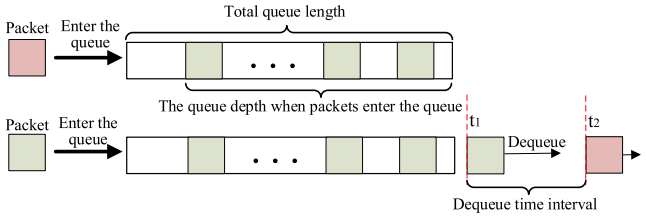

Fig. 2. The queue occupancy and dequeue time interval.

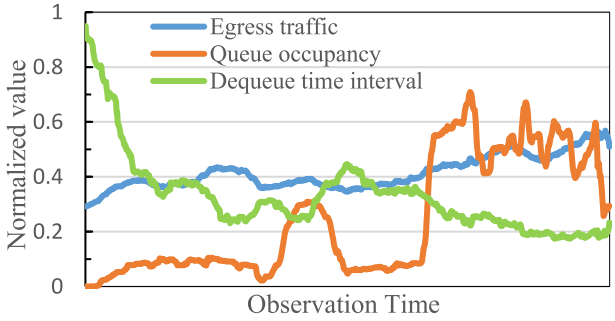

Fig. 3. The relationship between the egress traffic, queue occupancy and dequeue time interval of ports.

defined as the time difference between two packets leaving the queue of the egress port, and the *queuing trend* is defined as whether the current *queue occupancy* is formed by an increase from small to large or by a reduction from large to small. For example, when the current queue occupancy is 50%, it may be reduced from 60% to 50% or increased from 40% to 50%.

First, we build a traffic *observation dataset* under actual DCN workloads. Where we continually observe the running process of the DCN (shown in Fig. 7) with typical setups. Specially, ECMP is adopted as the load-balancing scheme, and two types of widely used workloads (i.e., Data Ming and Web Search) whose traffic distributions are shown in Fig. 8 are loaded into servers *h*1-*h*32 as the background traffic. Half of these hosts were configured as senders, and the other half receivers. Because almost all traffic within a network passes through spine switches, collecting traffic data at spine switches suffices to determine the running process of the network. We collected 200,000 groups of data on egress traffic (kbps), *queue occupancy* (%), and *dequeue time interval* (*us*) of egress ports of *S*1 and *S*2, as shown in Fig. 7. Second, data preprocessing was performed on the *observation dataset* to obtain normalized values such as maximum and minimum normalization and data cleansing. Fig. 3 shows their envelopes, and we note the following observations.

***Observation* 1**: The egress traffic of a given port was positively correlated with egress *queue occupancy*. The *Pearson correlation coefficient* reached up to 0.7824.

***Observation* 2**: The egress traffic of a given port was negatively correlated with egress *dequeue time interval*. The *Pearson correlation coefficient* reached up to −0.7308.

Clearly, *Observations* 1–2[1] demonstrate some inherent facts obtained in practice. For *Observation* 1, the *queue occupancy* of an egress port indicates the current queue state of the port when packets arrive, and the queue depth reflects the current status of the egress traffic of the port (i.e., the longer the queue depth of a port, the greater its egress traffic). Therefore, the smaller the *queue occupancy*, the lower the egress traffic. This observation is also consistent with the conclusions of queuing theory [49]. It should be noted that this observation is true only when there is queueing in the network. When the load of the entire network is extremely light such that all egress ports of the entire network have no queueing simultaneously (the *queue occupancy* of all egress ports is zero), this observation is not necessarily true. Of course, under this case, there is no need for load balancing scheme.

By the same token, for *Observation* 2, the egress *dequeue time interval* indicates the time required to form the current queue state. When the *queue occupancy* is certain, a longer egress *dequeue time interval* actually implies that packets are allocated by a longer time interval to the link corresponding with this egress port; in other words, the link is lighter. Therefore, in most cases, the longer the egress *dequeue time interval*, the less the egress traffic. For example, during an observing time (referred to as *R* seconds), if *M* packets are dispatched to the corresponding link of an egress port, where the size of all packets is *S* bits and their transmission delay is t seconds, thus, the egress traffic of this egress port is expressed as follows,

$$E\_traffic = \frac{M \times S}{R} = \frac{M \times S}{\sum_{j=1}^{M-1} T_j + t}(bit/s) \qquad (1)$$

where T_j is the *dequeue time interval* between the j^{th} and $(j+1)^{\text{th}}$ packet. Obviously, from equation (1), when T_j is larger, the egress traffic is naturally less. When we let the observation time *R* be sufficiently short (e.g., there is only one packet during *R*), the egress traffic approaches $\frac{S}{T_j}$ (bit/s). Thus, the negative correlation between egress traffic and egress *dequeue time interval* is expected. Considering the observed correlation between egress traffic and *dequeue time interval* is a moderate instead of extremely high level (the Pearson correlation coefficient reached up to −0.7308), we cannot completely conclude that a longer *dequeue time interval* indicates a lighter load. We discuss some special cases further in Section VII-C.

C. Motivations

Inspired by these observations, we followed several motivations to address the abovementioned challenges.

Motivation 1: Decision basis integrating with the current and future network states. In short, the load-balancing strategy is actually the choice of port, which determines the egress traffic of a given port (i.e., the corresponding link). Furthermore, the egress traffic of a port largely determines the congestion degree (or load) of a switch connected to the port. Clearly, the greater the egress traffic, the higher the congestion degree. Thus, *Observations* 1 and 2 show that the *queue occupancy* and egress *dequeue time interval* of a port can reflect the current congestion degree of the network connected to this port.

[1]In the *Observations* 1–2, the *egress traffic*, egress *queue occupancy*, and egress *dequeue time interval* refer to the state of the same egress port.

Furthermore, the current *dequeue time interval* actually reflects the number of historical packets injected into the corresponding link in the past; thus, it can also reflect the future congestion degree (more accurately, the congestion degree in this study mainly refers to the link load) of a given link. Therefore, when we take *dequeue time interval* as a decision basis of load balancing, it is actually based on an implicit congestion prediction.

Thus, when we consider the *queue occupancy* and *dequeue time interval* together as the decision basis, we actually consider the current and future congestion degree as a whole, and actually achieve the cooperation between switches either without depending on controllers or without explicit information transmission between switches.

Simultaneously, the *queuing trend* should be a part of the decision basis. If the current queue occupancy results from a growth, this indicates that the queue is increasing and the network congestion may be expected to become more severe; if the current queue occupancy results from a reduction, this indicates that the queue is decreasing and the network congestion may be expected to gradually ease. The load-balancing strategies applied in each case should differ.

In summary, compared with link utilization and FCT, *queue occupancy*, *dequeue time interval*, and *queuing trend* can more directly reflect current and future network congestion degrees from the bottom; moreover, the cost of measuring them in PDP is less.

Therefore, to address *Challenge* 1, based on the inherent relationship between the network's congestion degree and switches' queue behavior, our proposed approach uses queue behavior as the decision basis for load balancing. To summarize, our decision basis for load balancing creatively considers the evolutionary process that occurred to reach the current queue state, including how to arrive (i.e., *queuing trend*) and how long it took to arrive (i.e., *dequeue time interval*) at the current queue state (i.e., *queue occupancy*).

Motivation 2: Distributing traffic evenly instead of through the best path. In order to avoid network shock in *Challenge* 2 to achieve better load balancing, we designed a probabilistic forwarding strategy that distributes traffic evenly to each available port instead of selecting only the best port.

Motivation 3: The first observer is the first decision-maker. To decrease decision delays, decision-makers should be placed close to the network state that needs to be accessed, which may be described as *data locality*. The programmable switch located at the core of a network can directly and accurately observe the network behaviors of all flows that pass through the switch over short timescales. This programmability can support flexible load balancing strategies directly inside PDPs. Thus, to address *Challenge* 3, we employ programmable switches as the key decision-maker.

IV. System Design of QALL

A. Problem Statement

Considering a DCN with V switches and N hosts, we model the network as a directed graph $G = (V, E)$. Any two hosts among N hosts are referred toas an end-to-end host pair. In DCNs, communication for an end-to-end host pair contains a set of candidate paths where the K-shortest paths (KSP) [37] algorithm is used to calculate the candidate paths. By default, all queues in switches use first-in-first-out (FIFO). Therefore, its length increases when a new packet is inserted into a queue and decreases when dequeuing packets.

Programmable switches adopt a packet-triggered work scheme; that is, the execution of a P4 program for the load-balancing algorithm is triggered by a packet arrival event instead of by a strict period (periodically). This scheme involves some constraints on the P4 programming of the load-balancing algorithms.

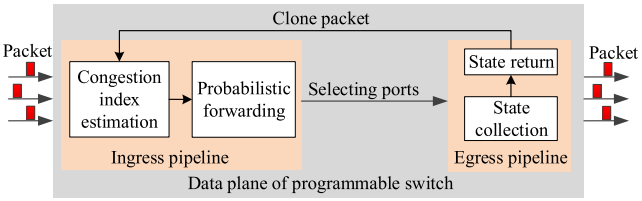

Fig. 4. Framework of QALL.

B. Workflow of QALL

Inspired by above-mentioned motivations, based on its queue behavior fine measured, the data plane of programmable switches probabilistically selects egress ports for traffic to achieve per-packet and per-flowlet load balancing. To summarize, the key idea of QALL is that the more idle a given port is (i.e., the lower the *queue occupancy* of the port and the greater its *dequeue time interval*), the greater the probability of its being selected, so the traffic is preferentially transferred to the idle port.

As shown in Fig. 4, QALL includes state collection, state return, congestion index estimation, and probabilistic forwarding modules. The first two modules (responsible for accessing the load-balancing decision basis (i.e., queue behaviors)) are implemented on the egress pipeline of the PDP, whereas the latter two modules (the load-balancing decision location) are deployed on the ingress pipeline of the PDP. The state collection and state return modules "periodically"[2] send the ingress pipeline queue behavior data of each egress port, which are used to compute the congestion index using the congestion index estimation module, and the probabilistic forwarding module uses this index to select an egress port.

QALL can achieve per-packet and per-flowlet load balancing. As shown in *Algorithm* 1, when packet j arrives at ingress port i, the following processing is triggered:

1) The ingress pipeline determines whether the packet j is a clone or a normal packet. The clone packet is generated by the P4 clone operation and implemented through *recirculation* feature that sends a packet (i.e., clone packet) back to the ingress pipeline from the egress pipeline.

[2]Under the packet-triggered work scheme, strict periodicity is impossible.

Algorithm 1 QALL

Function QALL(packet, register): A packet arrives a ingress port // Packet-triggered work scheme
/*In ingress pipeline processing*/

1: **If** clone_packet **then**
2: *Queue_register.**Write**(Egress_port)* // Updating the queue behavior data of corresponding egress port
3: ***Drop**(clone_packet)* // Discarding the clone packet.
4: **else**
5: Finding all available egress ports [*P*] according to the ingress port
6: ***Congestion_module***([*P*]): // Congestion index estimation module
7: *Queue_info= Queue_register.**Read***([*P*]) // Obtaining the queue behavior data of [*P*]
8: *C*[*P*]=***Compute***(*Queue_info*)//Computing congestion index *C*[*P*] for [*P*].
9: ***Return***
/* Per-packet granularity */
10: *Egress_port*= ***Probability***(*C*[*P*]) //Probabilistic forwarding module, and ***Algorithm*** 2 shows more details.
/* Per-flowlet granularity */
11: *Flowlet_index*=***Hash***(packet's five-tuples) //Generating a flowlet index
12: T_1=*standard_metadata.ingress_timestamp*//Getting the entry timestamp of current packet
13: T_2=*Timestamp_flowlet_register.**Read***(*Flowlet_index*) //Getting the entry timestamp of previous packet
14: $\Delta T_1 = T_1 - T_2$ //Computing the time interval between two packets ΔT
15: **If** $\Delta T_1 > T_{inter}$//Determining whether the time interval is larger than T_{inter} **then**
16: Creating a new flowlet
17: *Egress_port*= ***Probability***(*C*[*P*])//***Algorithm*** 2 shows *Probability*()
18: *Egress_register.**Write***(*Flowlet_index*, *Egress_port*) // Updating the forwarding port of current flowlet.
19: ***Else*** //The packet belongs to an existing flowlet
20: *Egress_port= Egress_register.**Read**(Flowlet_index)* // Obtaining the forwarding port of current flowlet
21: ***Return***
/* In egress pipeline processing */
22: *collect_feedback(clone_packet, Queue_info)* //State collection and return module: ***Algorithm*** 3 shows their more details.

Algorithm 2 Probabilistic Forwarding Module in QALL

Function *Probability (congestion index for ports set [P])*

1: *C*[*P*]= *congestion index for ports set [P]*
2: *W*[*P*]=***Weight***(*C*[*P*]) //Converting the congestion index into corresponding weight as shown equation (6).
3: *W_total*=***Sum***(*W*[*P*])
4: *Grid_ID* =***Random***(*W_total*)
5: Using *Grid_ID* to determine egress port: packet is forwarded from the egress port to which the *Grid_ID* belongs.
6: ***Return***

Algorithm 3 State Collection and Return Module in QALL

Function *collect_feedback(packet,register):* A packet arrives egress port *Egress_port* // Packet-triggered work scheme

1: T_1=*standard_metadata.egress_timestamp* //Obtaining the arriving time of the packet.
2: T_2=*Timestamp_port_register.**Read**(Egress_port)* //Obtaining the previous cloning time corresponding with *Egress_port*.
3: $\Delta T_2 = T_1 - T_2$ //Computing the time interval.
4: **if** $\Delta T_2 \leq T_b$ // The updating time has not come.
5: Continue collecting *Egress_port*'s queue behavior (*Queue_info*), which is saved in switch's registers.
6: ***Else*** //The updating time has come
7: ***Clone*** (*Queue_info*)
8: *Timestamp_port_register.**Write**(Egress_port)* //The cloning time corresponding with *Egress_port* is recorded

2) For clone packets, after the queue behavior data of the corresponding egress port is updated, packet *j* is discarded (Lines 1–3).
3) For normal packets, the load-balancing algorithm (per-packet or per-flowlet) is executed.

The algorithm finds all the available egress ports corresponding to the established candidate paths (denoted by [*P*]) according to the ingress port (Line 5). The congestion index estimation module computes the congestion index (denoted by *C*[*P*]) of each available egress port (Lines 6–9).

In the case of per-packet processing, the probabilistic forwarding module uses the index *C*[*P*] to select an egress port for packet *j* (Lines 10, more details are in *Algorithm* 2).

In the case of per-flowlet processing, whether packet *j* should be divided into a new flowlet or assigned to an existing flowlet is determined according to the time interval ΔT_1 between two packets (Lines 11–14). If the time interval is larger than the flowlet threshold T_{inter} (T_{inter} can be set to a value of the end-to-end delay level; in this study, $T_{inter} = 10\,\text{ms}$), we create a new flowlet and select an egress port for packet *j* (Lines 15–18); otherwise, we follow an existing flowlet to forward packet *j* (Line 20).

Regardless of whether per-packet or per-flowlet granularity is applied, the state collection and return module in the egress pipeline sends the ingress pipeline the queue behavior data of each egress port (Line 22; more details can be found in *Algorithm* 3).

C. Congestion Index Estimation Module

The congestion degree estimated by the congestion index estimation module is the decision basis for forwarding packets. Hence, the accuracy of this module directly affects the performance of the load-balancing strategy. Our proposed

approach uses the congestion index to evaluate the congestion degree of a network, where the higher the congestion index, the more severe the congestion degree. Thus, following *Motivation* 1 and *Observations* 1–2, which showed a positively/negatively correlated relationship between egress traffic, *queue occupancy,* and *dequeue time interval*, the equation for estimating the congestion index can be expressed as

$$C_i = L_i / T_i \times V_i, \tag{2}$$

where C_i is the congestion index of egress port *i*, L_i ($0 \leq L_i \leq 1$) is the *queue occupancy* of egress port *i*, and T_i is *dequeue time interval* of egress port *i* (it is an integer in *us* in programmable switch). Evidently, Equation (2) is applicable to any topology or traffic pattern. That is, QALL performs well not only for symmetrical topologies, but also for asymmetrical topologies.

L/T reflects the speed at which the queue depth changes. Thus, a port with a lower *L/T* should be selected preferentially. V_i is the *queuing trend* of egress port *i.* Given that the congestion degree decreases when the *queuing trend* decreases, ports with decreasing *queuing trend* should be selected preferentially. Thus, considering Equations (5) and (6), we should use the following rule to set the value of V_i (i.e., the weight of the congestion index): the value of V_i in cases of the *queue occupancy* increasing is larger than that in case of the *queue occupancy* decreasing. For example, in this study, $V_i = 2$ when *queue occupancy* is increasing, and $V_i = 1$ when the *queue occupancy* is decreasing. In other words, the weight in cases of the increasing is double that in case of the decreasing.

Considering that the programmable switch does not support division operations, and to maintain the negative correlation between *C* and *T*, by introducing the normalization factor τ, we simplify equation (2) to perform multiplication, shift, and addition/subtraction operations as follows.

$$C_i = L_i \times 1 - T_i/\tau \times V_i, \tag{3}$$

where τ is the time constant that is used to normalize *dequeue time interval T* to [0,1]. We should apply the following rule to set the value of τ : τ should be set a value of end-to-end delay level to ensure that it is greater than *T*. As an example, τ is ∼10 ms in this study.

Furthermore, considering that programmable switches do not support floating-point operations, we simplify the normalization in Equation (3) to a subtraction operation such that the operation results are guaranteed to be integers. Thus, the final equation for estimating congestion index can be expressed as

$$C_i = L_i \times (\tau - T_i) \times V_i. \tag{4}$$

Compared with other methods that depend only on the queue occupancy to estimate the congestion degree, Equation (4) can reflect the network congestion degree more accurately and perform load balancing more effectively. More importantly, the congestion index can be calculated directly from the local queue behavior. Specifically, we let C_{max} be the maximum value of C_i, which indicates the most severe congestion degree. Therefore, C_{max} is the value of C_i when $L_i = 1$, $T_i = 0$, and $V_i = 2$.

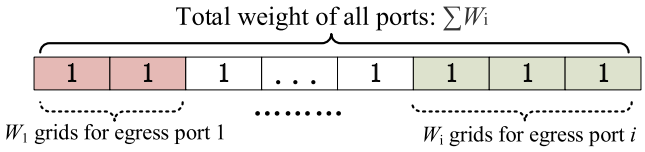

Fig. 5. Probabilistic selecting the egress port in a selecting grid manner.

D. Probabilistic Forwarding Module

The probabilistic forwarding module determines the probability of port selection according to the congestion index computed using Equation (4) to evenly schedule traffic to each available port. In this study, the probability of egress port *i* being selected for forward packets (denoted as P_i) is computed as follows,

$$P_i = W_i / \left(\sum W_i \right), \tag{5}$$

where the traffic forwarding weight of egress port *i* is referred to as W_i, which decreases with an increase in congestion index C_i. Furthermore, W_i can be determined by

$$W_i = C_{\max} - C_i, \tag{6}$$

Equation (5) depends on the division operation and P_i may be a floating-point value. However, programmable switches do not support floating point operations. Thus, our proposed approach uses a random function provided in the programmable switch to realize a uniform distribution, to achieve the probabilistic forwarding described in Equation (5). Specifically, we select the forwarded egress port with a selection grid manner. As shown in Fig. 5 and *Algorithm* 2, the traffic forwarding weight of egress port *i* is represented by W_i grids (Line 2), and a switch has ΣW_i grids, which is identified by 1, 2, 3…ΣW_i (Line 3). The random function randomly generates a grid identifier (*Grid_ID*) among [1, ΣW_i] (Line 4), and the packet is forwarded from the egress port (*Egress_port*) to which the grid identifier belongs (Line 5).

E. State Collection and State Return Module

The state collection and state return modules are responsible for the "regular" collection of the queue behavior of the egress port and send the data to the ingress pipeline. In fact, the frequency with which these behavior data are collected and sent determines the freshness of the decision basis for load balancing. However, excessively frequent state collection and sending result in additional overhead on the switch. Thus, we set an adjustable updating period factor (T_b) to achieve a tradeoff between performance and overhead. In this study, $T_b = 1$ ms. Further details regarding the overhead are provided in Section VI-F.

As shown in *Algorithm* 3, following the decision of the probabilistic forwarding module, a packet is scheduled to an egress port (*Egress_port*), and this event triggers the system to compute the time interval between this event and the last update event (ΔT_2, Lines 1–3). If ΔT_2 is less than T_b, the state collection module continues to collect the *Egress_port*'s

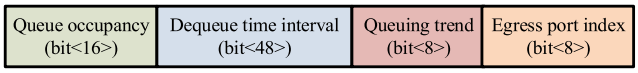

Fig. 6. The format of clone packet in QALL.

queue behavior and saves the data in corresponding registers of the switch (Lines 4–5).

If ΔT_2 exceeds T_b, the state return module reads *Egress_port*'s queue behavior data stored in the register and sends a clone packet that piggybacks with this data to the ingress pipeline of the switch (Lines 6–7). Finally, the cloning time is recorded by a register (Line 8). When this clone packet arrives at the ingress pipeline, update and discard operations are performed according to lines 1–3 in *Algorithm* 1. As shown in Fig. 6, the clone packet comprises 10 bytes, where *Egress port index* refers to the egress port to which the piggybacked queue behavior data corresponds.

V. Data-Driven QALL

The proposed load-balancing strategy allocates traffic by estimating the congestion degree of each egress port. Therefore, the accuracy of estimating the congestion index is crucial to determine the effectiveness of the load-balancing strategy. We designed QALL based on a qualitative analysis of the correlation between egress traffic, *queue occupancy*, and *dequeue time interval*. Subsequent experimental results confirmed that QALL performed well. Furthermore, based on the data-driven concept, this section aims to quantitatively fit the relationship between these variables through a regression analysis, so as to estimate the congestion index more accurately and further improve QALL's performance.

A. Data-Driven Congestion Index Estimation

To achieve above-mentioned fit, we used 150000 groups and 50000 groups of the *observation dataset* as a training and testing sets, respectively.

1) Data-Driven Methods: Many data-driven methods, such as neural networks [38] and least-squares methods [39], can be used to describe the relationships between variables. However, the outputs of neural network models trained using large volumes of data are generally not interpretable. The greatest advantage of the least- squares method is that we can obtain an explicit fitting function that explains the relationships between these variables. Based on this explicit function, a programmable switch can be used to design a congestion index estimation module and realize a load-balancing strategy. Therefore, we chose least-squares polynomial linear regression and least-squares polynomial nonlinear regression to fit the relationship between egress traffic, *queue occupancy*, and *dequeue time interval* in this study.

The basic principle of least-squares for fitting data is to use a polynomial function to approximate discrete sequences (X, Y). We assume that Y_i is the i^{th} sample of the fitted object and X_{ki} is the k^{th} feature of the i^{th} sample in X. This approach is used to obtain the polynomial function $f(X_i) = a_0 + a_1X_{1i} + a_2X_{2i} + \cdots + a_kX_{\mathrm{ki}}$ such that the sum of the squares of the differences between $f(X_i)$ and Y_i is minimized. That is, Equation (7) is minimized.

$$E = \sum_{i=1}^{n}(f(X_i) - Y_i)^2 = \sum_{i=1}^{n}(a_0 + a_1X_{1i} + a_2X_{2i} + \cdots + a_kX_{ki} - Y_i)^2. \quad (7)$$

In terms of fitting high-order polynomial regression, this approach is a non-linear regression model. In our proposed method, the higher-order independent variable in the polynomial is converted into a separate feature. For example, for feature X_1, if the sample data are [0, 1, 3], then for the quadratic and cubic terms of X_1, it is regarded as an independent feature, that is, the sample [0, 1, 9] and [0, 1, 27], respectively. In this way, these three features are input as independent features so that the nonlinear polynomial regression model is converted into a multivariate linear regression model, and the fitting equation is obtained by solving.

In this study, the egress traffic is taken as *Y*, the *queue occupancy L* and *dequeue time interval T* are taken as *X*, and then the explicit function *Y*=*f*(*X*) is fitted as the estimated congestion index. The congestion index can be calculated directly from the local queue behavior.

2) Fitting Results: Considering that the *queuing trend* is an attribute contained in the *queue occupancy* itself, to reduce the complexity of the fitting process, we did not add a *queuing trend* to fit the congestion index. Therefore, using the training set of the *observation dataset*, L and T were fitted as egress traffic based on least-squares polynomial linear regression, polynomial quadratic regression, and polynomial cubic regression which are referred to as *C-linear*, *C-poly*2, and *C-poly*3, respectively.

The fitting equation for bivariate polynomial linear regression is shown in Equation (8) below.

$$C\text{-}linear = 0.167 + 0.261 \times L - 0.468 \times T. \quad (8)$$

The fitting equation for the bivariate quadratic polynomial regression is shown in Equation (9).

$$C\text{-}poly2 - 0.175 + 0.797 \times L - 0.896 \times T - 1.068 \times L^2 + 1.422 \times L \times T + 0.845 \times T^2. \quad (9)$$

The fitting equation for the bivariate cubic polynomial regression is shown in equation (10).

$$C\text{-}poly3 = 0.186 + 0.892 \times L - 1.446 \times T - 2.514L^2 + 18.873 \times L \times T + 3.472 \times T^2 + 1.654 \times L^3 - 22.944 \times L^2 \times T - 58.49 \times L \times T^2 - 2.254 \times T^3. \quad (10)$$

It may be observed that equation (8)–(10) based on data-driven approaches and equation (4) based on theoretical derivation all show the correlation between the egress traffic, *queue occupancy*, and *dequeue time interval*. In particular, Equations (8)–(10) further quantitatively determine the weight of each variable in the equation to estimate the congestion index; thus, they can estimate the degree of congestion more accurately.

TABLE I
FITTING ACCURACY

Power of fitting function	R^2	*MRE*
Primary power	0.5772	0.3687
Quadratic	0.6277	0.2755
Cubic	0.6738	0.2739
Fifth power	0.7312	0.2696
Octave	0.7358	0.2638
Tenth power	0.7452	0.2114

Simultaneously, we observed how the power of the fitting function affected the fitting accuracy. Table I shows the *Goodness of Fit* (R^2) and Mean Relative Error (MRE) for the test set of observation dataset, which are widely used to evaluate fitting.

From Table I, it may be observed that R^2 and *MRE* improved to different degrees when the power of the fitting function was greater, indicating an improved fitting accuracy. However, the fitting accuracy did not improve significantly after cubic fitting, and the required computing resources increased exponentially with increasing power. Considering the limited computing resources of programmable switches, if a large amount of computing resources are consumed to compute the congestion index, the performance of forwarding normal packets decreases accordingly.

Therefore, we used primary power, quadratic, and cubic fitting to estimate the congestion index. Considering that programmable switches do not support floating-point operations, we multiplied the right side of Equations (8)–(10) by a scaling factor (100) and then truncated and rounded the data, converting the floating-point values to integers. Thus, the final equation used to estimate the congestion index is as follows.

$$C\text{-}linear = 16 + 26 \times L - 46 \times T, \tag{11}$$

$$C\text{-}poly2 = 17 + 79 \times L - 89 \times T - 106 \times L \times L + 142 \times L \times T + 84 \times T \times T, \tag{12}$$

$$C\text{-}poly3 = 18 + 89 \times L - 144 \times T - 251 \times L \times L + 188 \times L \times T + 347 \times T \times T, + 165 \times L \times L \times L - 2294 \times L \times L \times T - 5849 \times L \times T \times T - 225 \times T \times T \times T. \tag{13}$$

B. Data-Driven Load Balancing Scheme

To further improve the performance of QALL, Equation (4) can be replaced by Equations (11), (12), and (13) respectively, while the state collection, state return, and probabilistic forwarding modules remain unchanged. As an example, we aimed to optimize the per-packet granularity QALL-Pkt. These schemes that use *C-linear*, *C-poly*2, and *C-poly*3 for QALL-Pkt are referred to as QALL-linear, QALL-poly2, and QALL-poly3, respectively. Obviously, we can also optimize the per-flowlet granularity of QALL-Flowlet with the same method.

VI. EXPERIMENT EVALUATION

A. System Development

Following the experimental setup adopted in QDAPS [13], Contra [17], and CLB [27], etc., we evaluated the performance of QALL on a large-scale DCN constructed with Mininet+ BMv2, where Mininet was used to create a leaf-spine topology (Fig. 7) and BMv2 was installed as the software programmable switch. We implemented QALL using programming protocol-independent packet processors (P4). Each switch port included only a single FIFO queue. As shown in Fig. 7, following CONGA [1] and Clove [7], we used a two-tier Clos topology with two spine switches (*S*1 and *S*2) connecting two leaf switches (L1 and L2) under a set of actual DCN traffic workloads to test the proposed QALL method. Routing was performed such that all traffic received by a spine switch from one of the leaf switches was forwarded towards the other leaf switch. Each leaf switch was connected to either spine by two 400 Mbps links. This yielded a total of 1600 Mbps for the bisection bandwidth. Each leaf was connected to 16 servers with 100 Mbps links. This ensures that the network avoids oversubscribing, and the 16 servers on one leaf can saturate the 1600Mbps bandwidth together.

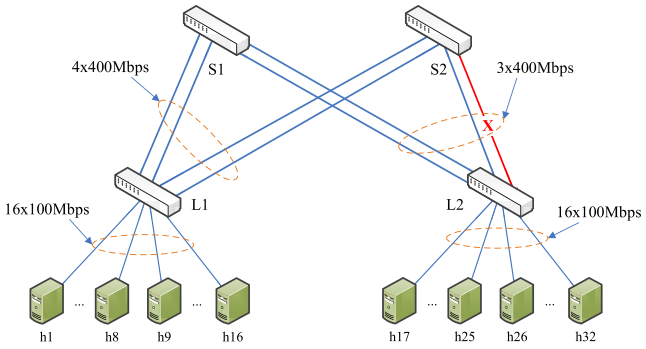

Fig. 7. Topology used in evaluation.

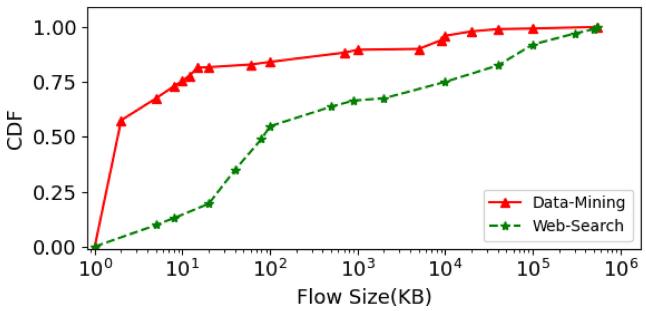

Fig. 8. Traffic distributions of DM and WS workload.

B. Actual DCN Traffic Workloads

Following the majority of studies on this subject [1], [2], [4], [5], we simulated actual DCN traffic using two types of widely used workloads, including Web-search (WS) and data-mining (DM) workloads. Fig. 8 shows the cumulative distribution function (CDF) of the flow sizes from the WS and DM workloads. In these workloads, most of the flows were mice flows with a size of less than 100 KB, whereas a smaller number of flows were elephant flows larger than 10MB. For example, in the WS workload, more than 60% of the flows were mice flows, and 25% were elephant flows. It may be

observed in the DM workload that 80% of the flows were mice flows and 10% were elephant flows.

The flows arrive according to the Poisson process with flow arrival rates λ (flows/s), and the source and destination of each flow are selected uniformly at random. To emulate various degrees of load, we scaled the flow interarrival times. That is, we used different values of λ to simulate different traffic load levels ρ, where $\rho = \frac{\lambda \times E(F)}{\text{Link bandwidth}}$ and $E(F)$ is the average flow size, and *Link bandwidth* is the bandwidth of the link. In this study, we varied λ and $E(F)$ dynamically to let ρ be 10%-90%, and the default value of ρ was 70%.

C. Performance Evaluation Methodology

The QALL under per-packet granularity is referred to as QALL-Pkt, and QALL under per-flowlet granularity is referred to as QALL-Flowlet. Because QALL is a per-packet/per-flow granularity load-balancing method located at switches, we compared QALL with other load-balancing schemes located at switches, including a typical per-packet scheme (DRILL), two typical per-flowlet schemes (CONGA and LetFlow), and a typical per-flow scheme (ECMP). We used *decision delay*, flow completion time (FCT), *network shock* (evaluated by the *Variance* of queue occupancy), *load sharing accuracy*, packet reordering (evaluated by the number of TCP duplicate ACKs), and system overhead (evaluated by the resource and control loop overhead) to test their performance at scale.

The *load sharing accuracy* varies over time, and the *load sharing accuracy* at i^{th} time slot is defined as $P_i = \frac{U_i^{\text{Max}} - U_i^{\text{Min}}}{U_i^{\text{Min}}} = \frac{U_i^{\text{Max}}}{U_i^{\text{Min}}} - 1$, where U_i^{Max}=Max$\{U_{i1}, U_{i2}, \ldots U_{ij}, \ldots\}$, U_i^{Min}=Min$\{U_{i1}, U_{i2}, \ldots U_{ij}, \ldots\}$, and U_{ij} is the link utilization of the j^{th} link at the i^{th} time slot. In other words, U_i^{Max} and U_i^{Min} are the link utilization of the busiest and idlest links at the i^{th} time slot, respectively. The definition actually reflects the difference between the lightly loaded and heavily loaded links; evidently, the closer the load-sharing accuracy is to 0, the better the load balance. Thus, our results suggest that the *load sharing accuracy* can quantify the level of load balancing in a DCN. Network operators prefer lower *load sharing accuracy*, which implies maximizing link utilization without packet loss and reducing investment costs.

In short, *decision delay* is a metric for evaluating the ability of the load-balancing scheme to cope with microbursts, *FCT* is a metric for evaluating the quality of experience for users, *load sharing accuracy* is a metric for evaluating the resource utilization efficiency of network operators, and system overhead is a metric for evaluating QALL's scalability.

D. Experimental Results

1) Decision Delay: In this study, decision delay refers to the time required to update the decision basis. Because the traffic load can change on a very small timescale, a long decision delay leads to a corresponding deviation from the expected load distribution for a longer duration. The decision delays for QALL, DRILL, CONGA, HULA, LetFlow, and ECMP are presented in Table II.

TABLE II
DECISION DELAY OF DIFFERENT SCHEMES

Schemes	Decision delay
QALL	0.615ms
CONGA	>=19.8ms
HULA	2.4ms-8.6ms
DRILL	0.621ms
LetFlow/ECMP	-

In CONGA, the decision-maker is the source leaf switch, where the destination leaf switches use a piggybacking network state manner to update its decision basis. Therefore, the decision delay depends on the normal traffic from the destination host to the source host. If the destination temporarily does not send packets to the source host, the decision delay increases significantly.

In HULA, each switch is a decision-maker, and probes are used to proactively disseminate link utilization information to all switches in the network. Without waiting for piggyback from the destination leaf switch, HULA's decision delay decreased compared with that of CONGA. HULA's decision delay depends on the time required for the probe packets to reach each switch from the leaf switch and is greatly affected by the network state.

In QALL and DRILL, the decision basis is updated directly within the local switch. Their decision delay comprises only the cloning time from the egress pipeline to the ingress pipeline. Thus, their decision delay was less than those of CONGA and HULA.

Compared with CONGA and HULA, QALL's decision delay was at least 96.8% and 74.3%–92.8% less, respectively. In particular, the decision delay of QALL was slightly shorter than that of DRILL, primarily because DRILL needs to read the congestion state from the registers multiple times and compare the ports corresponding to the minimum value. Compared to QALL, these operations require more processing delays.

LetFlow and ECMP do not need to sense the network state, instead directly forward traffic through a prepared fixed flow table. Thus, the decision delay is close to zero. However, such hardwired mapping without sensing is definitely performed at the expense of performance.

2) Average FCT Under Symmetrical Topology: Fig. 9 and Fig. 10 show the average FCT of QALL, DRILL, CONGA, LetFlow, and ECMP under different load levels.

We found that (*i*) the smaller the load-balancing granularity, the smaller the FCT, where QALL-Pkt <DRILL< QALL-Flowlet <CONGA <LetFlow <ECMP. In fact, this result is relatively straightforward. A smaller load-balancing granularity was found to result in a better chance of evenly distributing traffic to each available path. (*ii*) Under the same load-balancing granularity, congestion-aware schemes such as CONGA and QALL-Flowlet can perform better than non-congestion-aware load-balancing schemes (e.g., LetFlow). This is the case because congestion-aware schemes can schedule traffic according to the network state and have a better chance of allocating traffic to paths with lighter loads, rather than simply distributing traffic randomly to paths.

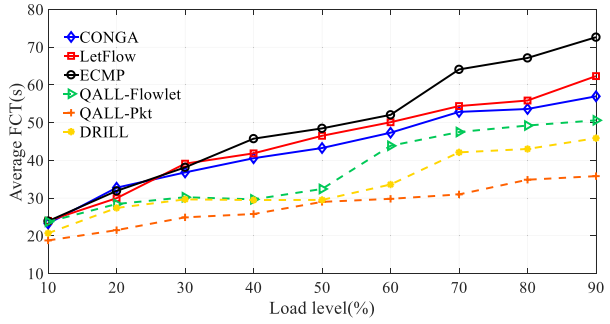

Fig. 9. Average FCT for data-mining under symmetrical topology.

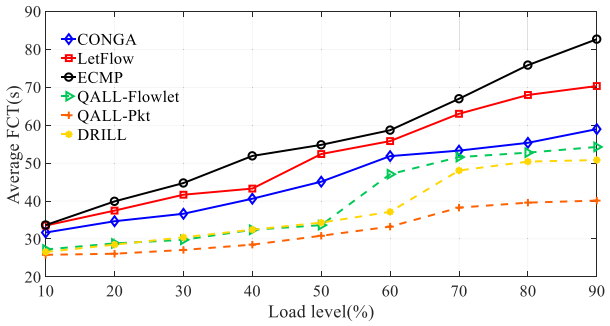

Fig. 10. Average FCT for Web-search under symmetrical topology.

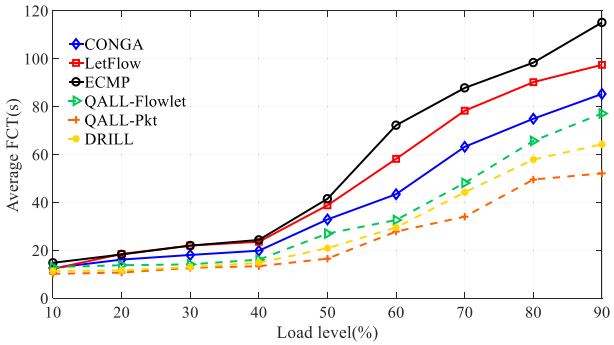

Fig. 11. Average FCT for data-mining under asymmetrical topology.

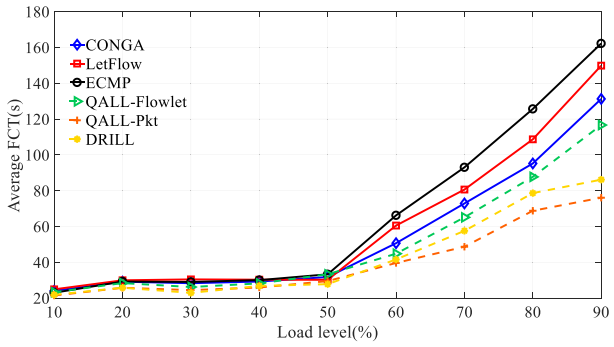

Fig. 12. Average FCT for Web-search under asymmetrical topology.

Fig. 9 shows the average FCT for data-mining, where QALL-Pkt performed best, as expected. The FCT of QALL-Pkt was lower by up to 51.7%, 43.1%, and 41.4% compared with those of ECMP, LetFlow, and CONGA, respectively. In terms of per-flowlet, the FCT of QALL-Flowlet was lower by up to 30.3% and 26.9% compared with that of LetFlow and CONGA, respectively. The main reason that QALL-Flowlet outperformed CONGA is that, depending on the manner of piggybacking, CONGA is a passive congestion-aware method and does not update the congestion state in time, whereas QALL is based on active clone packets for congestion-aware traffic management, which can update the congestion state in a more timely manner.

In terms of per-packet, the FCT of QALL-Pkt was lower by up to 26.4% compared with that of DRILL. The main reason for this result is that (*i*) equation (4) is a better method for estimating congestion, which creatively takes into account the process of how the current queue state was reached. (*ii*) Thanking for equation (3)–(4), a better method for evenly distributing the traffic to each available port, instead of selecting only the best port, as in DRILL.

Fig. 10 shows the average FCT for Web-search, which exhibited more elephant flows than data mining. As expected, QALL-Pkt yields the best results. The FCT of QALL-Pkt was lower by up to 51.4%, 42.9%, and 35.8% compared with ECMP, LetFlow, and CONGA, respectively. In terms of per-flowlet, the FCT of QALL-Flowlet was reduced by up to 35.8% and 25.4% compared with LetFlow and CONGA, respectively. In terms of per-packet, the FCT of QALL-Pkt was reduced by up to 21.5% compared with that of DRILL.

Finally, from Fig. 9 and Fig. 10, under both data mining and Web search, it may be observed that the advantages of per-packet QALL-Pkt compared with the other schemes increased with increasing load. That is, performance under per-packet granularity can be ensured in the case of a heavy load, which mainly benefits from the fine-grained load-balancing strategy, and traffic can still be evenly distributed to each available port.

3) Average FCT Under Asymmetrical Topology: To simulate asymmetry in the baseline symmetric topology, we disabled one of the 400Mbps links connecting the spine $S2$ with leaf switch $L2$. The average FCT under the asymmetric topology is shown in Fig. 11 and Fig. 12. We can see that the FCT of all schemes increased rapidly with the increase in load after the load level was greater than 50% and is larger than that of the symmetrical topology under the same load level.

Fig. 11 shows the average FCT for data-mining, where QALL-Pkt performed the best, as expected. The FCT of QALL-Pkt was lower by at most 61.4%, 57.3%, and 49.7% compared with those of ECMP, LetFlow, and CONGA, respectively. In terms of per-flowlet, the FCT of QALL-Flowlet was lower by at most 43.8% and 24.7% compared with that of LetFlow and CONGA, respectively. In terms of per-packet, the FCT of QALL-Pkt was lower by at most 23% compared with that of DRILL.

Fig. 12 shows the average FCT for Web-search which has more elephant flows than data-mining. As expected, QALL-Pkt yields the best results. The FCT of QALL-Pkt was reduced by at most 53.1%, 49.2%, and 42.1% compared with that of ECMP, LetFlow, and CONGA, respectively. In terms of per-flowlet, the FCT of QALL-Flowlet was lower by at most 25.9% and 11.5% compared with those of LetFlow and

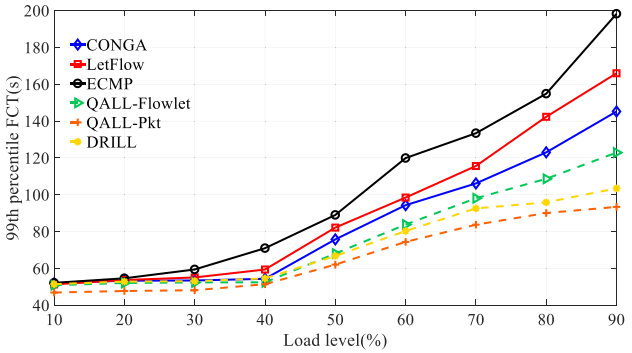

Fig. 13. 99th percentile FCT for data-mining under asymmetrical topology.

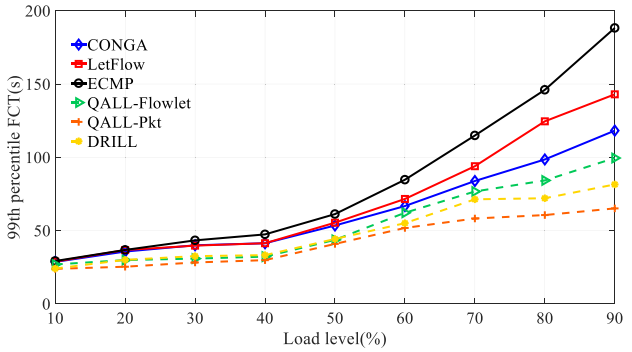

Fig. 14. 99th percentile FCT for Web-search under asymmetrical topology.

CONGA, respectively. In terms of per-packet, the FCT of QALL-Pkt was up to 15.4% lower than that of DRILL.

In factually, a symmetric topology naturally has a certain load balancing ability; for example, ECMP depends on this feature to achieve load balancing. In an asymmetric topology, the available bandwidth is reduced, which tests the capabilities of the load balancing strategy further. However, based on more accurately estimating congestion, more evenly distributing traffic, and locally making decisions, QALL can still achieve better load balancing performance under an asymmetric topology with a heavy traffic load. Thus, for most load levels, the advantage of QALL compared to other schemes under an asymmetric topology was larger than that under a symmetric topology, especially compared to ECMP.

4) 99th Percentile FCT Under Asymmetrical Topology: Different from average FCT, from another view, we used the 99th percentile FCT to evaluate the tail latency of load balancing. As an example, Fig. 13 and Fig. 14 show the 99th percentile FCT under asymmetrical topology where QALL-Pkt was the best. In fact, there was also a similar result under a symmetrical topology.

Fig. 13 shows the 99th percentile FCT for data-mining. The 99th percentile FCT of QALL-Pkt was lower by at most 52.9%, 43.7%, and 35.7% compared with those of ECMP, LetFlow, and CONGA, respectively. In terms of per-flowlet, the 99th percentile FCT of QALL-Flowlet was reduced by at most 25.9% and 15.4% compared with that of LetFlow and CONGA, respectively. In terms of per-packet, the 99th percentile FCT of QALL-Pkt was reduced by at most 9.7% compared with that of DRILL.

TABLE III
THE VARIANCE OF QUEUE OCCUPANCY

Schemes	The variance of queue occupancy
QALL-Pkt	0.0667
DRILL	0.0712
QALL-Flowlet	0.0755
CONGA	0.0922
LetFlow	0.1028
ECMP	0.1593

Fig. 14 shows the 99th percentile FCT for the Web-search workload. The 99th percentile FCT of QALL-Pkt was lower by up to 65.4%, 54.4%, and 44.9% compared with those of ECMP, LetFlow, and CONGA, respectively. In terms of per-flowlet, the 99th percentile FCT of QALL-Flowlet was lower by at most 32.3% and 22.7% compared with those of LetFlow and CONGA, respectively. In terms of per-packet, the 99th percentile FCT of QALL-Pkt was lower by up to 20.1% compared with that of DRILL.

In summary, we can also see that the advantage of QALL compared to other schemes under the Web-search was greater than for data-mining workloads. However, Web-search have more elephant flows than data-mining workloads. In other words, in terms of the 99th percentile FCT, QALL can better help elephant flows than mice flows, because the tail latency is more important for elephant flows than for mice flows. At the same time, in terms of the 99th percentile FCT, the fine-grained schemes were better than the coarse-grained schemes, and QALL-Pkt performed better than DRILL because DRILL suffered from network shock caused by its coarse-grained port selection strategy.

5) Network Shock Under Symmetrical Topology: Many load-balancing schemes (e.g., CONGA, HULA, and DRILL) usually adopt a coarse-grained port selection strategy. Such a strategy easily results in frequent changeover of paths (i.e., *network shock*). Such *network shock* will eventually be manifested as a shock in queue occupancy. In this section, we use the *Variance* of queue occupancy to evaluate the *network shock*. As an example, for the relatively high load level (i.e., ρ =70%) under symmetrical topology, the *Variance* of queue occupancy is shown in Table IV. In fact, we also obtained a similar result under an asymmetrical topology.

In essence, the load balancing scheme aims to adjust and finally achieve a reasonable space-time distribution of traffic in the network. The change of traffic distribution in the network eventually leads to a change in the queue behavior of the switches. In other words, an unbalanced traffic distribution leads to a large *Variance* in queue occupancy and affects the FCT. Thus, from Table III, we can see that the pros and cons of *Variance* under various schemes are basically the same as the pros and cons of FCT as discussed above. The *Variance* of QALL-Pkt was 1.5× and 2.4× lower than those of LetFlow and ECMP, respectively. These results further verify *Motivation* 1, in which QALL uses the relationship between the congestion degree, *queue occupancy*, and *dequeue time interval* as part of its decision basis.

More importantly, well-balanced and non-shocked queue occupancies can allow the network delay across all paths

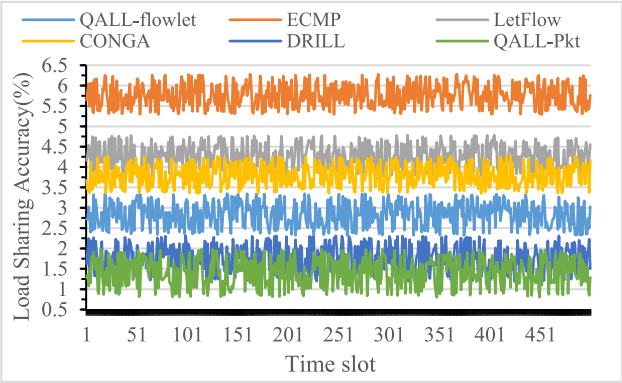

Fig. 15. Load Sharing Accuracy.

between every source and destination pair to be similar, thereby reducing packet reordering.

6) Load Sharing Accuracy: Network operators desire lower *load sharing accuracy* (generally less than 5%). Under the relatively high load level (i.e., ρ =70%), the *load sharing accuracy* is shown in Fig. 15, where every 10s is one time slot.

The *load sharing accuracy* of QALL can always be maintained within 5%, where the average *load sharing accuracy* of QALL-Pkt and that of QALL-Flowlet was 1.414% and 2.8%, respectively. The average *load sharing accuracy* of DRILL, CONGA, Flowlet, ECMP were 1.75%, 3.8%, 4.27%, and 5.76%, respectively, and the *load sharing accuracy* of ECMP was always higher than 5%. Obviously, these results were caused by the decision delay. As shown in Table II, the decision delay of QALL was the shortest, indicating that QALL was able to perceive the network state in real time and adapt to changing traffic patterns more flexibly. Clearly, the help of sensing with a shorter decision delay inevitably leads to a smaller *load sharing accuracy* and finally to a smaller FCT and *Variance* in queue occupancy. Thus, as expected, the advantages and disadvantages of *load sharing accuracy* under various schemes are basically the same as those of the FCT and *Variance* as mentioned above. These results further verify *Motivation* 3, in which QALL had better *data locality*. In short, these results confirm that QALL mitigated traffic imbalance, which was reflected by performance improvements as mentioned (i.e., lower FCT).

7) Average FCT of Data-Driven QALL: Fig. 16 and Fig. 17 show the average FCT of QALL-Pkt, QALL-Linear, QALL-Poly2, and QALL-Poly3 under a symmetrical topology. We also obtained similar results under an asymmetrical topology.

Compared with QALL-Pkt, QALL-Poly2 and QALL-Poly3 improved the average FCT; however, QALL-Linear did not improve. For example, under data-mining and Web-search with a load level of 80%, compared with QALL-Pkt, QALL-Poly2 was reduced by at most 5.4% and 4.7%, respectively. Under data-mining and Web-search with a load level of 80%, compared with QALL-Pkt, QALL-Poly3 was reduced by up to 7.8% and 4.9%, respectively. Finally, Fig. 16 and Fig. 17 show that the advantages of QALL-Poly2 and QALL-Poly3

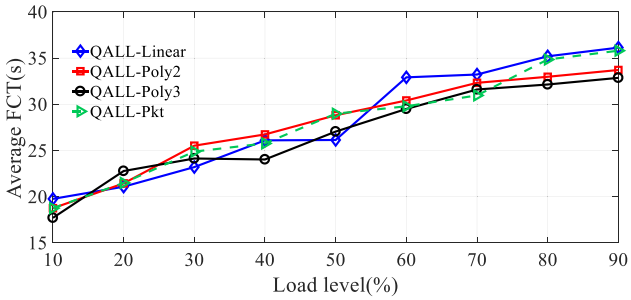

Fig. 16. Average FCT of data-driven QALL for data-mining under symmetrical topology.

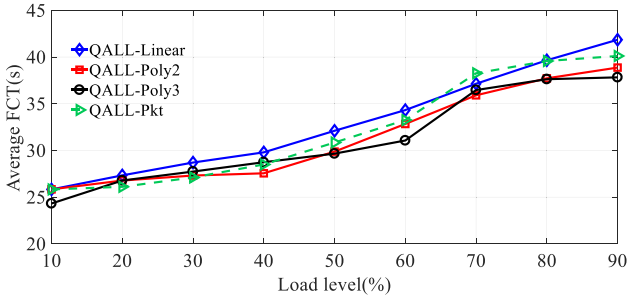

Fig. 17. Average FCT of data-driven QALL for Web-search under symmetrical topology.

compared with QALL-Pkt increase with increasing levels of traffic load level for both data-mining and Web-search.

Obviously, the reason behind these results is the quantitative fitting of the relationship between egress traffic, *queue occupancy*, and the *dequeue time interval*. The V_i and τ of QALL-Pkt are empirically determined according to some reasonable rules. However, the V_i and τ of QALL-Linear, QALL-Poly2, and QALL-Poly3 have been more accurately determined by data-driven method. Table I shows that the fitting accuracy improved when the power of the fitting function increased. Correspondingly, in terms of average FCT, the following trend was clearly evident: QALL-Poly3<QALL-Poly2<QALL-Pkt ≈QALL-Linear. In short, the higher the fitting accuracy, the lower the FCT. In particular, the fitting accuracy of the primary power for QALL-Linear was too low to help improve the FCT.

E. Packet Reordering

Packet reordering may trigger a duplicate ACK mechanism and could thus degrade TCP performance. Because TCP detects packet loss [4] and then reduces its transmission rate when duplicate ACKs exceed the retransmission threshold. Many load-balancing schemes such as ECMP [3], CONGA [1], and Presto [8] avoid packet reordering by balancing coarser units of traffic, but at the expense of performance.

It is well known that queuing delay is the main source of network delay in DCNs [4]. In QALL, the well-balanced load (Fig. 15) and extremely low variance of *queue occupancy* (Table III) imply that packets experience almost identical queuing delays regardless of the paths they take (i.e., packets nearly always arrive in order despite traversing different paths).

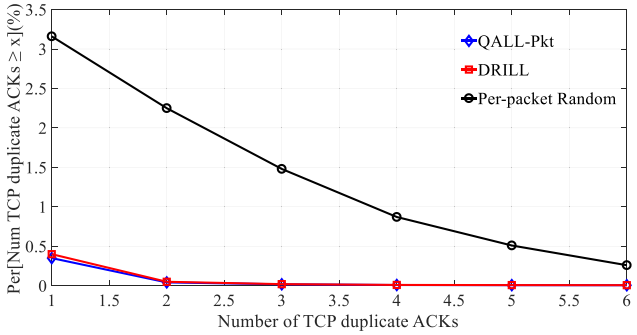

Fig. 18. Number of TCP duplicate ACKs under 70% load level.

Our experimental results confirm this hypothesis. Therefore, packet reordering is minimal in QALL-Pkt. Fig. 18 shows the amount of reordering measured in terms of the number of TCP duplicate ACKs under a load level of 70%. We compared QALL-Pkt to DRILL and per-packet *Random* (a typical no load-awareness scheme), which forwards each packet along an independent random shortest path.

Per-packet load balancing makes fine-grained forwarding decisions for each packet independent of other packets of the same flow. This is expected to cause excessive packet reordering. However, QALL-Pkt and DRILL can also cause minimal packet reordering if the delays along multiple paths differ by less than the time between packets in a flow [4]. Fig. 18 confirms that the degree of reordering under QALL-Pkt and DRILL rarely reached the TCP retransmission threshold, even under heavy load. For QALL-Pkt and DRILL, only 0.35% and 0.41% of the flows have one or more duplicate ACKs, respectively.

Furthermore, in terms of QALL-Pkt, only 0.018% of the flows exceeded the typical TCP retransmission threshold of 3. This was lower by 10.1% and 98.8% compared to DRILL and per-packet Random, respectively. This observation confirms that TCP performance is not significantly impacted and also further explains why QALL's FCT is low despite reordering.

In addition, when certain specialized applications are required to eliminate all packet reordering, recent techniques for building reordering-resilient network stacks can address occasional reordering. For example, similar to prior works [4], [8], we can optionally deploy a buffer in the host generic receive offload (GRO) layer to restore the correct ordering.

F. System Overhead

This section presents the system overhead added by load balancing schemes to switches, including the resource and control loop overheads.

1) The Resource Overhead: Table IV presents the additional CPU and memory utilizations added by the load balancing schemes to the switches under a load level of 70%. We can observe that QALL consumes the least resources (whether CPU or memory) compared to DRILL, ECMP, CONGA, and LetFlow. This small resource overhead does not affect the line-speed forwarding of switches. On the one hand, the reason behind these results is the extremely low complexity

TABLE IV
RESOURCE OVERHEAD OF LOAD-BALANCING SCHEMES

Schemes	CPU utilization	Memory utilization
QALL-Pkt	6.51%	2.27%
QALL-Flowlet	7.42%	2.88%
DRILL	7.84%	3.74%
ECMP	7.98%	4.41%
CONGA	8.89%	3.59%
LetFlow	7.56%	4.35%

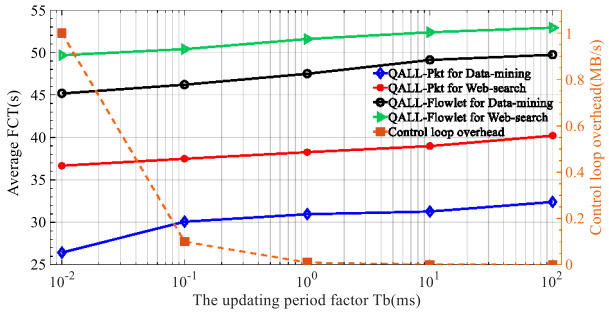

Fig. 19. Average FCT of QALL effected by the updating period factor under symmetrical topology.

of QALL, because each switch in QALL independently makes decision for load balancing according to the fine-grained-measured local queue behavior, not requiring cooperating with other switches (as shown in CONGA), or comparing with the previous load state (as shown in DRILL). The time complexity and space complexity of QALL is $O(1)$ and $O(P)$ respectively, where P is the number of egress ports. However, the P is a very small value which is less than 4 in the widely used leaf-spine or Fat-tree topology. On the other hand, in order to further improve the computing and storing efficiency, we use a hash operation to replace directly storing the flowlet index based on packet's five-tuples, as shown in Line 11 of Algorithm 1.

2) The Control Loop Overhead: For every T_b (i.e., the updating period factor), QALL sends a clone packet back to the ingress pipeline from the egress pipeline (i.e., the control loop). As shown in Fig. 6, a clone packet requires 10 bytes; thus, the actual control loop overhead was $\frac{10}{T_b}$ (bytes/s). In this study, T_b was set to 1ms by default. We observed the effect of a faster or slower control loop (i.e., a smaller or larger T_b) on the control loop overhead introduced by the clone packets and the accuracy of the decision. Obviously, the faster the control loop (i.e., the lower the T_b), the fresher (more accurate) the decision-making basis becomes, and the greater the control loop overhead. A faster control loop implies that QALL can better deal with cases of queue quickly building up, such as microbursts.

As shown by Fig. 19, when the T_b is decreased from 1ms to 0.01ms, the control loop overhead is increased from 0.01MB/s to 1MB/s, however the average FCT is decreased by up to 17%. In other words, the most control loop overhead (i.e., 1MB/s) is also little very much to the switches. The experimental results also confirmed that QALL can deal with microbursts better at the cost of a small overhead when T_b decreases. On the other hand, we can also see that, the overhead decreases to 100B/s when the T_b is decreased to

100ms, however QALL can yet reduce FCT compared to ECMP, CONGA, LetFlow, and DRILL.

VII. Practical Issues and Future Work

In this section, we analyze some relevant practical issues to suggest some avenues for future research.

A. Improving Generalization of Data-Driven QALL

As discussed in Section V, data-driven QALL uses a fitting function of egress traffic, *queue occupancy,* and *dequeue time interval* to aid in making load-balancing decisions. The experimental results show that the data-driven QALL method performs better than the original QALL method. Specifically, the greater the power of the fitting function, the better the load-balancing performance. Such fitting is also a machine learning-based method, but its generalization is relatively poor. Because the fitted function is actually a tailored load-balancing strategy for a specific traffic pattern (i.e., if the traffic pattern changes, the fitting function becomes unsuitable and the performance of load balancing worsens), further improvement in the generalization of data-driven QALL should be investigated in future research. The following ideas should be considered to facilitate these investigations.

(1) Employing reinforcement learning techniques on programmable switches to learn the queue behavior. Reinforcement learning techniques have the self-learning ability to adapt to dynamically changing traffic patterns.

(2) In DCNs, there are four typical applications including data-mining, Web-search, cache, and Hadoop. Thus, based on the combination of the four main applications, there are at most $C_4^1 + C_4^2 + C_4^3 + C_4^4 = 15$ patterns. For these 15 patterns, we can use least squares or other fitting methods to fit 15 sets of fitting functions, similar to Equations (11)–(13), which are one of these 15 patterns, and then store them in each switch. When a certain pattern appears in the DCN, the switches select the equation corresponding to the pattern and use Equations (3) and (4) to perform load balancing.

B. Improving Performance

The space-time mismatch between the load-balancing decision location and its decision basis should be expected to decrease the performance of QALL. In particular, the time required by clone packets is the main factor in decision delay. However, currently, a few commodity programmable switches can access queue behavior in the ingress pipeline (e.g., Tofino 2 [40]). That is, removing the clone operation can reduce the decision delay, and the decision basis can more accurately reflect the state of the network. Based on the real-time network states, we believe that the performance of QALL should be further improved in future studies.

C. Discussions for Observations

Considering that the observed correlation between *dequeue time interval* and egress traffic rate is moderate rather than extremely high, the conclusion that a longer packet *dequeue time interval* is indicative of a lighter load is not completely fixed. There are a few special cases. One example is when *queue occupancy* is high and *dequeue time interval* is long, which may be an indication of the outgoing link potentially being congested. Another example is when we further consider the size of packets: a smaller number of large packets within the same time window as a higher number of small packets have an equal or greater likelihood of causing congestion. It should be noted that Equations (2)–(4) only indicate the positively/negatively correlated relationship between C and L as well as T, and this correlation is also rough and approximate. Therefore, when L is high and T is long simultaneously in Equations (2)–(4), C may be small or large (corresponding to light or congested conditions), which is determined by the relative value of L and T. In other words, Equations (2)–(4) implicitly include the above-mentioned special cases. In addition, the observations in this study are based on experiments in BMv2 software switches, and whether these observations in hardware switch deployment hold true remains an open issue. In the future, we will observe the queue behavior during hardware switch deployment.

VIII. Conclusion

We observed that queue behavior on a switch can reflect the current and future congestion degrees in a network. Therefore, we have proposed an in-network load-balancing scheme called QALL. In QALL, each switch independently selects the egress port probabilistically, according to the fine-grained-measured local queue behavior. The key concept is that QALL creatively takes account the evolutionary process of reaching the current queue state into its decision basis for load balancing. Based on an accurate fitting of the queue behavior, we have also proposed a data-driven QALL to improve the load-balancing performance further. The experimental results under actual DCN workloads show that QALL performed better than several existing schemes in terms of lower FCT, shorter decision delay, and smaller load sharing accuracy. In addition, QALL does not depend on the symmetrical characteristics of the network topology.

In future works, the following factors can be considered. First, the experimental results show that data-driven QALL can perform slightly better. One possible reason for this is that there are insufficient training data to fit a more accurate function. In fact, from R^2 and *MRE* values in Table I, we can see that the fitting function has room for improvement. Because a higher fitting accuracy can result in a lower FCT, we need to find a simpler fitting function with greater accuracy to improve QALL. Second, we can further explore data-driven QALL with better generalizations for different types of traffic patterns.

Appendix
Observing Elephant Flows/Mice Flows Alone

Considering only elephant flows and mice flows alone, QALL exhibited excellent performance for data-mining and Web-search traffic, under either symmetrical topology or asymmetrical topology.

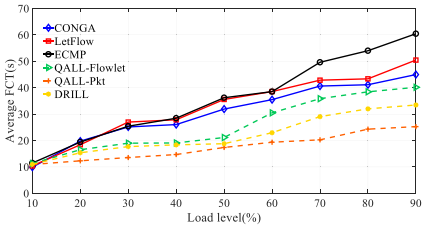

Fig. 20. Average FCT for mice flows of data-mining under symmetrical topology.

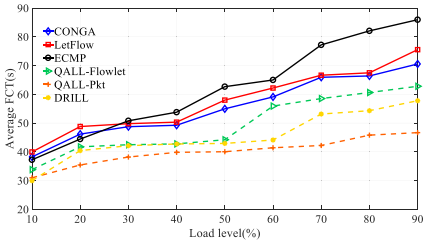

Fig. 21. Average FCT for elephant flows of data-mining under symmetrical topology.

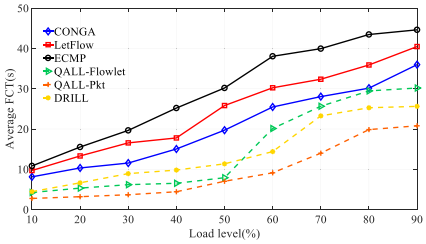

Fig. 22. Average FCT for mice flows of Web-search under symmetrical topology.

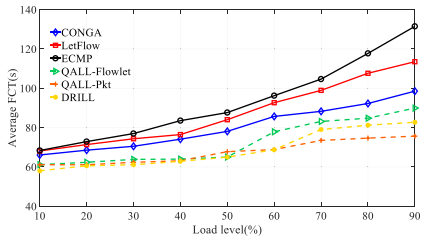

Fig. 23. Average FCT for elephant flows of Web-search under symmetrical topology.

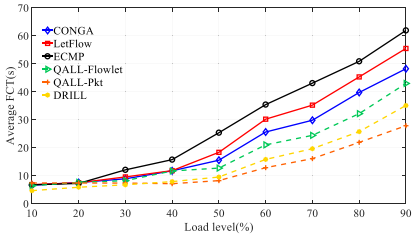

Fig. 24. Average FCT for mice flows of data-mining under asymmetrical topology.

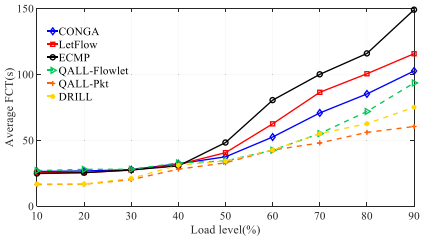

Fig. 25. Average FCT for elephant flows of data-mining under asymmetrical topology.

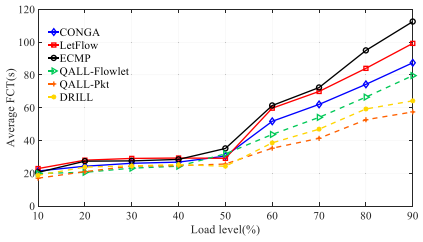

Fig. 26. Average FCT for mice flows of Web-search under asymmetrical topology.

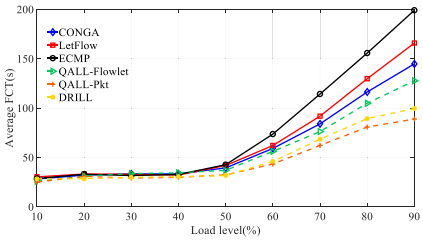

Fig. 27. Average FCT for elephant flows of Web-search under asymmetrical topology.

A. Average FCT Under Symmetrical Topology

The average FCT under data-mining is shown in Fig. 20 and Fig. 21. For data-mining mice flows, the FCT of QALL-Pkt was lower than that of ECMP, LetFlow, and CONGA by up to 45.7%, 38.2%, and 33.8%, respectively. In terms of per-packet, the FCT of QALL-Pkt was lower by up to 24.5% compared with that of DRILL. In terms of per-flowlet, the FCT of QALL-Flowlet was reduced by at most 20.4% and 10.7% compared with that of LetFlow and CONGA, respectively. For data-mining elephant flows, the FCT of QALL-Pkt was reduced by 58%, 49.9%, 43.8%, and 19.2% compared with that of ECMP, LetFlow, CONGA, and DRILL, respectively. In terms of per-flowlet, the FCT of the QALL-Flowlet was reduced by at most 16.9% and 11% compared with that of LetFlow and CONGA, respectively.

The average FCT for Web search is shown in Figs. 22 and 23. For Web search mice flows, the FCT of QALL-Pkt was lower by up to 42.45%, 23.3%, and 23.2% compared with that of ECMP, LetFlow, and CONGA, respectively. In terms of per-packet, the FCT of QALL-Pkt was lower by up to 18.8% compared with that of DRILL. In terms of per-flowlet, the FCT of QALL-Flowlet was lower by up to 25.4% and 16.1% compared with that of LetFlow and CONGA, respectively. For Web search elephant flows, the FCT of QALL-Pkt was reduced by up to 53.39%, 48.7%, 42.2%, and 8.5% compared with that of ECMP, LetFlow, CONGA, and DRILL, respectively. In terms of per-flowlet, the FCT of QALL-Flowlet was reduced by at most 20.7% and 5.8% compared with that of LetFlow and CONGA, respectively.

B. Average FCT Under Asymmetrical Topology

The average FCT under Data Mining is shown in Fig. 24 and Fig. 25. The average FCT under Web-search is shown in Fig. 26 and Fig. 27.

REFERENCES

[1] M. Alizadeh et al., "CONGA: Distributed congestion-aware load balancing for datacenters," in *Proc. ACM Conf. SIGCOMM*, 2014, pp. 503–514.

[2] N. P. Katta, M. Hira, C. Kim, A. Sivaraman, and J. Rexford, "HULA: Scalable load balancing using programmable data planes," in *Proc. Symp. SDN Res.*, 2016, pp. 1–12.

[3] C. Hopps, "Analysis of an equal-cost multi-path algorithm," Internet Eng. Task Force, Fremont, CA, USA, RFC 2992, 2000.

[4] S. Ghorbani, Z. Yang, P. B. Godfrey, Y. Ganjali, and A. Firoozshahian, "DRILL: Micro load balancing for low-latency data center networks," in *Proc. Conf. ACM Special Interest Group Data Commun.*, 2017, pp. 225–238.

[5] E. Vanini, R. Pan, M. Alizadeh, P. Taheri, and T. Edsall, "Let it flow: Resilient asymmetric load balancing with flowlet switching," in *Proc. 14th USENIX Symp. Netw. Syst. Design Implement. (NSDI)*, 2017, pp. 407–420.

[6] M. Al-Fares, S. Radhakrishnan, B. Raghavan, N. Huang, and A. Vahdat, "HeDera: Dynamic flow scheduling for data center networks," in *Proc. NSDI*, vol. 10, 2010, pp. 89–92.

[7] N. P. Katta et al., "Clove: Congestion-aware load balancing at the virtual edge," in *Proc. 13th Int. Conf. Emerg. Netw. Exp. Technol.*, 2017, pp. 323–335.

[8] K. He, E. Rozner, K. Agarwal, W. Felter, J. Carter, and A. Akella, "Presto: Edge-based load balancing for fast datacenter networks," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 45, no. 4, pp. 465–478, 2015.

[9] X. Diao, H. Gu, X. Yu, L. Qin, and C. Luo, "Flex: A flowlet-level load balancing based on load-adaptive timeout in DCN," *Future Gener. Comput. Syst.*, vol. 130, pp. 219–230, May 2022.

[10] J.-L. Ye, C. Chen, and Y. H. Chu, "A weighted ECMP load balancing scheme for data centers using P4 switches," in *Proc. IEEE 7th Int. Conf. Cloud Netw. (CloudNet)*, 2018, pp. 1–4.

[11] B. Augustin, T. Friedman, and R. Teixeira, "Multipath tracing with Paris traceroute," in *Proc. 5th IEEE/IFIP Workshop End-to-End Monitor. Techn. Services*, 2007, pp. 1–8.

[12] S. Zou, J. Huang, J. Wang, and T. He, "RMC: Reordering marking and coding for fine-grained load balancing in data centers," *IEEE Trans. Commun.*, vol. 69, no. 12, pp. 8363–8374, Dec. 2021, doi: 10.1109/TCOMM.2021.3118467.

[13] J. Huang, W. Lyu, W. Li, J. Wang, and T. He, "Mitigating packet reordering for random packet spraying in data center networks," *IEEE/ACM Trans. Netw.*, vol. 29, no. 3, pp. 1183–1196, Jun. 2021, doi: 10.1109/TNET.2021.3056601.

[14] K.-F. Hsu, P. Tammana, R. Beckett, A. Chen, J. Rexford, and D. Walker, "Adaptive weighted traffic splitting in programmable data planes," in *Proc. Symp. SDN Res.*, 2020, pp. 103–109.

[15] N. Shelly, B. Tschaen, K.-T. Förster, M. A. Chang, T. Benson, and L. Vanbever, "Destroying networks for fun (and profit)," in *Proc. 14th ACM Workshop Hot Topics Netw.*, 2015, pp. 1–7.

[16] M. Noormohammadpour and C. S. Raghavendra, "Datacenter traffic control: Understanding techniques and tradeoffs," *IEEE Commun. Surveys Tuts.*, vol. 20, no. 2, pp. 1492–1525, 2nd Quart., 2018.

[17] K.-F. Hsu, R. Beckett, A. Chen, J. Rexford, and D. Walker, "Contra: A programmable system for performance-aware routing," in *Proc. 17th USENIX Symp. Netw. Syst. Design Implement. (NSDI)*, 2020, pp. 701–721.

[18] J. Liu, J. Huang, W. Li, and J. Wang, "AG: Adaptive switching granularity for load balancing with asymmetric topology in data center network," in *Proc. IEEE 27th Int. Conf. Netw. Protocols (ICNP)*, 2019, pp. 1–11.

[19] M. Shafiee and J. Ghaderi, "A simple congestion-aware algorithm for load balancing in datacenter networks," *IEEE/ACM Trans. Netw.*, vol. 25, no. 6, pp. 3670–3682, Dec. 2017.

[20] H. Zhang, J. Zhang, W. Bai, K. Chen, and M. Chowdhury, "Resilient datacenter load balancing in the wild," in *Proc. Conf. ACM Special Interest Group Data Commun. (SIGCOMM)*, 2017, pp. 253–266.

[21] M. Handley et al., "Re-architecting datacenter networks and stacks for low latency and high performance," in *Proc. Conf. ACM Special Interest Group Data Commun. (SIGCOMM)*, 2017, pp. 29–42.

[22] F. Wang et al., "Dynamic distributed multi-path aided load balancing for optical data center networks," *IEEE Trans. Netw. Service Manag.*, vol. 19, no. 2, pp. 991–1005, Jun. 2022, doi: 10.1109/TNSM.2021.3125307.

[23] W.-X. Liu, C. Liang, Y. Cui, J. Cai, and J.-M. Luo, "Programmable data plane intelligence: advances, opportunities, and challenges," *IEEE Netw.*, early access, Dec. 5, 2022, doi: 10.1109/MNET.124.2200113.

[24] X. Chen, S. L. Feibish, Y. Koral, J. Rexford, and O. Rottenstreich, "Catching the microburst culprits with snappy," in *Proc. Afternoon Workshop on Self-Driving Netw.*, 2018, pp. 22–28.

[25] "The P4Language consortium. P416Language specications." (2018). [Online]. Available: https://p4.org/p4-spec/docs/P4–16-v1.0.0-spec.pdf

[26] S. Liu, J. Huang, W. Jiang, J. Wang, and T. He, "Reducing flow completion time with replaceable redundant packets in data center networks," in *Proc. IEEE Int. Conf. Distribut. Comput. Syst. (ICDCS)*, 2019, pp. 46–56.

[27] D. D. Robin and J. I. Khan, "CLB: Coarse-grained precision traffic-aware weighted cost multipath load balancing on PISA," *IEEE Trans. Netw. Service Manag.*, vol. 19, no. 2, pp. 784–803, Jun. 2022, doi: 10.1109/TNSM.2022.3142106.

[28] N. Rikhtegar, O. Bushehrian, and M. Keshtgari, "DeepRLB: A deep reinforcement learning-based load balancing in data center networks," *Int. J. Commun. Syst.*, vol. 34, no. 15, 2021, Art. no. e4912.

[29] Z. Cui et al., "Closer: Scalable load balancing mechanism for cloud datacenters," *China Commun.*, vol. 18, no. 4, pp. 198–212, 2021.

[30] Q. Shi, F. Wang, and D. Feng, "IntFlow: Integrating per-packet and per-flowlet switching strategy for load balancing in datacenter networks," *IEEE Trans. Netw. Service Manag.*, vol. 17, no. 3, pp. 1377–1388, Sep. 2020, doi: 10.1109/TNSM.2020.2990868.

[31] C. H. Benet and A. J. Kassler, "FlowDyn: Towards a dynamic flowlet gap detection using programmable data planes," in *Proc. IEEE 8th Int. Conf. Cloud Netw. (CloudNet)*, 2019, pp. 1–7.

[32] E. Dong, X. Fu, M. Xu, and Y. Yang, "Low-cost datacenter load balancing with multipath transport and top-of-rack switches," *IEEE Trans. Parallel Distrib. Syst.*, vol. 31, no. 10, pp. 2232–2247, Oct. 2020, doi: 10.1109/TPDS.2020.2989441.

[33] A. H. Alawadi and S. Molnár, "Oddlab: Fault-tolerant aware load-balancing framework for data center networks," *Ann. Telecommun.*, vol. 77, pp. 641–662, Oct. 2022.

[34] K. Sharma and R. N. Yadav, "An adaptive, fault tolerant, flow-level routing scheme for data center networks," *Comput. Netw.*, vol. 175, Jul. 2020, Art. no. 107235.

[35] J. Xue, M. Usama Chaudhry, B. Vamanan, T. N. Vijaykumar, and M. Thottethodi, "Dart: Divide and specialize for fast response to congestion in RDMA-based datacenter networks," *IEEE/ACM Trans. Netw.*, vol. 28, no. 1, pp. 322–335, Feb. 2020.

[36] W.-X. Liu, J. Lu, J. Cai, Y. Zhu, S. Ling, and Q. Chen, "DRL-PLink: Deep reinforcement learning with private link approach for mix-flow scheduling in software-defined data-center networks," *IEEE Trans. Netw. Service Manag.*, vol. 19, no. 2, pp. 1049–1064, Jun. 2022, doi: 10.1109/TNSM.2021.3128267.

[37] J. Y. Yen, "Finding the K shortest loopless paths in a network," *Manag. Sci.*, vol. 17, no. 11, pp. 712–716, 1971.

[38] Y. Zhang, P. Tiňo, A. Leonardis, and K. Tang, "A survey on neural network interpretability," *IEEE Trans. Emerg. Topics Comput. Intell.*, vol. 5, no. 5, pp. 726–742, Oct. 2021, doi: 10.1109/TETCI.2021.3100641.

[39] Å. Björck, "Least squares methods," in *Handbook of Numerical Analysis*, vol. 1. Amsterdam, The Netherlands: Elsevier, 1990, pp. 465–652.

[40] Z. Yu et al., "Programmable packet scheduling with a single queue," in *Proc. ACM SIGCOMM Conf.*, 2021, pp. 179–193.

[41] N. Michael and A. Tang, "HALO: Hop-by-hop adaptive link-state optimal routing," *IEEE/ACM Trans. Netw.*, vol. 23, no. 6, pp. 1862–1875, Dec. 2015.

[42] Y. Lei, L. Yu, V. Liu, and M. Xu, "PrintQueue: performance diagnosis via queue measurement in the data plane," in *Proc. ACM SIGCOMM Conf. (SIGCOMM)*, New York, NY, USA, pp. 516–529.

[43] M. A. Qureshi et al., "PLB: congestion signals are simple and effective for network load balancing," in *Proc. ACM Special Interest Group Data Commun. (SIGCOMM)*, New York, NY, USA, 2022, pp. 207–218.

[44] Y. Yuan et al., "Unlocking the power of inline floating-point operations on programmable switches," in *Proc. 19th USENIX Symp. Netw. Syst. Design Implement. (NSDI)*, 2022, pp. 683–700.

[45] Y. Li et al., "HPCC: High precision congestion control," in *Proc. ACM Special Interest Group Data Commun. (SIGCOMM)*, 2019, pp. 44–58.

[46] S. Abdous, E. Sharafzadeh, and S. Ghorbani, "Burst-tolerant datacenter networks with Vertigo," in *Proc. 17th Int. Conf. Emerg. Netw. Exp. Technol. (CoNEXT)*, New York, NY, USA, 2021, pp. 1–15.
[47] Q. Zhang, V. Liu, H. Zeng, and A. Krishnamurthy, "High-resolution measurement of data center microbursts," in *Proc. Internet Meas. Conf.*, 2017, pp. 78–85.
[48] G. Kim and W. Lee, "LossPass: Absorbing microbursts by packet eviction for data center networks," *IEEE Trans. Cloud Comput.*, vol. 10, no. 4, pp. 2717–2728, Oct.–Dec. 2022, doi: 10.1109/TCC.2021.3054664.
[49] T. Hellemans and B. Van Houdt, "Improved load balancing in large scale systems using attained service time reporting," *IEEE/ACM Trans. Netw.*, vol. 30, no. 1, pp. 341–353, Feb. 2022, doi: 10.1109/TNET.2021.3110186.

Wai-Xi Liu (Member, IEEE) received the Ph.D. degree from Sun Yat-sen University, China, in 2013. He is currently an Associate Professor with the Department of Electronic and Communication Engineering, Guangzhou University, China. He has published over 25 research papers. His research interests focus on future networks, SDN, and programmable data plane.

Jun Cai received the B.S. degree from Hunan Normal University, China, in 2003, the M.S. degree from Jinan University, China, in 2006, and the Ph.D. degree from Sun Yat-sen University, China, in 2012. He is currently a Professor with the School of Cyber Security, Guangdong Polytechnic Normal University, China. He is interested in the research of complex networks, anomaly detection, SDN.

Sen Ling is currently pursuing the master's degree in electronic and communication engineering with Guangzhou University. His current research interests are in the area of software defined networking and data center network traffic measuring.

Jian-Yu Zhang is currently pursuing the master's degree in electronic and communication engineering with Guangzhou University. His current research interests are in the area of programmable data plane.

Qingchun Chen (Senior Member, IEEE) received the B.Sc. and M.Sc. degrees (Hons.) from Chongqing University, China, in 1994 and 1997, respectively, and the Ph.D. degree from Southwest Jiaotong University, China, in 2004. He is currently a Professor with Guangzhou University. He has published over 100 research papers. His research interest includes wireless communication.