IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received December 3, 2021, accepted December 16, 2021, date of publication December 30, 2021,
date of current version January 7, 2022.

Digital Object Identifier 10.1109/ACCESS.2021.3139642

Real-Time In-Network Microburst Mitigation on Programmable Switch

YU-JIE LIN[1], CHI-HSIANG HUNG[2], (Member, IEEE), AND CHARLES H.-P. WEN[1], (Member, IEEE)
[1]Department of Electrical and Computer Engineering, National Yang Ming Chiao Tung University, Hsinchu 30010, Taiwan
[2]Advanced SDN Technology Research Center, National Yang Ming Chiao Tung University, Hsinchu 30010, Taiwan

Corresponding author: Chi-Hsiang Hung (hungch@g2.nctu.edu.tw)

This work was supported by the Center for Open Intelligent Connectivity from The Featured Areas Research Center Program within the framework of the Higher Education Sprout Project by the Ministry of Education (MOE), Taiwan.

ABSTRACT Microbursts in the datacenter network (DCN) last for an extremely short time in switches and are difficult to discover from a coarse-grained perspective. Most prior works are dedicated to in-network detection of microbursts and have not yet attempted to mitigate them in real time. Therefore, a realtime microburst mitigator, called RIMM, is proposed in this work and mainly applies **orderly detours** with **flowlet intervals** for preventing packet loss and packet retransmission in the network. RIMM mainly consists of three key components: (1) detour launcher, (2) packet sequencer, and (3) post-detour handler, and can entirely work in-network on a programmable switch. Experimental results demonstrate that RIMM is capable of preventing packet loss, allowing the reduction of packet retransmissions, while effectively reducing microbursts, resulting in an enhanced data-center network performance.

INDEX TERMS Microburst mitigation, P4, programmable switch, in-network.

I. INTRODUCTION

With the rapid development of network technologies, various services and applications are migrated to the cloud environment. In addition to the network bandwidth, the network delay is also a key factor to user experience. For example, delays in real-time video streaming often cause intermittent connection. Even if the memory buffering [1] alleviates bandwidth problems, a time gap between the viewed screen and the actual media source still remains. Another example is financial trading [2] where network delays significantly influence the fairness of transactions, further inducing potential economic losses.

The network delay can be attributed to various factors, including transmission mediums, transmission distances, and computing capabilities of network equipment (e.g., switches or routers) Among all, the dominating factor is the queuing time in the network equipment. For a switch, such queuing delay is often caused by network congestion (i.e., packets from multiple ingress ports forwarded to one egress port). As network congestion occurs, the queue of the egress port starts to be piled up. For a newly incoming packet, the longer queue refers to the longer queuing delay. More seriously, once the queue is longer than the pre-allocated size, packets are dropped (i.e., overflow). It causes retransmissions and a long delay in flow completion.

In particular, if the network congestion only lasts for an extremely short time (e.g., <100 microseconds reported by Cisco [3] or 1∼100 milliseconds reported by Huawei [4]), it is termed a ***microburst***. As a matter of fact, *microbursts* are not a new type of traffic patterns [5] in the datacenter network (DCN) and have been discovered during financial trading [2], [6]. However, microbursts cannot be observed by traditional network management tools due to insufficient resolutions. Although a variety of congestion-control mechanisms (including loss-based, ECN (Explicit Congestion Notification)-based and RTT (Round Trip Time)-based approaches [7]–[11]) work effectively for traditional networks, their responses are not quick enough for handling microbursts.

For dealing with microbursts, micro-burst-aware transport control protocol (MATCP) [12] is proposed to consider the slope of the evolution of the queue length for adjusting sliding windows to throttle the sending rate from the sender. Another work, flow-aware adaptive pacing [13], dynamically adjusts the time interval of adjacent packets according to the flow concurrency from the sender and can jointly work with an existing congestion-control mechanism to

The associate editor coordinating the review of this manuscript and approving it for publication was Petros Nicopolitidis.

lower the probability of microbursts from occurrences. These congestion-control-based solutions first need to modify the sender's TCP protocol stack and thus are only applicable to TCP flows. Moreover, they are not precise enough to detect the microbursts, which only last for <100 microseconds.

Along with the evolution of switching technologies [14]–[16], the emergence of P4 (Programming Protocol-Independent Packet Processor) switches [17]–[19], offers potential for real-time detecting and mitigating microbursts in switches. P4 enables the customization on header fields and processing pipeline of packets in switches, as well as the support of *in-band network telemetry (INT)*. With *INT*, a P4 switch (e.g., Tofino^{TM} based Edgecore Wedge 100BF-32X switch [19]) can retrieve internal information of switches (e.g., switch ID, timestamp of packet ingress/egress, queue depth) and furthermore embed such information into the headers of packets on the nanosecond scale. Therefore, a P4 switch provides sufficient resolution for dealing with microbursts in real time.

Two recent works [20], [21] have been developed on P4 switches for microburst detection. However, these works only aim at detecting microbursts, but have not yet addressed the mitigation of the instantaneous latency caused by microbursts. For alleviating network congestion, two prior works [6], [10] applied the ECN marking with TCP to observe the characteristics of network traffic under different applications. They predict the upcoming queue overflows and, in advance, slow down the senders to prevent possible future congestion. However, the response time of such ECN-based methods works in milliseconds, which is not precise enough for microbursts. Thus a new, real-time solution for microburst mitigation needs to be developed.

In this work, a real-time in-network microburst mitigator named **RIMM** is proposed and mainly consists of three components: (1) ***detour launcher***, (2) ***packet sequencer***, and (3) ***post-detour handler***. *Detour launcher* is responsible for detecting a microburst through the observation on the depth of the egress queue on the P4 switch. Once a microburst starts, *detour launcher* promptly detours packets to an alternative port for another neighboring switch to prevent congestion. Before a detoured packet is sent out, *packet sequencer* marks a sequence number in the header of such packet to prevent the disorder of packets in the future. Last, as the microburst ends, the packet detouring is deactivated and *post-detour handler* takes over to make use of the idle time between two bursts (i.e., flowlet in TCP), for releasing detoured packets in order. As a result, RIMM successfully mitigates microbursts in the network and effectively prevents packet loss as well as packet retransmission without the intervention of the SDN controller or modification on sender's TCP protocol stack.

The proposed real-time in-network microburst mitigator (RIMM) was implemented on a Tofino^{TM} based Edgecore Wedge 100BF-32X P4 switch in a leaf-spine network as a proof of concept. The evaluations were conducted for comparing the network performance of the basic forwarding [22] and RIMM. As a result, 20 events of microbursts that occurred in the basic forwarding were mitigated by RIMM and the averagely 31 packet losses in each event also were also successfully prevented. Meanwhile, in contrast to 30 to 150 retransmissions per event due to the disorder of packets in the basic forwarding, RIMM caused only 1 to 3 retransmissions per event. Moreover, for the bursty flows in 20 events, RIMM also reduced the flow-completion time (FCT) of the basic forwarding from 423.05 ms to 290.07 ms on average. The jitter time of the basic forwarding was also improved from 7.89 ms to 2.15 ms by RIMM.

The rest of the paper is organized as follows. Section II provides the background information and related works of microburst detection, packet detouring and TCP flowlet. In Section III, the architecture of the real-time in-network microburst mitigator (RIMM) is detailed, as well as its design details. In Section IV, the experiments compare the basic forwarding [22] with RIMM from different perspectives of network performance on a Tofino-based P4 switch in a leaf-spine network. Last, extended discussions and conclusions are drawn in Section V and Section VI, respectively.

II. RELATED WORKS

Before introducing the proposed ***real-time in-network microburst mitigator (RIMM)***, we shall review related research, including: (1) microburst detection, (2) packet detouring and (3) TCP flowlet.

A. DETECTING MICROBURSTS

For detecting microbursts in P4 switches, two prior works are proposed. The first work, BurstRadar [21], checks the queue depth and sees if it is greater than a threshold value. If yes, the network packet will be mirrored and copied to an external server for microburst analysis. The second work, Snappy [20], developed by Chen *et al.* is a round-robin snapshot-based approach and catches culprit flows that cause microbursts in a switch by registers. In general, Snappy is a trade-off between memory space and prediction accuracy. In contrast to BurstRadar, Snappy provides a shorter response time for detecting a microburst in a switch.

However, both works [20], [21] are only responsible for microburst detection and do not address how to mitigate microbursts. Considering the extremely short life span of a microburst, detecting and mitigating them in switches is better than dealing with them on an external server. As a result, the proposed in-network microburst mitigator (RIMM) is designed to detect a microburst in a switch and to mitigate it with the aid of one neighboring switch. Once a microburst occurs, RIMM needs no external servers and can prevent packet loss/retransmission in real time.

B. DETOURING PACKETS

Most detour-based solutions are inspired by two prior works, DIBS [23] and PABO [24]. In DIBS, packets destined to the congested port are detoured to a random neighboring port when the network congestion occurs. For simplicity, nothing needs to be tuned in DIBS, making it easy to be installed

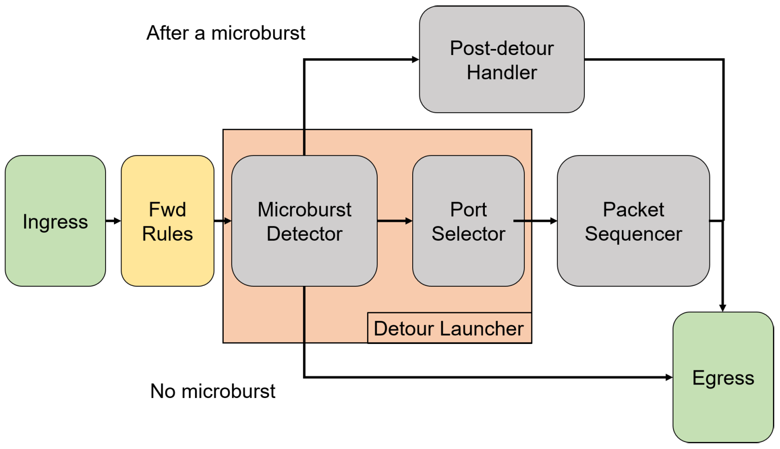


FIGURE 1. Three key components in RIMM: (1) Detour Launcher, (2) Packet Sequencer and (3) Post-detour Handler.

in a datacenter switch. However, the randomness from the selection of the detouring port may be an issue. DIBS cannot guarantee that the detour will not affect the selected neighboring port, especially when such port is also busy. Later, PABO enhances DIBS where the packet will only be detoured to the upstream ports, reducing the impact on throughput from other irrelevant flows. On the other hand, in PABO, the congestion status can eventually be sent back to the source host and the host can adjust the congestion windows, accordingly. However, PABO requires modification on both the end hosts and switches. And the probabilistic process of decision making in PABO used to decide whether to forward or to bounce a packet takes high computational effort and thus is difficult to be implemented on a physical switch.

Even if both works can effectively mitigate congestion and reduce packet loss, they have not yet minimize the disorder of packets caused by detouring. DIBS presents a strategy to provide a trade-off between packet retransmission and packet loss while PABO aims at reducing packet loss with tunable parameters through simulation. However, as mentioned earlier, packet retransmission caused by disorder of packets is also another serious problem as packet loss and may worsen the overall network performance. Therefore, the proposed RIMM also incorporates a corresponding mechanism for coping with packet reordering properly.

C. TCP FLOWLET

In general, the TCP protocol typically contains a flowlet as a burst of packets from the same flow followed by an idle interval, called a TCP flowlet. In RIMM, the TCP flowlet plays an important role in reducing packet loss. The bursty feature in the TCP protocol typically accompanies some waste of link bandwidth. Instead, some prior works utilize this feature in diverse applications. The concept of flowlet-switching is first introduced in [25] and used in multi-path routing. Because flowlet enables switching of the path for a flow, several studies built on top of flowlet were proposed, including load balancing [26]–[28] and flow scheduling [29]. Similarly, flowlet intervals in RIMM is used to batch release of detoured packets and prevents packet retransmission with the scheme of sequence numbers.

III. REAL-TIME IN-NETWORK MICROBURST MITIGATOR (RIMM)

In a modern high-speed programmable switch (e.g., Tofino^{TM} based Edgecore Wedge 100BF-32X P4 switch), the switch can monitor the queue depth of the outgoing port for every packet. Detecting a microburst by monitoring the queue depth in a switch can be done on a nanosecond scale. With the aid of packet detouring, real-time detection and mitigation can be realized on a modern high-speed programmable switch. Besides, the switch can embed more information into the pre-defined headers of packets, such as the sequence number of packets, for guaranteeing the order of packets. RIMM is mainly inspired by combining the flexibility of self-defined headers and the technique of packet detouring to achieve real-time detection and mitigation of microbursts. Figure 1 shows the three key components of RIMM: (A) **Detour Launcher**, (B) **Packet Sequencer** and (C) **Post-detour Handler**.

A. DETOUR LAUNCHER

Microburst mitigation in RIMM mainly relies on packet detouring. Detouring packets is applied to postpone the arrival of packets by using vacant queue buffers of the neighboring switch(s), so the original queue would have more time to digest the remaining packets. Otherwise, the packets are dropped if they arrive simultaneously and cause an overflow on the buffer. More specifically, ***Detour Launcher*** is composed of (1) *microburst detector* and (2) *detouring-port selector*, elaborated as follows.

1) MICROBURST DETECTOR

Several metrics (e.g., *queue depth*, *network latency* and *packet loss*) can be applied to indicate a microburst. Considering that RIMM aims at eliminating microbursts to achieve

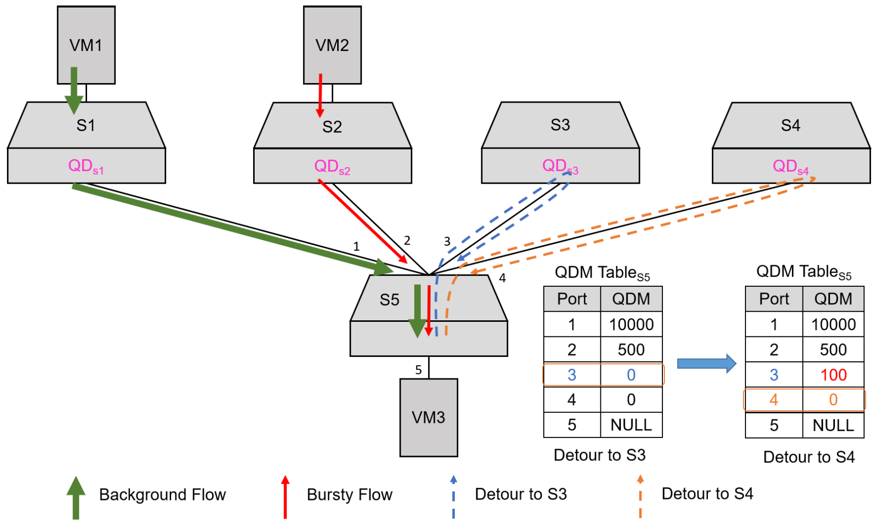


FIGURE 2. Change of detouring path According to QDM Table.

zero packet loss, the *queue depth* is chosen because it is the most straightforward and light-weighted metric that can promptly respond to microbursts among all as mentioned in [21] and [24]. Once a packet arrives at the programmable switch, the traffic rules are checked first. After a series of match-action processes, the egress port will be looked up. Later, the switch checks the corresponding queue depth for the egress port. A threshold is pre-defined for the queue depth for triggering the alert of a microburst. In Figure 1, **Microburst Detector** plays a critical role in determining if a microburst is about to start.

Based on the status of the queue depth, only one of three actions will be taken in Microburst Detector: (1) if the queue depth is less than the threshold during regular transmission, the packet is forwarded to the egress directly; (2) if the queue depth is greater than the threshold during transmission, suggesting the occurrence of a microburst, the packet is forwarded to Port Selector and packet detouring will be activated accordingly; (3) if the detouring mechanism is activated but the queue depth is less than the threshold, the packet is forwarded to Post-detour Handler (more details will be described later).

As mentioned above, Microburst Detector is responsible for signaling an alert once the queue depth of the target port exceeds the threshold value. Later, the switch starts to detour packets by modifying the egress port. However, in a modern switch design (either software or hardware), the information related to the queue status can only be accessible in the egress pipeline. That is, the action of modifying the output port needs to be done in the ingress pipeline [18]. Considering this dilemma, Microburst Detector proposes an idea to fully utilize the data plane for communication between the ingress and egress ports. As a packet arrives in the egress pipeline, the switch clones the packet header, augments the queue depth into header and sends out this message packet (to a neighboring switch). As a result, the packet is guaranteed to return the current switch because of carrying the same destination in the packet header. Once the message packet returns, the ingress pipeline can digest the queue status of the egress port. Note that the message packet does not contain the payload and carries only necessary information (e.g., the packet header and the queue status of the target egress port) to reduce communication overhead. Moreover, RIMM takes the short two-way link delay between the current and neighboring switches to respond to a microburst, whereas prior congestion-control solutions (e.g., [12], [13]) may take a long round-trip time (RTT) between two hosts to adjust the TCP sender's window size.

Additionally, if every packet is cloned and forwarded, the link bandwidth between two switches can be highly occupied by these message packets, leading to degradation of the network performance. In this regard, instead of cloning all packets and digesting the messages carried by them in the ingress, RIMM evaluates the alert condition of a microburst only at the egress before forwarding packets to the alternative port. In the egress pipeline, the latest queue depth of each port is recorded. Before cloning packets, the current switch filters the message packets first. All message packets will be dropped except those carrying the change on the queue depth of the target egress port. For instance, for a packet in the egress, if the previously recorded queue depth is less or greater than the threshold, opposite to the current queue depth, then such packet carrying meaningful message will be cloned and forwarded. As a result, these message packets effectively deliver the queue depth of the target egress port but have a negligible impact to the overall network.

2) DETOURING-PORT SELECTOR

After detecting the occurrence of a microburst on the target egress port, RIMM can restrain the rapid increase on the queue buffer because the later packets of the same flow are detoured Cuntil the queue stops to pile up. This works

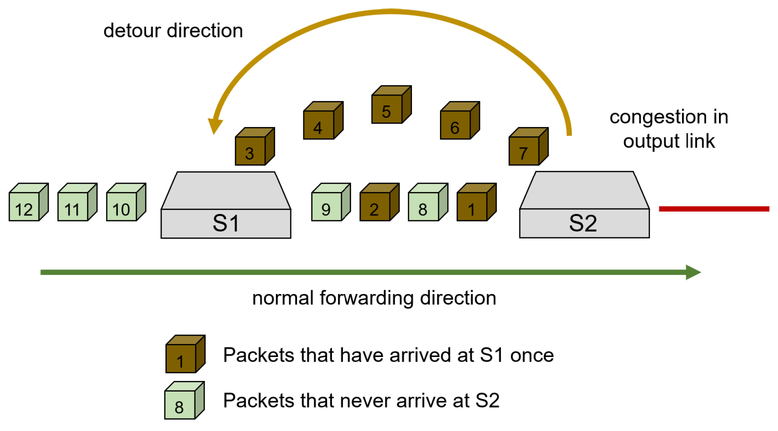

detour direction
congestion in output link
3
4
5
6
7
12
11
10
S1
9
2
8
1
S2
normal forwarding direction
Packets that have arrived at S1 once
Packets that never arrive at S2

FIGURE 3. **Disorder of packets after packet detouring in RIMM.**

effectively for the scenario of one background flow and one bursty flow where the detouring packets only consume negligible bandwidth. Nevertheless, in a real datacenter network (DCN), most switches are busy and links are highly utilized. Concurrent flows always share the network bandwidth. In addition, under different network applications, it is also possible that the queue in the next switch can also be congested as the current switch intends to detour packets backward. In this case, packet loss may not be resolved and merely moved from the current switch to the next one, called *congestion propagation* [30].

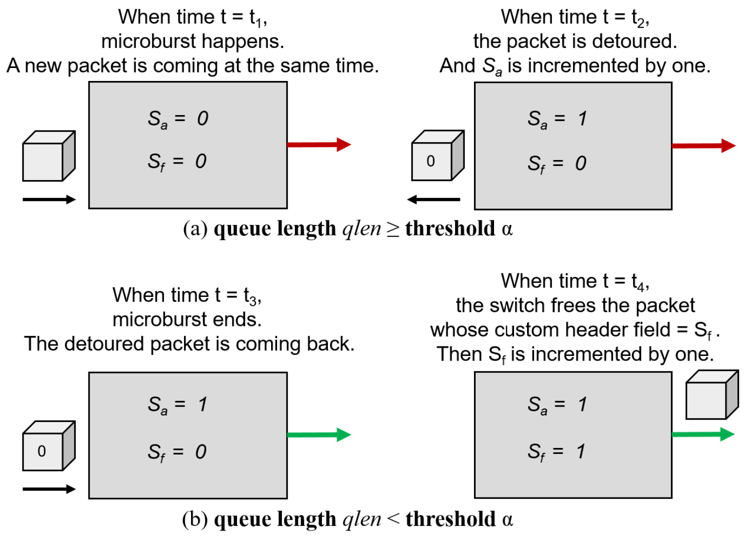
To avoid *congestion propagation*, unlike detouring packets farther as in [24], RIMM adopts a different strategy and finds the least congested port from all neighboring switches for detouring packets. As mentioned in [24], [30], detouring packets to other switches, not on the original routing path of such flow, is risky and may propagate congestion to other switches. Therefore, RIMM incorporates a Queue-Depth Monitor (QDM), which recognizes the minimum queue depth of the egress port among all neighboring switches to prevent *congestion propagation*.

Different from **Microburst Detector** to monitor the queue depth of the egress port on the current switch, QDM keeps storing the queue depths of egress ports from all neighboring switches using in-band telemetry (INT) supported by P4 switches. On each switch, the queue depth of one egress port is embedded by an extra header field of the original packets and then sent to the neighboring switch. Note that such queue depth for QDM travels through one link to the neighboring switch and is different from that in Microburst Detector, only existing in one switch. Each switch maintains a QDM table by using registers to store queue depths of all neighboring switches. As a result, the size of the QDM table is equal to the number of ports, and the port number is its index to the QDM table. If one port of the switch is connected to a neighboring switch, the QDM value of the port is set to the queue depth received from the neighboring switch. Otherwise, the QDM value is set to NULL. Once microburst occurs in the switch, Detouring-Port Selector uses the minimal one as the detouring port. Note that, since the values in QDM are dynamically changing during packet detouring, RIMM may change the detouring port to the one with the minimum queue depth in QDM.

Figure 2 shows an example of the QDM table with the queue-depth values of neighboring switches. Switch S5 connects to 4 neighboring switches (S1, S2, S3 and S4) and 1 host (VM3), and stores 5 values into its QDM table. The QDM values (QD_{s1} to QD_{s4}) for port 1 to port 4 represent the queue depths received from S1 to S4. The QDM value for port 5 is set to NULL because the port connects to host VM3. As VM1 and VM2 are sending traffic to VM3 simultaneously and trigger a microburst by Microburst Detector, S5 will first choose port 3 as the detouring port. After detouring packets from S5 to S3, QD_{s3} increases due to the detoured packets piled up in S3. After the QDM Table updates, S5 changes the detouring port from port 3 to port 4. As one can see, detoured packets sent to different neighboring switches may further cause the disorder of packets. Therefore, RIMM implements Packet Sequencer, which assigns a unique sequence number for each detouring packet to guarantee the correctness of the packet order.

B. PACKET SEQUENCER

One of the most controversial points of packet detouring is the disruption of packet orders. Whether to detour a packet or not is determined by the queue depth of the port, so packets of one flow may be detoured to different neighboring switches. Because we cannot guarantee that the packets traveling along different routing paths arrive in order, *packet reordering* is inevitable. Since the packet-detouring port in RIMM is the egress port of one neighboring switch, both old and new packets will be mixed together after packet detouring starts, leading to the even worse disorder of packets. Figure 3 shows an example where the number marked on a packet represents the order of each packet in a flow: the smaller, the earlier. As one can see, packet detouring causes the disorder of



When time t = t_1, microburst happens. A new packet is coming at the same time.
S_a = 0
S_f = 0
When time t = t_2, the packet is detoured. And S_a is incremented by one.
0
S_a = 1
S_f = 0
(a) queue length $qlen \geq$ threshold α
When time t = t_3, microburst ends. The detoured packet is coming back.
0
S_a = 1
S_f = 0
When time t = t_4, the switch frees the packet whose custom header field = S_f. Then S_f is incremented by one.
S_a = 1
S_f = 1
(b) queue length $qlen <$ threshold α

FIGURE 4. Sequence numbers before and after a microburst occurs.

packets. New and old packets can be mix up during packet detouring. Therefore, RIMM replies on Packet Sequencer to augment an additional sequence number in the packet header to maintain the order of packets. Packet Sequencer has two following types of operations:

- **queue depth** $qlen \geq$ **threshold** α
  To maintain the correct order of detoured packets, the switch prepares a pair of sequence numbers[1] for each flow: (1) next attached sequence number S_a and (2) last freed sequence number S_f. As shown in Figure 4(a), these two sequence numbers are 32-bit long and are set to 0 by default. Once packet detouring is activated, an additional 32-bit header is inserted into the detoured packet. If not yet assigned, the only field in the header copies the current value of S_a, representing its order among all detoured packets, and then S_a increments by one. If the arrival packet is already assigned with a sequence number and packet detouring is still activated, then it is detoured again without inserting the 32-bit header and copying S_a.
- **queue depth** $qlen \leq$ **threshold** α
  Under this circumstance, the switch starts to release packets in order because the target queue is no longer congested. Note that not every packet received by the switch can be released immediately; they must follow the order based on the embedded S_a values. Figure 4(b) shows an example where the switch first extracts the sequence number S_a of the processing packet, compares this S_a with current S_f maintained in the switch and frees such packet if S_a equals S_f. Then, S_f increments by one. Those packets with larger sequence numbers of the same flow will be detoured again. After the match-and-release process in Packet Sequencer, all the detoured packets will be released in the correct order.

Moreover, tens of thousands of flows may concurrently exist in a datacenter network (DCN). A physical switch with limited resources can hardly be equipped with sufficient memory for storing the status of each flow. However, the hashing technique has been widely used in recent years for the memory blow-up problem and one well-known example is SilkRoad [31]. Likewise, hashing is also applied to flows in RIMM for maintaining sequence numbers. That is, the programmable switch hashes the five-tuple value (104 bits) of a flow into a 12-bit index[2] to accommodate sequence numbers of a flow in a switch. Even if two flows happen to have the same index, it can still work as long as all detoured packets are released in the correct order by the sequence numbers.

However, if the congestion (not a microburst) lasts for a long time and makes a lot of packets which may exhaust all the 2^{32} sequence numbers, then the switch will not be able to distinguish old packets from new packets with the same sequence number. Under this circumstance, the correctness of the packet order cannot be guaranteed. However, it is rare for microbursts and requires the total data size of more than 1TB to be transmitted in less than 1 millisecond (i.e., 1 Pbps). Packet detouring will be deactivated after a period of time (typically 1 ms) if a microburst turns into congestion which must resort on other control solutions [7]–[11].

C. POST-DETOUR HANDLER

Once a microburst occurs, Detour Launcher and Packet Sequencer activate the packet detouring on the programmable

[1] In a P4 switch, stateful objects such as registers can be used to keep track of sequence numbers for packets. Since each flow needs a pair of sequence numbers, a five-tuple packet header is used for identification.

[2] The index could be even shorter, depending upon the pre-defined total number of flows.

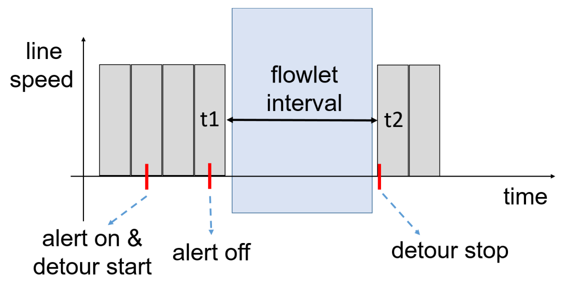


FIGURE 5. Flowlet interval to release detoured packets.

switch in real time and effectively prevents packet loss from the overflow on the queue buffer of an egress port. However, if the switch stops detouring packets as the queue depth decreases and is less than the threshold, not all detoured packets can be released immediately. There may be a number of packets still queued in neighboring switches and their sequence numbers have not yet been checked. Meanwhile, some new packets of the same flow are sent to the switch. Under these circumstances, new and old packets are mixed together, leading to the severe disorder of packets. As a result, the destination host needs to request TCP retransmissions for reordering packets, inducing degradation on network performance like packet loss.

Once the alert to the microburst turns off, there always exist some detoured packets that have not yet returned in the neighboring switch, mixing with the new packets of the same flow. This situation makes even worse disorder of packets. However, in the TCP protocol, the packets of a TCP flow are divided into multiple **flowlets** sent in a bursty way. Several prior works take advantage of the feature of flowlets to perform path switching safely. Similarly, RIMM uses the idle time (also called *interval*) between two bursts right after the alert to the microburst turns off for releasing all detoured packets before the new packets of the same flow in the next flowlet arrive. To cope with the mix of new and old packets in a flow after the alert turns off, the programmable switch records timestamps of newly coming packets for each flow to identify the start time of the next flowlet. Similar to the situation in Packet Sequencer, if a timestamp needs to be maintained per flow, the memory can be exhausted. As a result, hashing is applied again. The hash functions only need to be calculated once for packets of one flow and the hashing fields are the same as those used in Packet Sequencer.

Figure 5 illustrates the scheme of releasing detoured packets during the flowlet interval in TCP. Once a microburst is about to start, the alert is turned on (*alert on*) and packet detouring starts to perform (*detour start*). After the egress queue stops piling up and drops below the threshold, the alert to such microburst turns off (*alert off*). The programmable switch will continuously examine the time difference ($t_{diff} = t_2 - t_1$) between two consecutive newly coming packets (the boxes in grey in Figure 5). If t_{diff} is less than a pre-defined value χ, the latest packet is thought of belonging to the same flowlet as the old packets and keeps being detoured.

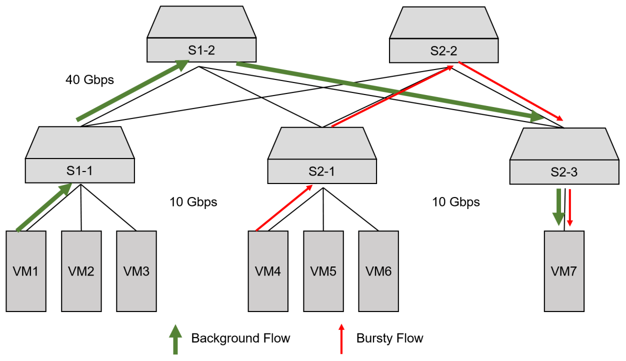


FIGURE 6. Topology for evaluation.

If t_{diff} is greater than or equal to χ, packet detouring will be deactivated (*detour stop*). All detoured packets are assumed to be entirely released during the last flowlet interval. Note that the flowlet interval may not be sufficient for long-lasting congestion. The value (χ) is pre-defined and long enough to serve as the threshold of the flowlet interval for stopping packet detouring. These flowlet intervals that last for milliseconds are typically sufficiently long to digest all detoured packets in microseconds.

IV. PROOF-OF-CONCEPT (POC) IMPLEMENTATION AND EVALUATION ON P4 SWITCH

For validating the proposed RIMM, a proof-of-concept (POC) prototype is demonstrated onto Tofino$^{\text{TM}}$ based Edgecore Wedge 100BF-32X P4 switches. All components (i.e., detour launcher, packet sequencer and post-detour handler) of RIMM are implemented through operations on match-action tables of the multi-stage TCAM (Ternary Content Addressable Memory) [32] in the forwarding pipeline of the P4 switch. Therefore, with a fixed number of TCAM accesses, RIMM takes only the constant time complexity ($O(1)$) for each incoming packet.

In our experiments, the testbed consists of two Edgecore Wedge 100BF-32X P4 Switches and 7 VM hosts. As shown in Figure 6, these two physical P4 switches are reconfigured as five logically independent P4 switches by connecting two ports on the same switch (i.e., a loop) to form a logically two-tier leaf-spine topology. The link speed of each port on P4 switches is 40 Gbps. Each link between switches is connected by a 40 Gbps DAC cable, and each link between switches and VMs is connected by 1×40 Gbps (switch side) to 4×10 Gbps (VM side) breakout cable. In our implementation, RIMM consists of 1050 lines of code in P4.

To generate a microburst, we simultaneously send short flows from multiple source hosts to the same destination host. However, how to synchronize the time of all hosts is a bigger challenge in the experimental environment. Even if we can start the transmission of these short flows simultaneously, there is no way to guarantee that these flows can arrive at the switch at the same time. In addition, using TCP flows, it is also difficult to control the growth of the queue depth in the switch due to the slow start in TCP. Hence, to effectively generate a microburst, iPerf3 is used to generate a background flow (about 9.42 Gbps from VM1 to VM7) and a bursty flow (5 MB per flow from VM4 to VM7). In our experiments, 5 MB is enough for setting a bursty flow and causes the queue overflow that only lasts for tens of microseconds (e.g., 59.1 microseconds for the measurable minimum duration) under the 9.42 Gbps background traffic.

As mentioned earlier, RIMM mainly aims at mitigating microbursts in real time by detouring packets to one neighboring switch which can both prevent packet loss and reduce packet retransmission simultaneously. To evaluate the performance of RIMM, three metrics are measured in our experiments, including packet loss, packet retransmission, and flow completion time (FCT)/Jitter. For other settings, the queue-length threshold is defined as 20000 segments (since the total size of the queue is 24000) and 20 microburst events are generated in each experiment.

A. PACKET LOSS

In a microburst event, the bursty flow causes a queue overflow at switch S2-3 for few microseconds. As a result, switch S2-3 which only applies the basic forwarding induces about 31 dropped packets on average during each microburst event. On the other hand, after running RIMM on the P4 switch, no packet loss is observed for all 20 microburst events. The result shows that RIMM can effectively avoid the occurrence of packet loss by detouring packets to neighboring switches before the queue overflow.

B. PACKET RETRANSMISSION

As discussed earlier, a detour-based approach may cause the disorder of packets. It can be resolved by packet retransmission when the TCP protocol is used for packet loss. In this experiment, we observe the number of packet retransmission caused by the disorder of packets after applying different strategies. Table 1 compares the numbers of packet retransmissions for three different strategies, including (1) the basic forwarding, (2) the detour-based forwarding and (3) the proposed RIMM, in 20 microburst events. 30 to 150 packet retransmissions can be observed under the basic forwarding. Even worse, for the detour-based forwarding, more than 1000 packet retransmissions occur due to the worse disorder of packets after detouring packets to neighboring switches. However, in RIMM, the number of packet retransmission is significantly reduced to 1 to 3. However, this number may not be zero because partial detoured packets have not yet returned from the neighboring switch until the flowlet interval finishes. One possible way to completely prevent packet retransmission is to reserve more registers for storing new incoming packets until all detoured packets are correctly released.

C. FLOW COMPLETION TIME AND NETWORK JITTERS

Flow completion time (FCT) and jitters are also used to evaluate the overall network performance of RIMM. To observe FCT and Jitters of the background flow and the bursty flow

TABLE 1. Packet retransmissions measured.

number of measured retransmissions		
basic forwarding	detour-based forwarding	RIMM
30-150	>1000	1∼3

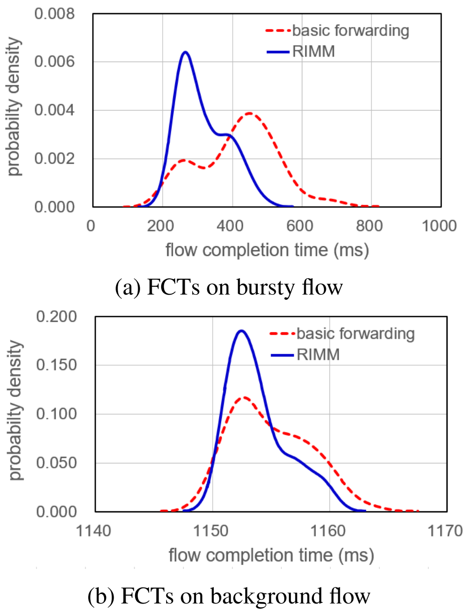


FIGURE 7. FCT comparison for basic forwarding and RIMM.

simultaneously, the duration for each microburst event is set as 1 second, in which the FCTs and jitters of the background flow and the bursty flow are recorded, respectively. Figure 7 and Figure 8 show the cumulative distribution function (CDF) of FCTs and jitters for 20 microburst events, respectively. In Figure 7, for either the bursty flow or the background flow, after applying RIMM, FCT becomes shorter even if many packets are detoured. In particular, for the bursty flow, RIMM reduces FCT from 423.05 ms to 290.07 ms on average. In Figure 8, for all 20 events on the bursty flow, RIMM also reduces jitters from 7.89 ms to 2.15 ms on average, comparing to the basic forwarding, whereas there is no much difference for the background flow. The result indicates that RIMM also effectively improves the overall network performance in FCTs and jitters.

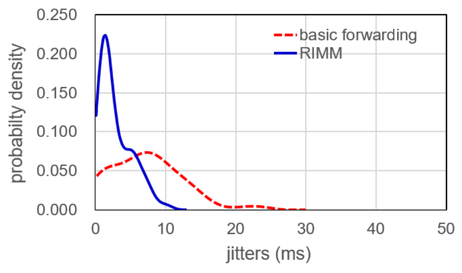
V. EXTENDED DISCUSSION

According to the evaluation from hardware proof-of-concept (POC), several issues on the experimental settings can be extended for discussion:

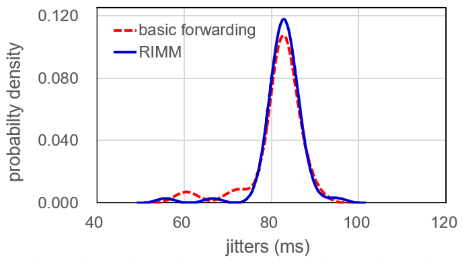
- **Comparison with Recent Prior Works**
  At present, there are several congestion control-based works such as [6], [12], [13] which are able to suppressing microbursts efficiently. However, congestion control based solutions need to modify the protocol stack of OS at the sender and/or the receiver, and only operate on TCP connections. Table 2 shows the comparison between RIMM and three recent congestion

TABLE 2. Comparison with recent prior works.

	RIMM	S-ECN [6]	MATCP [12]	AP [13]
protocol stack modification	No	Yes	Yes	Yes
non-TCP connection mitigation	Yes	No	No	No
in-network solution	Yes	No (with the help of hosts)	No (with the help of hosts)	No (with the help of hosts)
packet-loss avoidance	Yes	Yes	Yes	Yes
reaction time for mitigation	1 RTT to neighbor switch	1 RTT between hosts	1 RTT between hosts	1 RTT between hosts



(a) Jitters on bursty flow



(b) Jitters on background flow

FIGURE 8. Jitter comparison for basic forwarding and RIMM.

control-based works [6], [12], [13]. RIMM provides a major advantage in the detection of microbursts by observing the depth of the egress queue on the P4 switch without any modification of the protocol stack and without assistance from a host or controller. As opposed to host-to-host communication used in congestion control-based works, RIMM allows for a faster response time for microburst mitigation and only takes the time of a round-trip to its neighboring switch. Due to these facts, RIMM is more easily and more efficiently deployed in the network than these congestion control-based approaches.

- **Computational Complexity of RIMM**
  In RIMM, the P4 switch normally forwards packets to the destination port according to the forwarding table. Nevertheless, if a microburst occurs, Detouring-Port Selector consults QDM Table first and then related packets are detoured to a neighboring switch. In the meantime, these detoured packets will be tagged with appropriate sequence numbers to avoid packet reordering. Given a n-port switch with m entries in the forwarding table, the look-up operation is linear and has with the time complexity of O(m). More specifically, each block of Figure 1 in the packet-processing pipeline of the switch has its computational complexity analyzed separately.

  1) **Fwd Rules**: The time complexity of executing a look-up in Forwarding Table with m entries is O(m).
  2) **Microburst Detector**: The P4 switch will check whether packet detouring is activated or not, and compare *qlen* against the threshold value, leading to a computational complexity of O(1).
  3) **Port Selector**: As packet detouring is activated, the look-up operation in QDM Table selects the corresponding port for detouring the incoming packets with a computational complexity of O(n).
  4) **Packet Sequencer**: After a hash (5-tuple) operation (with the complexity of O(1)) retrieves the recorded sequence number S_a, S_a value that increment with a computational complexity of O(1).
  5) **Post-detour Handler**: Similarly, another hash (5-tuple) operation (with a complexity of O(1)) retrieves the recorded sequence number S_f, and then S_f will increment once the sequence number attached in the current packet is equal to S_f. The computational complexity of this action is O(1).

  As Figure 1 shows, the longest packet-processing time occurs when the switch needs to perform Microburst Detector, Port Selector, and Packet Sequencer on the packet during a microburst. As a result, the total complexity will be O(m) + O(1) + O(n) + O(1) = O(m+ n). Due to the fact that the look-up operations (with O(m) and O(n)) in two tables are implemented by TCAM in RIMM, the final processing time of each packet in the P4 switch can be considered constant.
- **Size of Memory Usage**
  In the POC, each queue depth and the sequence number have the lengths of 2 Bytes and 4 Bytes, respectively. For an n-port switch, RIMM maintains two queue-depth values (2 Bytes) for one port and its connecting switch (in the QMD Table). Meanwhile, for the maximum number of 4K detouring flows, two sequence numbers (4 Bytes) are also required. In summary, RIMM consumes the total size of memory usage as $n*(2 * 2$ Bytes)+4K$*$ ($2 * 4$ Bytes) = $4 * n + 32$K Bytes. All of these values are stored onto the TCAM-based register arrays in the Tofino^{TM}-based P4 switch and can be accessed in constant time.
- **Change of Queue Depth**
  In RIMM, the queue depth of the egress port is sent back to the ingress pipeline by a message packet through a neighboring switch. Once the queue depth exceeds the

TABLE 3. Monitoring queue depth of egress port.

	measured queue depth		
	mean	**standard deviation**	**maximum**
# segments	20251.5	201.2396	20728
% (on basis of α)	101.26%	1.01%	103.64%

*The alert condition of a microburst is set as $\alpha = 20000$

user-defined threshold until the ingress pipeline starts to detour packets, there remain multiple incoming packets sending to the egress queue. Therefore, the exceeded number queue depth can be regarded as the response time of Detour Launcher. In our experiment, the threshold that Microburst Detector shall signal an alert to Detour Launcher is set as 20000 segments (as the total size of the queue is 24000). Microbursts are generated and measured by the queue depth in a P4 switch 20 times. Table 3 shows that, during 20 microburst events, the queue depth stops increasing right after exceeding the threshold. The average excess of queue depth is about 1.3% of the threshold and the maximum excess is 3.4%. The standard deviation is 1.01%. The results indicate that RIMM can effectively mitigate microbursts in microseconds.

- **Length of Flowlet Interval**
  RIMM cannot guarantee that an arbitrary flowlet interval is long enough for releasing all detoured packets. As a matter of fact, a more reliable way is to check if S_a equals S_f on the current switch. Since there are always enough flowlet intervals between TCP bursts to release detoured packets, the current switch only needs to determine the time (i.e., S_a equals S_f) for deactivating packet detouring. However, simultaneous access to the same register twice for managing S_a and S_f is not allowed in the Tofino-based Edgecore Wedge 100BF-32X P4 switch. As a result, a long-enough value (χ) is predefined to serve as the threshold of the flowlet interval for stopping packet detouring. However, for microbursts that last for $<$ 1 millisecond, these flowlet intervals that span a few milliseconds are sufficient for releasing all detoured packets.
- **Impact of Hash Collision**
  Any work that applies the hashing technique needs to cope with collision. However, in our application, collision scarcely occurs because Packet Sequencer only works if an alert to a microburst is signaled. Therefore, in normal circumstances, the switch need not maintain sequence numbers. Even if a collision occurs in RIMM and causes multiple flows to share the same pair of sequence numbers, the respective order of detoured packets for each flow remains intact. Accordingly, the correctness of the packet order can be guaranteed on each flow. Therefore, hashing is effective in preventing the problem of memory blowup in RIMM, and its potential collision has no impact on the packet order.
- **Possibility of Microburst Diffusion**
  In RIMM, the programmable switch chooses one egress port for packet detouring. During detouring packets, packets may disperse to multiple, not only one, neighboring switches. Whenever a packet comes in the switch, the QDM table keeps updating the stored values. Since RIMM always chooses the detouring port with the smallest value in the QDM table for detouring packets, the queue depth of the corresponding neighboring switch increases. Once the increasing value in the QDM table is no longer the smallest one, RIMM chooses another egress port for packet detouring. Considering that the lifespan of a microburst is about a few microseconds and packets take turns to be detoured to different neighboring switches, unlike congestion propagation [30], it is difficult for RIMM to diffuse a microburst to another neighboring switch. Note that RIMM is designed for preventing packet loss and retransmission from microbursts. If packet detouring lasts longer than a user-defined period of time, it refers to the occurrence of network congestion. RIMM will automatically release all detoured packets and then terminate. Other congestion control should be activated, accordingly.

VI. CONCLUSION

Microbursts that last in microseconds have captured much attention in the research of datacenter networks (DCN). However, prior congestion-control works are too coarse-grained and thus cannot be used for microbursts in real time. As a result, in this paper, a real-time microburst mitigator, called **RIMM**, is proposed and mainly applies packet detouring as well as flowlet intervals to reduce packet loss and packet retransmission in the network without any intervention of the SDN controller or any modification on sendsers' TCP protocol stack. RIMM consists of three core components: (1) **detour launcher** detects a microburst in real time by monitoring the queue depth for the egress port and launches packet detouring promptly to avoid queue overflows; (2) **packet sequencer** facilitates the switch to release detoured packets in the correct order with a pair of sequence numbers; (3) **post-detour handler** releases new packets interleaved with old ones of the target flow in order after such microburst is mitigated. Furthermore, RIMM is implemented onto Tofino^{TM}-based programmable switches with P4/INT and entirely works in the network. The experiment results show that, during 20 microburst events, RIMM can all achieve zero packet loss with only 1 to 3 retransmissions per event. Moreover, RIMM also reduces the flow completion time (FCT) of the basic forwarding from 423.05 ms to 290.07 ms on average. The jitter time of the basic forwarding was also improved from 7.89 ms to 2.15 ms by RIMM. To the best of our knowledge, RIMM is the first in-network solution, capable of mitigating microbursts in real time for the datacenter network (DCN).

ACKNOWLEDGMENT

The authors would like to greatly thank Intel/Barefoot, Inventec Corporation, and Edgecore Networks for the support of this research work.

REFERENCES

[1] S. Sen, J. L. Rexford, J. K. Dey, J. F. Kurose, and D. F. Towsley, "Online smoothing of variable-bit-rate streaming video," *IEEE Trans. Multimedia*, vol. 2, no. 1, pp. 37–48, Mar. 2000.

[2] J. Teall. (Mar. 2018). *Financial Trading and Investing 2nd Edition*. [Online]. Available: https://www.elsevier.com/books/financial-trading-and-investing/teall/978-0-12-811116-1

[3] Cisco Systems, Inc. (Apr. 2021). *Cisco Nexus 9000 Series NX-OS Quality of Service Configuration Guide*. [Online]. Available: https://www.cisco.com

[4] Huawei Technologies Co., Ltd. (Nov. 2020). *What is a Microburst? How to Detect a Microburst*. [Online]. Available: https://support.huawei.com/enterprise/en/doc/EDOC1100086962

[5] M. Allman and E. Blanton, "Notes on burst mitigation for transport protocols," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 35, no. 2, pp. 53–60, Apr. 2005.

[6] D. Shan, F. Ren, P. Cheng, R. Shu, and C. Guo, "Micro-burst in data centers: Observations, analysis, and mitigations," in *Proc. IEEE ICNP*, Sep. 2018, pp. 88–98.

[7] M. Alizadeh, A. Greenberg, D. A. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta, and M. Sridharan, "Data center TCP (DCTCP)," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 40, no. 4, pp. 63–74, Oct. 2010.

[8] S. Kunniyur and R. Srikant, "End-to-end congestion control schemes: Utility functions, random losses and ECN marks," *IEEE/ACM Trans. Netw.*, vol. 11, no. 5, pp. 689–702, Oct. 2003.

[9] R. Mittal, V. T. Lam, N. Dukkipati, E. Blem, H. Wassel, M. Ghobadi, A. Vahdat, Y. Wang, D. Wetherall, and D. Zats, "TIMELY: RTT-based congestion control for the datacenter," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 45, pp. 537–550, Aug. 2015.

[10] D. Shan and F. Ren, "Improving ECN marking scheme with micro-burst traffic in data center networks," in *Proc. INFOCOM*, Atlanta, GA, USA, May 2017, pp. 1–9.

[11] H. Wu, J. Ju, G. Lu, C. Guo, Y. Xiong, and Y. Zhang, "Tuning ECN for data center networks," in *Proc. 8th Int. Conf. Emerg. Netw. Exp. Technol.*, Dec. 2012, pp. 25–36.

[12] D. Shan, F. Ren, P. Cheng, R. Shu, and C. Guo, "Observing and mitigating micro-burst traffic in data center networks," *IEEE/ACM Trans. Netw.*, vol. 28, no. 1, pp. 98–111, Feb. 2020.

[13] S. Zou, J. Huang, J. Wang, and T. He, "Flow-aware adaptive pacing to mitigate TCP incast in data center networks," *IEEE/ACM Trans. Netw.*, vol. 29, no. 1, pp. 134–147, Feb. 2021.

[14] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese, and D. Walker, "P4: Programming protocol-independent packet processors," *SIGCOMM Comput. Commun. Rev.*, vol. 44, pp. 87–95, Jul. 2014.

[15] Open Networking Foundation. *Software-Defined Networking (SDN)*. Accessed: Jun. 15, 2021. [Online]. Available: https://opennetworking.org/sdn-definition/

[16] Open Networking Foundation. *OpenFlow Switch Specification v1.3.0*. Accessed: Jun. 15, 2021. [Online]. Available: https://opennetworking.org/wp-content/uploads/2014/10/openflow-spec-v1.3.0.pdf

[17] T. P. L. Consortium. (Nov. 2018). *P416 Language Specification*. [Online]. Available: https://p4.org/p4-spec/docs/P4-16-v1.1.0-spec.pdf

[18] T. P. L. Consortium. (Nov. 2018). *P416 Portable Switch Architecture*. [Online]. Available: https://p4.org/p4-spec/docs/PSA-v1.1.0.pdf

[19] Edgecore Networks. *Wedge 100BF-32X Switch*. [Online]. Available: https://www.edge-core.com/

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

[21] R. Joshi, T. Qu, M. C. Chan, B. Leong, and B. T. Loo, "BurstRadar: Practical real-time microburst monitoring for datacenter networks," in *Proc. 9th Asia–Pacific Workshop Syst.*, Aug. 2018, pp. 1–8.

[22] Open Networking Foundation. *P4 Tutorial-Basic Forwarding*. Accessed: Jun. 15, 2021. [Online]. Available: https://github.com/p4lang/tutorials/tree/master/exercises/basic

[23] K. Zarifis, R. Miao, M. Calder, E. Katz-Bassett, M. Yu, and J. Padhye, "DIBS: Just-in-time congestion mitigation for data centers," in *Proc. 9th Eur. Conf. Comput. Syst. (EuroSys)*, 2014, pp. 1–14.

[24] X. Shi, L. Wang, F. Zhang, K. Zheng, and Z. Liu, "PABO: Congestion mitigation via packet bounce," in *Proc. IEEE Int. Conf. Commun. (ICC)*, May 2017, pp. 1–6.

[25] S. Sinha, S. Kandula, and D. Katabi, "Harnessing TCPs burstiness using flowlet switching," in *Proc. SIGCOMM Workshop Hot Topics Netw.*, Nov. 2004. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.100.5390

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

[27] M. Alizadeh, T. Edsall, S. Dharmapurikar, R. Vaidyanathan, K. Chu, A. Fingerhut, V. T. Lam, F. Matus, R. Pan, N. Yadav, and G. Varghese, "CONGA: Distributed congestion-aware load balancing for datacenters," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 4, pp. 503–514, 2014.

[28] 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.

[29] M. Al-Fares, S. Radhakrishnan, B. Raghavan, N. Huang, and A. Vahdat, "Hedera: Dynamic flow scheduling for data center networks," in *Proc. 7th USENIX Conf. Netw. Syst. Design Implement., (NSDI)*, 2010, pp. 89–92.

[30] D. E. Newman, N. D. Sizemore, B. A. Carreras, and V. E. Lynch, "Growth and propagation of disturbances in a communication network model," in *Proc. 35th Annu. Hawaii Int. Conf. Syst. Sci.*, 2002, pp. 867–874.

[31] R. Miao, H. Zeng, C. Kim, J. Lee, and M. Yu, "SilkRoad: Making stateful layer-4 load balancing fast and cheap using switching ASICs," in *Proc. ACM SIGCOMM*, Aug. 2017, pp. 15–28.

[32] F. Zane, G. Narlikar, and A. Basu, "Coolcams: Power-efficient TCAMs for forwarding engines," in *Proc. IEEE INFOCOM 22nd Annu. Joint Conf. IEEE Comput. Commun. Societies*, Mar. 2003, pp. 42–52.

YU-JIE LIN received the B.S. and M.S. degrees in electrical and computer engineering from the National Yang Ming Chiao Tung University, in 2017 and 2019, respectively. He has been a Software Engineer at A10 Networks, Taiwan, since 2019, working on network dataplane L4 to L7 layer development. His research interests include software-defined networking, programmable switch, and congestion control.

CHI-HSIANG HUNG (Member, IEEE) received the Ph.D. degree from the Graduate School of Engineering Science and Technology, National Yunlin University of Science and Technology, in 2016. He is currently a Postdoctoral Research Fellow at the Advanced SDN Technology Research Center, National Yang Ming Chiao Tung University, Hsinchu, Taiwan. His research interests include software-defined networking, programmable switch, mobile edge computing, and network security.

CHARLES H.-P. WEN (Member, IEEE) received the Ph.D. degree in verification and test of very large scale integration (VLSI) designs from the University of California at Santa Barbara, Santa Barbara, CA, USA, in 2007. Since 2020, he has been a Distinguished Professor with the National Yang Ming Chiao Tung University, Hsinchu, Taiwan, where he is also a specialist in computer engineering. Over the past few years, his research has been focused on applying data mining and machine learning techniques to system-on-chip designs (including radiation hardening, functional verification, and timing analysis) and cloud networking (especially on performance analysis and architecture design of large-scale data centers).

• • •