INT-assisted Adaptive Packet Scheduling in PDP Switches for End-to-end Latency Control

Zichen Xu, Xiaoliang Chen, and Zuqing Zhu†
School of Information Science and Technology, University of Science and Technology of China, Hefei, China
†Email: SA20006080@mail.ustc.edu.cn, xlichen@ieee.org, zqzhu@ieee.org†

***Abstract*—In today's Internet, ensuring the end-to-end (E2E) latency of delay-sensitive flows is becoming increasingly challenging due to the rapid growth of network services, users and traffic volume. In this work, we propose a packet scheduling scheme that leverages in-band network telemetry (INT) to catch packets' forwarding status in realtime and realizes fine-grained and adaptive packet scheduling accordingly for E2E latency control, and implement it in P4-based programmable data plane switches (PDP-SWs). Specifically, we use INT to make a packet convey its own realtime status (*e.g.*, experienced latency) as it traverses the network, and program PDP-SWs in the network to schedule the packet with the weighted round-robin (WRR) approach based on the realtime status, such that the packet's quality-of-service (QoS) demand on E2E latency can be satisfied in the most effective way. We design the packet processing procedure in PDP-SW, propose an effective packet scheduling algorithm, and optimize the algorithm to ensure that it can be realized with P4 programs. The proposed scheme is then implemented and evaluated in a real-world network testbed built with hardware PDP-SWs. Experimental results confirm the effectiveness of our proposal and suggest that it achieves better control of E2E latency than a few existing benchmarks.**

***Index Terms*—In-band network telemetry, P4, Weighted round-robin scheduling, End-to-end latency control.**

I. Introduction

Over the past decade, network services, users and traffic volume in the Internet have grown explosively due to the fast development of data-centers (DCs) [1–4] and 5G networks [5–7], putting great pressure on satisfying the quality-of-service (QoS) demands on end-to-end (E2E) latency, especially for emerging delay-sensitive applications such as virtual reality, autonomous driving, and distributed machine learning [8]. Moreover, the Internet infrastructure is undergoing dramatic changes with wide deployment of new networking technologies (*e.g.*, software-defined networking (SDN), virtual network slicing, and network function virtualization (NFV)), making networks more and more dynamic and complex [9–12]. These challenges have promoted operators to continuously look for mechanisms that can control E2E latency of delay-sensitive flows in dynamic and complex network environments.

The E2E latency of flows can be controlled with the control plane, data plane and coordination of the two planes. The control plane based approaches conduct network monitoring with polling-based schemes (*e.g.*, SNMP [13], sFlow [14], and Netflow [15]) and invoke traffic engineering (TE) to reroute the delay-sensitive flows whose E2E latency violates their QoS requirements. However, the response speed of these approaches depends on their polling periods, making it difficult for them to react to sudden network changes timely. In the data plane, the E2E latency of flows can be managed by either end-hosts or switches/routers. As for end-hosts, people have developed congestion control mechanisms such as TCP [16] and DTCP [17], which let source end-host reduce its sending rate when abnormally-long E2E latency occurs. Nevertheless, the response speed is limited by the round-trip time of a flow and reducing its sending rate might also affect the QoS. Switches/routers control the E2E latency of flows with packet scheduling, and can respond to sudden network changes almost instantly. However, how to assign priorities to packets for enforcing E2E-latency-guaranteed scheduling in dynamic and complex network environments is still an open question, *i.e.*, it is challenging to realize fine-grained and adaptive packet scheduling for ensuring QoS demands on E2E latency [18].

The programmable data plane (PDP) [19] has provided efficient ways of controlling E2E latency with the coordination of control and data planes. For instance, in-band network telemetry (INT) [20] can be realized with PDP switches (PDP-SWs) to achieve fine-grained and realtime network monitoring for each flow. INT lets each PDP-SW encode the selected network status that a packet sees when passing through the PDP-SW as telemetry data in the packet, then as the packet traverses the network, it accumulates telemetry data hop by hop, and finally all the telemetry data is extracted and forwarded to the control plane at the egress PDP-SW. Therefore, the control plane can invoke TE much more timely when abnormally-long E2E latency happens for a flow [21, 22]. Nevertheless, as these E2E latency control schemes still count on the control plane to make TE decisions, their response speed is still slow, making it hard to respond to sudden network changes on time.

To further increase the response speed of INT-based E2E latency control, researchers have tried to realize instant packet scheduling in PDP-SWs based on telemetry data collected by INT [23–27]. The studies in [23, 24] leveraged multiple first-in-first-out (FIFO) queues in a PDP-SW and realized strict priority (SP) based packet scheduling with them, while the priority of each packet was determined based on the telemetry data encoded in it. Similarly, the authors of [25–27] used single or multiple FIFO queues together with SP-based packet scheduling to simulate a push-in-first-out (PIFO) queue for latency control. Although these approaches did leverage INT and instant packet scheduling to send out the packets whose demands on E2E latency are more stringent earlier, they still

bear the following issues, which prevent them from controlling the E2E latency of each delay-sensitive flow precisely.

First, they only schedule a packet with more stringent E2E latency requirement to a higher-priority queue, but do not provide a mechanism that can select the most appropriate queue for a packet according to its actual demand on E2E latency. Second, some of them (*e.g.*, those in [26, 27]) leveraged recirculated packets to convey the lengths of output queues in each PDP-SW, but as recirculation takes time, the obtained queue lengths might not be accurate, leading to incorrect packet scheduling. Third, SP-based packet scheduling has the intrinsic drawback of starving low-priority packets when the load of high-priority packets is high, and since the memory space in each PDP-SW is limited, it would be impossible to store all the E2E-latency-constrained packets in the highest-priority queue, when incoming traffic approaches to the level of congestion. Hence, although SP-based packet scheduling can shorten the average flow completion time [25, 27, 28], it might not always satisfy the E2E latency requirements of packets. Lastly but most importantly, an E2E-latency-constrained packet might need to go through multiple hops to reach its destination, but these packet scheduling schemes only consider local optimization, and thus they might not be able to ensure the E2E latency of delay-sensitive flows accurately.

In this work, we propose an INT-assisted adaptive packet scheduling scheme, which leverages INT to convey network status (*e.g.*, queue lengths) to PDP-SWs and lets them perform packet scheduling based on the remaining hops of each packet accordingly, for precise E2E latency control, and implement our proposal (namely, remaining-hop-aware (RHA) packet scheduling) in P4-based hardware PDP-SWs with Tofino ASICs. Specifically, our RHA scheduling scheme obtains real-time network status with INT and uses weighted round-robin (WRR) scheduling to avoid the starvation of packets caused by the SP approach. Similar to existing approaches, we also leverage packet recirculation to obtain queue lengths at each egress port, but to improve the accuracy of E2E latency control for delay-sensitive flows, we derive the mapping relation between the obtained queue lengths and estimated queuing delay to help each PDP-SW select a proper queue for each packet. Experimental results verify that our proposal can control the E2E latency of flows better than a few existing benchmarks.

Our major contributions can be summarized as follows:

- We explore the relation between the obtained queue lengths and estimated queuing delay to enable hardware PDP-SWs to perform adaptive packet scheduling for satisfying the E2E latency requirement of each packet.
- We improve the packet recirculation for getting queue lengths at each egress port of a PDP-SW, by mitigating the impact of recirculation delay to make obtained queue lengths work more accurately in the ingress pipeline.
- We utilize WRR scheduling approach to avoid starvation of packets, and propose a RHA packet scheduling scheme for precise E2E latency control of delay-sensitive flows.
- We build a small-scale but realistic network testbed to demonstrate our proposal experimentally, and verify its advantages over existing benchmarks.

The rest of the paper is organized as follows. Section II surveys the related work briefly. We design our proposed scheme in Section III. The experimental demonstrations are shown in Section IV. Finally, Section V summarizes the paper.

II. Related Work

Since network congestion can severely affect E2E latency of flows, host-based congestion control mechanisms such as those in [16, 17] were the earliest ways of E2E latency control. However, the response speed of these host-based schemes is limited by the round-trip time of each flow and they might not react to network changes before congestion actually happens to cause packet losses. To improve the response speed of E2E latency control such that sudden network changes can be handled instantly, we need to realize packet scheduling in switches/routers with realtime monitoring, which can be facilitated by INT [20]. People have implemented INT with P4 [29] and POF [30], and deployed them to monitor various networks [31, 32]. Most of the existing INT-assisted E2E latency control schemes still involve the control plane (*e.g.*, in [22]), and thus cannot react to network changes instantly.

The studies in [23, 24] realized instant packet scheduling in PDP-SWs based on the telemetry data collected by INT, but as they use SP-based packet scheduling, the starvation of packets cannot be avoided. The authors of [23] programmed each PDP-SW to make scheduling decisions on packets based on their forwarding status so far, without considering the status of the current hop, while the work in [24] simply lets a PDP-SW send a packet to the highest priority queue if it finds out that the queue to which the packet was originally planned to go has abnormally-long queuing delay. On the other hand, the studies in [25–27] proposed to simulate a PIFO queue and implement various scheduling algorithms based on it in PDP-SWs. To save hardware resources, Yu *et al.* [26] only leveraged one FIFO queue for packet scheduling, which might lead to unexpected packet losses, and the issue was relieved in [25, 27] after considering multiple FIFO queues. However, these studies focused on realizing existing scheduling algorithms in PDP-SWs, but have not addressed how to optimize the scheduling algorithms for precise E2E latency control.

In addition to the scheduling algorithms considered in [25–27], there are more sophisticated ones in the literature, such as the shortest remaining processing time first (SRPT) [28], start time fair queuing [33], and quick fair queuing [34], but it is difficult for them to be implemented in P4-based PDP-SWs due to hardware limitations. RPQ [35] proposed to leverage WRR-based scheduling for delay-sensitive flows, but it did not consider the packet scheduling for precise E2E latency control.

III. System Design

In this section, we will first give an overview of our proposed system, and then describe its design in detail to explain its principle of ensuring E2E latency of packets.

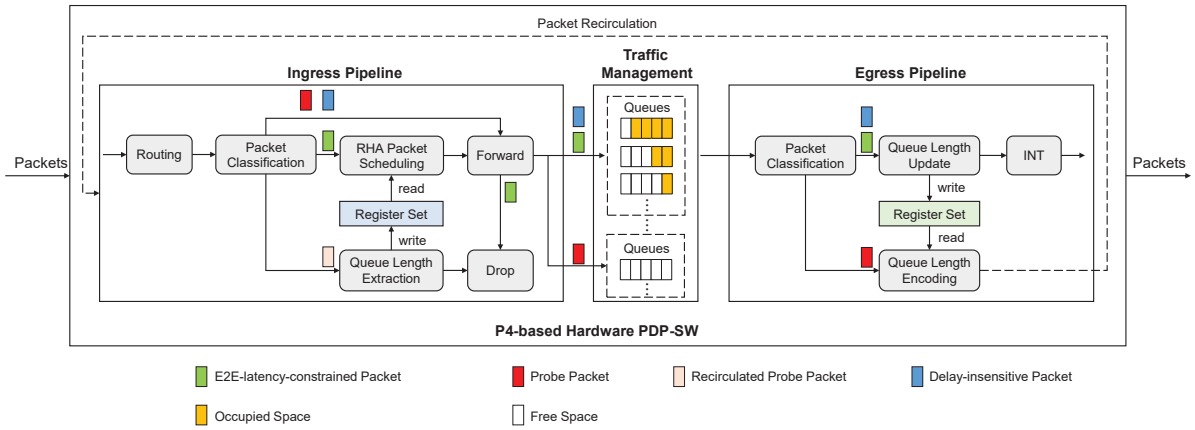


Fig. 1. System overview of INT-assisted RHA packet scheduling for E2E latency control.

A. Design Overview

Fig. 1 shows the overview of the proposed INT-assisted RHA packet scheduling system, which is designed based on a hardware PDP-SW with Tofino ASICs for E2E latency control. To facilitate RHA scheduling, we categorize packets into four types, *i.e.*, the probe packets, recirculated probe packets, E2E-latency-constrained packets, and delay-insensitive packets, where the delay-sensitive and delay-insensitive packets are from active applications in the network, the probe packets are generated according to instructions from the control plane for realizing queue length measurements in realtime, and the recirculated probe packets are probe packets that have experienced recirculation in PDP-SWs and thus carry information about queue lengths. We design modules in a PDP-SW to process the four types of packets and allocate registers to store the obtained queue lengths in both ingress and egress pipelines.

When a packet arrives, the PDP-SW parses it and sends it to the ingress pipeline. In the ingress pipeline, the packet passes through the routing and packet classification modules in sequence, where the former determines its egress port and the latter figures out its type based on the hashed 5-tuple. If it is a probe packet, we set its egress port as the dedicated recirculation port. If it is an E2E-latency-constrained packet, we process it by the RHA packet scheduling module, which selects an appropriate queue for it based on the status data encoded in its INT fields, its remaining hops stored in the PDP-SW, and the lengths of the queues at its egress port (stored in registers), or drops it if an appropriate queue cannot be found. For a recirculated probe packet, the queue length extraction module gets queue lengths at an egress port from it to store in registers, before dropping it. Finally, if it is a delay-insensitive packet, we determine its egress port according to its flow entry stored in the PDP-SW. Next, the packets are sent to the traffic management module that contains the queues dedicated to each egress port. As the recirculation port is dedicated for packet recirculation, only probe packets will be sent to it, and thus probe packets will not share any queues with E2E-latency-constrained or delay-insensitive packets.

In the egress pipeline, the packet classification module forwards probe packets to the queue length encoding module and sends E2E-latency-constrained and delay-insensitive packets to the queue length update module. Here, we introduce the probe packets to solve the problem that in a Tofino-based PDP-SW, the queue lengths at egress ports cannot be obtained directly in the ingress pipeline [26, 27]. Then, the queue length encoding module modifies the actual queue lengths to compensate the impact of recirculation delay (the principle of the queue length modification will be introduced in Section III-D), and encodes the results in probe packets, which will then bring the queue lengths to the ingress pipeline after recirculation. As for the E2E-latency-constrained and delay-insensitive packets, the queue length update module obtains the current lengths of the queues that they just experienced and updates the corresponding registers, and then the INT module inserts the selected status data in them as INT fields.

B. Selection of Queue Type and Scheduling Scheme

FIFO queue is one of the simplest queues, and as shown in Fig. 2(a) (the numbers represent the priorities of the packets), it dequeues packets according to their arrivals in sequence, regardless of their priorities. On the other hand, PIFO queue can change the positions of packets in it according to their priorities, ensuring higher-priority packets to be sent out earlier (as shown in Fig. 2(b)). However, it is difficult to realize a PIFO queue in a Tofino-based PDP-SW, because it does not support moving packets in a queue dynamically.

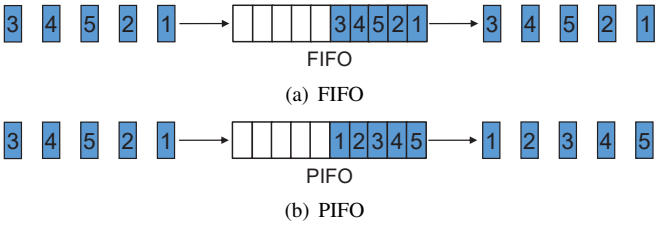
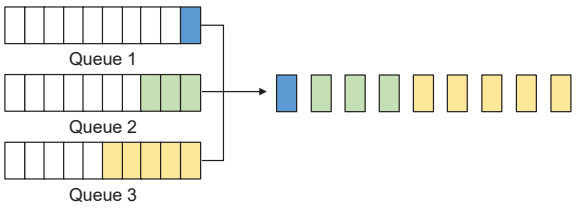
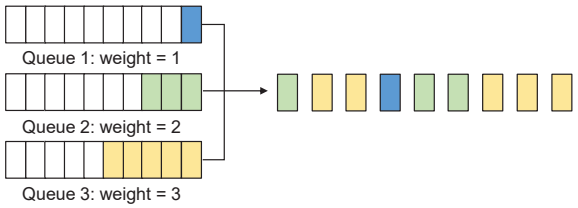


Fig. 2. Working principle of queues.

Although PIFO queues can not be realized directly in a Tofino-based PDP-SW, we can emulate one by combining



(a) SP scheduling



(b) WRR scheduling

Fig. 3. Principle of SP and WRR scheduling schemes.

multiple FIFO queues and assigning a priority to each of them (as shown in Fig. 3(a)), *i.e.*, applying the SP scheduling to the FIFO queues [25, 27]. However, SP scheduling can lead to the starvation of packets in a low-priority queue. This issue can be resolved by the WRR scheduling, and as shown in Fig. 3(b), we assign a weight to each FIFO queue and determine the lower-bound of dequeuing rate R_i of the i-th queue as

$$R_i = \frac{w_i}{\sum_{j=1}^{N} w_j} \cdot R, \tag{1}$$

where w_i is the weight of the i-th queue, N is the number of queues, and R is data-rate of the egress port. Hence, WRR scheduling provides a higher dequeuing rate to a queue whose weight is higher, but will not starve any of the queues. Note that, R_i is just the lower-bound, and a queue's actual dequeuing rate will be higher if any of the other queues are idle. Considering these benefits, we use multiple FIFO queues and WRR scheduling to design our RHA scheduling scheme.

C. Packet Status Tracking

We leverage INT to track the forwarding status of each E2E-latency-constrained packet in realtime, and the packet format is shown in Fig. 4, which is adapted from the standard INT packet format [20] with only minor modifications. Specifically, we insert an INT header in between the TCP/UDP header and payload, and leverage the "protocol" field in the IP header to indicate that a packet contains the INT header (*i.e.*, it is an INT packet). Specifically, the field is modified to *0xfe* or *0xff* by the packet's ingress PDP-SW if it is a TCP or UDP packet, respectively, and will be restored to its original value by the egress PDP-SW. As shown in Fig. 4, each INT header contains fields of *Hop Count* and *Total Queue Time* followed by a number of *INT Fields*, where *Hop Count* is a one-byte field that tells the number of hops that the packet has experienced, *Total Queue Time* stores the total queuing delay that the packet has experienced in a 6-byte field, and they are updated by the

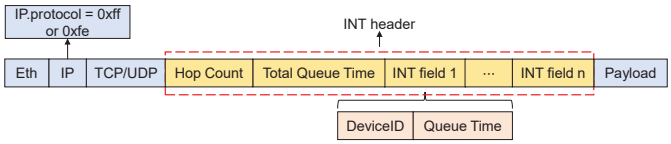


Fig. 4. Format of INT packets.

INT module in the egress pipeline of a PDP-SW (as shown in Fig. 1). Each *INT Field* contains two 4-byte subfields, *i.e.*, *DeviceID* and *Queue Time*, where *DeviceID* stores the unique ID of the PDP-SW that encodes the *INT Field*, and *Queue Time* is the queuing delay of the packet in the PDP-SW.

When a PDP-SW receives an E2E-latency-constrained packet whose unique index is i, it first extracts the value of *Total Queue Time* to t_i, and then the remaining E2E tolerable queuing delay (TQD) T_i of the packet can be calculated as

$$T_i = \tau_i - t_i, \tag{2}$$

where τ_i is the total E2E TQD calculated by excluding E2E transmission delay, tested when no congestion occurs, from the packet's required E2E latency. The required E2E latency is set based on the service-level agreement (SLA) of the packet's application, while the E2E transmission delay can be obtained in advance by leveraging probe packets. T_i will be used for the subsequent RHA packet scheduling.

D. Estimation of Queuing Delay

To ensure the E2E latency of a packet, we have to estimate the queuing delay at each hop, which is the major challenge of designing our RHA packet scheduling scheme. We try to explore the relation between the length and queuing delay of each queue in a PDP-SW such that the queuing delay can be estimated approximately. When a queue does not approach to congestion, its length would be close to zero, and thus packets almost experience zero queuing delay in it. On the other hand, when a queue approaches to congestion, its length and queuing delay increase simultaneously. As we use WRR scheduling, the queuing delay T_{q_i} of the i-th queue q_i can be estimated based on the queue length Q_i as follows if we assume that the queuing delay changes linearly with the queue length

$$T_{q_i} = \frac{R}{r_i} \cdot k \cdot Q_i, \tag{3}$$

where r_i is the dequeuing rate and k is the constant coefficient that reflects the linear relation between the queuing delay and queue length when the dequeuing rate equals the data-rate of the egress port (R). Although the dequeuing rate r_i can change dynamically, we can get its lower and upper bounds with Eq. (1) and the data-rate of the egress port, respectively, *i.e.*, $r_i \in [R_i, R]$. Then, the queuing delay falls in $T_{q_i} \in [k \cdot Q_i, \frac{R}{R_i} \cdot k \cdot Q_i]$, *i.e.*, it can be derived with the queue length.

Although the length of each queue can be obtained in the egress pipeline of a PDP-SW, we need to estimate its queuing delay before scheduling an E2E-latency-constrained packet, which is performed in the ingress pipeline. Therefore, recirculation of probe packets has to be used to bring the queue

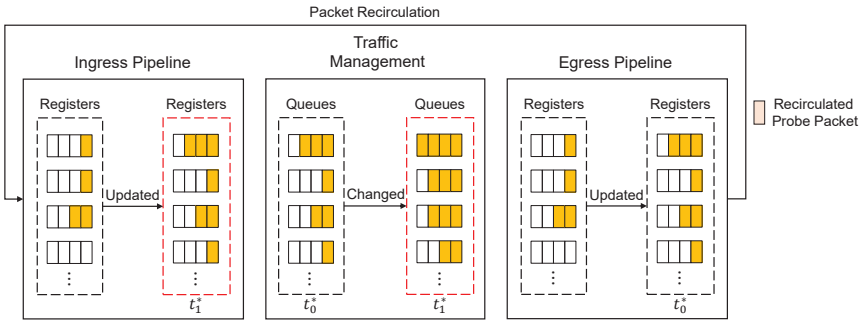


Fig. 5. Example on inaccurate queue lengths caused by packet recirculation.

Match	Action	Action Data
Queue 1: $[Q_1^{(1)}, Q_2^{(1)})$: range Queue 2: $[Q_1^{(2)}, Q_2^{(2)})$: range ⋮ Queue N: $[Q_1^{(n)}, Q_2^{(n)})$: range	modify_queue_depth()	$c_1^{(1)}, c_2^{(1)}, \cdots, c_n^{(1)}$
Queue 1: $[Q_3^{(1)}, Q_4^{(1)})$: range Queue 2: $[Q_3^{(2)}, Q_4^{(2)})$: range ⋮ Queue N: $[Q_3^{(n)}, Q_4^{(n)})$: range	modify_queue_depth()	$c_1^{(2)}, c_2^{(2)}, \cdots, c_n^{(2)}$
⋮	⋮	⋮
Queue 1: $[Q_{2n-1}^{(1)}, Q_{2n}^{(1)})$: range Queue 2: $[Q_{2n-1}^{(2)}, Q_{2n}^{(2)})$: range ⋮ Queue N: $[Q_{2n-1}^{(n)}, Q_{2n}^{(n)})$: range	modify_queue_depth()	$c_1^{(n)}, c_2^{(n)}, \cdots, c_n^{(n)}$
Default	No_Action	

Fig. 6. MAT for improving packet recirculation.

lengths obtained in the egress pipeline to the ingress pipeline (as shown in Fig. 1), but the delay of each recirculation will degrade the accuracy of the queue lengths inevitably. Specifically, as shown in Fig. 5, after E2E-latency-constrained or delay-insensitive packets have updated the queue lengths stored in registers in the egress pipeline, a probe packet will carry the updated queue lengths to the ingress pipeline at t_0^*. Due to its recirculation delay, the recirculated probe packet arrives and updates the queue lengths stored in the ingress pipeline at t_1^* ($t_1^* > t_0^*$). However, the actual queue lengths might have already changed at t_1^*, making the ingress pipeline schedule packets with inaccurate queue lengths.

Therefore, we need to modify the queue lengths to account for their changes during each packet recirculation, which is performed by the queue length encoding module shown in Fig. 1, before encoding them in a probe packet to the ingress pipeline. Specifically, we design a match-action table (MAT) as shown in Fig. 6 in the queue length encoding module, which increases the queue length Q_i of each queue q_i by a constant. For example, assuming that there are three queues whose lengths are Q_1, Q_2 and Q_3, respectively, and the queue lengths sent back to the ingress pipeline by a probe packet will be $Q_1 + c_1^{(1)}$, $Q_2 + c_2^{(1)}$ and $Q_3 + c_3^{(1)}$ if the queue lengths match to the first entry in the MAT. The actual values of the constants are set based on congestion status, *i.e.*, the value of a constant increases with the current length of its corresponding queue. In other words, for each queue, we provide it with a larger increment when its length is longer, and *vice versa*.

E. RHA Packet Scheduling

As illustrated in Fig. 1, the RHA packet scheduling module needs to use an adaptive scheduling algorithm to select a proper queue for each E2E-latency-constrained packet based on the modified queue lengths stored in the ingress pipeline. Specifically, when receiving an E2E-latency-constrained packet, we first calculate its remaining E2E-TQD with Eq. (2), and get the modified queue lengths for its egress port from the corresponding registers. Note that, it is difficult to realize arbitrary multiplication operations in a hardware PDP-SW, and we can only approximate the result of multiplying a variable by a constant "*MathUnit*" in runtime. Hence, we precalculate the values of $\lceil \frac{R}{R_i} \cdot k \rceil$ for different w_i and store them in the PDP-SW as *MathUnits*, and then the range of queuing delay can be estimated with Eq. (3) in runtime.

***Theorem** 1:* For a packet, if for an arbitrary queue, the queuing delay upper-bound estimated based on the current queue length is longer than its TQD t_0, its actual queuing delay in any queue at its egress port should be longer than t_0.

Proof: Considering a packet p whose TQD is t_0, if for an arbitrary queue at its egress port, the queuing delay upper-bound estimated based on the current queue length is longer than t_0, we know that all the queues at its egress port currently have packets to be sent, and thus they are all scheduling packets at the lower-bounds of their dequeuing rates. If we assume that there is a queue q_i that can satisfy the packet's TQD t_0, the queuing delay t_2 of the packet in q_i should be $t_2 \leq t_0$. Then, if by the time when the packet is dequeued from q_i, the queue q_i has always been scheduling packets at the lower-bound of its dequeuing rate continuously, the packet's actual queuing delay will be longer than t_0, which violates the assumption. Therefore, at least one other queue at the packet's egress port needs to empty its packets before that time, to increase the dequeuing rate of q_i. If we assume that there is such a queue and it has taken t_1 to empty its packets, we have $t_1 > t_0$ because the packet has not been sent to the queue (*i.e.*, if we have $t_1 \leq t_0$, the queue can satisfy the packet's TQD and thus we should have sent the packet to it at the first place). However, as the dequeuing rate of q_i only gets improved after t_1, the actual queuing delay of the packet in q_2 should be longer than t_1, resulting in $t_2 > t_1 > t_0$, but

this still contradicts the assumption $t_2 \leq t_0$. Hence, we have proved that the actual queuing delay of p in any of the queues at its egress port should be longer than its TQD t_0. ■

Algorithm 1: RHA packet scheduling scheme

Input: Queue set $Q = \{q_1, q_2, \cdots, q_N\}$ and weight set $W = \{w_1, w_2, \cdots, w_N\}$, $w_i < w_{i+1}, \forall i \in [1, N-1]$.

1 **if** *PDP-SW gets an E2E-latency-constrained packet* p_i **then**
2 $T_q^u = \emptyset$, $count = 0$, $flag = 1$;
3 extract t_i from the *Total Queue Time* field in p_i;
4 get E2E-TQD, remaining hops h_i and egress port of p_i;
5 calculate remaining E2E-TQD T_i of p_i with Eq. (2);
6 $\widetilde{h}_i = \lceil \log_2(h_i) \rceil$;
7 shift T_i right for $\widetilde{h}_i$ bit(s) to get TQD $\widetilde{T}_i$;
8 **for** *each queue* q_j *in* Q **do**
9 get its queue length from corresponding register;
10 estimate the upper-bound of queuing delay $T_{q_j}^u$ with preset "*MathUnit*" and store it in set T_q^u;
11 **end**
12 **while** $count \leq \widetilde{h}_i$ **do**
13 **for** *each queuing delay* $T_{q_k}^u$ *in set* T_q^u **do**
14 **if** $T_{q_k}^u \leq \widetilde{T}_i$ **then**
15 send packet p_i to queue q_k;
16 encode telemetry data in p_i and update length of q_k stored in the egress pipeline;
17 $count = \widetilde{h}_i$, $flag = 0$;
18 **break**;
19 **end**
20 **end**
21 shift $\widetilde{T}_i$ left for one bit;
22 $count = count + 1$;
23 **end**
24 **if** $flag = 1$ **then**
25 drop the packet;
26 **end**
27 **end**
28 **if** *PDP-SW gets a probe packet* **then**
29 forward it to the dedicated recirculation port;
30 get queue lengths from registers in the egress pipeline and modify them based on their status;
31 encode modified queue lengths to the probe packet and sent it back to the ingress pipeline;
32 **end**
33 **if** *PDP-SW gets a recirculated probe packet* **then**
34 extract queue lengths from it and update corresponding registers in the ingress pipeline;
35 drop the packet;
36 **end**

With *Theorem 1*, we design the RHA scheduling algorithm as shown in *Algorithm* 1 to only focus on the queuing delay upper-bound of each queue. Here, we consider the programming constraints of a P4-based hardware PDP-SW to make sure that *Algorithm* 1 can be implemented in it directly. *Lines* 1-27 are for the processing of an E2E-latency-constrained packet. *Lines* 2-5 are for the initialization. For an E2E-latency-constrained packet p_i, its TQD $\widetilde{T}_i$ is calculated based on its remaining E2E-TQD T_i and remaining hops h_i as

$$\widetilde{T}_i = \lfloor \frac{T_i}{h_i} \rfloor, \tag{4}$$

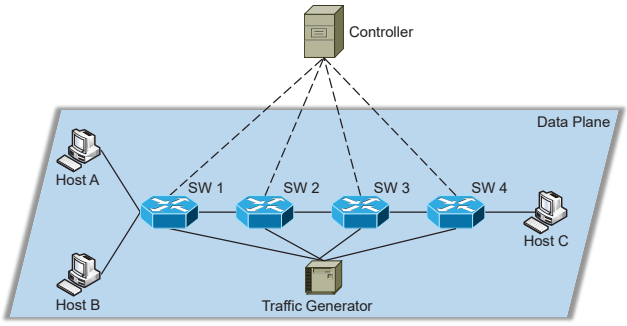


Fig. 7. Experimental setup.

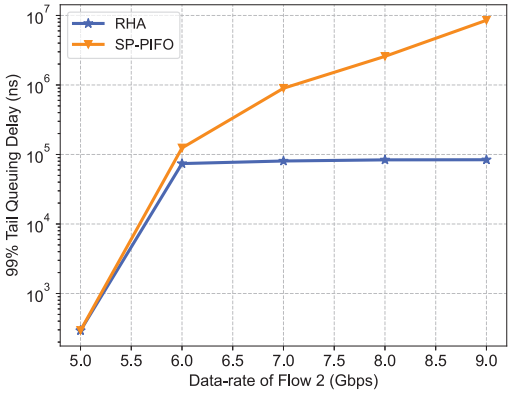
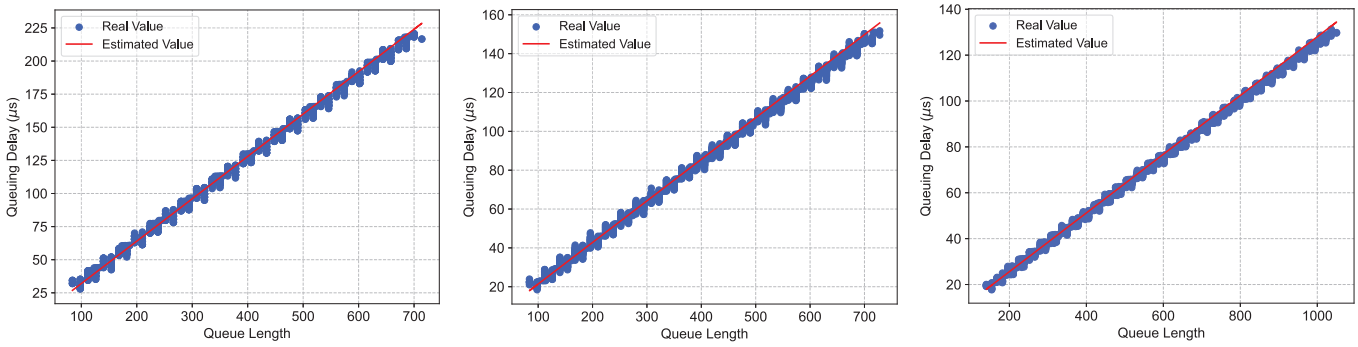


Fig. 8. Experimental results on 99% tail queuing delay.

by assuming that the quota of its remaining E2E-TQD is evenly allocated to each of its remaining hops (*Lines* 6-7). As each packet's routing path is planned by the control plane, each intermediate PDP-SW on the routing path knows exactly about the remaining hops. *Lines* 8-11 estimate the queuing delay upper-bounds of all the queues at the packet's egress port. Then, we check all the queues to select an appropriate one for the packet in *Lines* 12-23. Specifically, we relax the TQD $\widetilde{T}_i$ of the packet by doubling its value in each iteration, and try to find a proper queue satisfying it. If such a queue can be found within $\widetilde{h}_i$ iterations, *Lines* 15-17 schedule the packet to it and update the packet and the PDP-SW's status accordingly. For example, if the number of remaining hops is $h_i \in [5, 8]$, we have $\widetilde{h}_i = 3$ according to *Line* 6. Otherwise, if an appropriate queue cannot be found for the packet, we drop it as its requirement on E2E latency cannot be satisfied (*Lines* 24-26). *Lines* 28-32 and *Lines* 33-36 show the procedures for processing a probe packet and a recirculated probe packet, respectively. As the PDP-SW will just forward a delay-insensitive packet normally without any special operations, we omit the procedure here to save space. Note that, although we assume Tofino-based PDP-SWs here, our RHA scheduling is also applicable to other types of PDP-SWs as long as the required parameters can be obtained.

IV. Experimental Demonstrations

In this section, we discuss the experiments with a real-world testbed to verify the effectiveness of our RHA packet



(a) *Queue* 1 with weight $w_1 = 2$ (b) *Queue* 2 with weight $w_2 = 3$ (c) *Queue* 3 with weight $w_3 = 5$

Fig. 9. Validation of linear relation between queue length and queuing delay.

scheduling scheme and compare it with existing benchmarks.

A. Experimental Setup

We implement our proposed INT-assisted RHA packet scheduling scheme in a real-world network testbed, and Fig. 7 shows the configuration of the testbed. Here, the data plane consists of three end-hosts, four Tofino-based PDP-SWs, and a traffic generator, which are all connected through 10GbE connections. Both the end-hosts and traffic generator are based on Linux servers. The end-hosts generate E2E-latency-constrained flows, and the traffic generator is used to introduce congestion in the PDP-SWs. In the control plane, we have a controller running on a server to manage the PDP-SWs with P4Runtime [36], which provides E2E-TQDs and remaining hops of E2E-latency-constrained packets, and generates probe packets to measure queue lengths on each PDP-SW.

B. Feature Validation

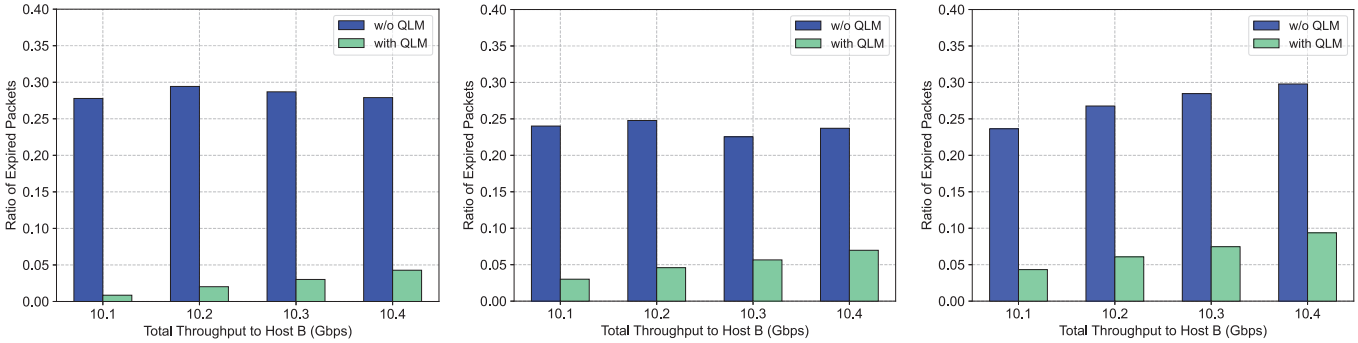
We first verify that our RHA scheduling can alleviate the starvation of packets caused by the SP-PIFO scheduling [25]. We select two queues (namely, *Queues* 1 and 2, respectively) in the *SW* 1 shown in Fig. 7, and set the priority/weight of *Queue* 2 to be higher than that of *Queue* 1 when implementing RHA or SP-PIFO scheduling with the two queues. Then, we let *Host A* and the traffic generator simultaneously send E2E-latency-constrained flows *Flows* 1 and 2 to *Host B*, where the E2E-TQD of *Flow* 2 is shorter than that of *Flow* 1, and schedule packets of the two flows with RHA and SP-PIFO, respectively. Specifically, our RHA scheduling will let each PDP-SW select an appropriate queue for each packet adaptively according to our design, while the SP-PIFO scheduling simply targets the packets of *Flows* 1 and 2 to *Queues* 1 and 2, respectively, to ensure that packets with shorter E2D-TQD are always forwarded out earlier.

The experiments fix the data-rate of *Flow* 1 as 4.1 Gbps and measure the queuing delay of *Queue 1* when the data-rate of *Flow* 2 changes. Fig. 8 plots the experimental results on 99% tail queuing delay (obtained after testing 1000 packets sent from *Queue* 1), where each data point is obtained by averaging the results from 5 independent experiments. It can be seen that the 99% queuing delay from SP-PIFO increases exponentially with the data-rate of *Flow* 2, making packets in *Queue* 1 experience queuing delay at milliseconds or even longer when the data-rate of *Flow* 2 is 7 Gbps or higher. On the other hand, the 99% queuing delay from RHA is similar to that from SP-PIFO when the data-rate of *Flow* 2 is less than 6 Gbps, but it stays at around 100 μs as the data-rate of *Flow* 2 keeps increasing. The experimental results in Fig. 8 suggest that SP-PIFO can cause abnormally-long queuing delay to low-priority packets due to the starvation of packets, which is avoided effectively by our RHA. This is because our RHA uses WRR, which ensures the lower-bound of dequeuing rate for each queue, avoiding the starvation of packets better.

TABLE I
RESOURCE USAGE IN A TOFINO-BASED PDP-SW

Resource	Usage	Percentage (%)
TCAM	20	6.94
SRAM	39	4.06
Hash Bit	113	2.26
Stateful ALU	11	22.92

Next, we conduct experiments to validate Eq. (3), which is derived after our RHA scheduling assumes that the queuing delay of each queue changes linearly with its length. We set $k = 64$ in Eq. (3), measure the queuing delays for different queue lengths, and compare them with the estimated results by Eq. (3). All of the following experiments consider the case of each egress port having three queues, whose weights are $w_1 = 2$, $w_2 = 3$, and $w_3 = 5$, respectively. The results are shown in Fig. 9, where the queue length is in terms of "*cells*", each of which is an 80-byte unit in one queue of Tofino-based PDP-SW. We can see that for all the three queues, the estimated queuing delays by Eq. (3) match well with the real values measured in the experiments. This confirms the approximate linear relation between the length and queuing delay of each queue. Meanwhile, by comparing the results in Figs. 9(a)-9(c), we observe that increasing a queue's weight makes its queuing delay for a specific queue length shorter. Fig. 9 validates that we can precisely estimate queuing delay based on queue length with Eq. (3), justifying the algorithmic basis of our proposal. Table I shows the resource usage in a Tofino-based PDP-SW with our RHA scheduling scheme deployed.



(a) E2E-TQD is 24 μs (b) E2E-TQD is 20 μs (c) E2E-TQD is 16 μs

Fig. 10. Effectiveness of improving accuracy of queue length estimation with queue length modification (QLM) at egress pipeline.

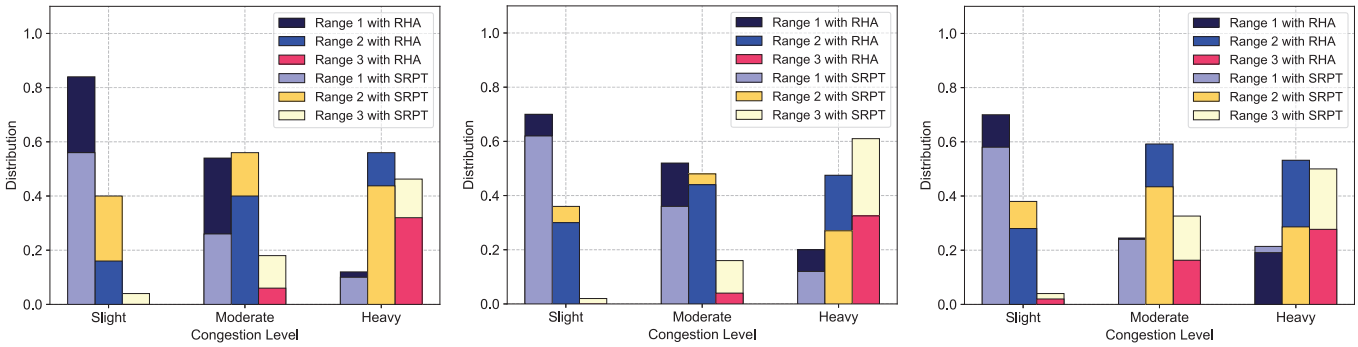
Finally, we measure the accuracy of queuing delay estimation after packet recirculation when our proposed queue length modification (QLM) is applied (with the MAT in an egress pipeline, as shown in Fig. 6). According to the operation principle in Fig. 1, each PDP switch will directly drop the E2E-latency-constrained packets whose required E2E latency has expired. However, this cannot rule out the possibility that the receiving end gets E2E-latency-expired packets, because an E2E-latency-constrained packet whose required E2E latency has not expired by the time when it enters the last hop might be scheduled inappropriately to make its E2E latency longer than the required value at the receiving end. Therefore, we design the experiments to let *Host A* and traffic generator simultaneously send E2E-latency-constrained traffic (with E2E-TQDs randomly selected from $\{16, 20, 24\}$ μs) to *Host B* through *SW* 1, generating different levels of congestion there.

The results are shown in Fig. 10, where each data point is also obtained by averaging the results from 5 independent runs, and the ratio of expired packets is defined as the proportion of expired packets to all the packets ($10,000$ of them) received at *Host B*. It can be seen that QLM effectively reduces the ratio of expired packets over the scheme without it, verifying that it can indeed improve the accuracy of queuing delay estimation by modifying the queue lengths obtained at the egress pipeline to compensate for the inaccuracy due to packet recirculation. We also observe that for the scheme with QLM, the ratio of expired packets increases with the total throughput to *Host B*. This is because when the congestion level becomes severer, it is more difficult for RHA scheduling to ensure E2E latency of each packet. However, with QLM, the ratio of expired packets is still significantly lower than that without it, which further confirms the effectiveness of QLM. By comparing the results in Figs. 10(a)-10(c), we find that the recirculation delay indeed affects the performance of scheduling, causing a relative large portion of expired packets if QLM is not applied, but QLM can effectively mitigate the impact of recirculation delay and reduce the portion of expired packets significantly.

C. Performance Benchmarking

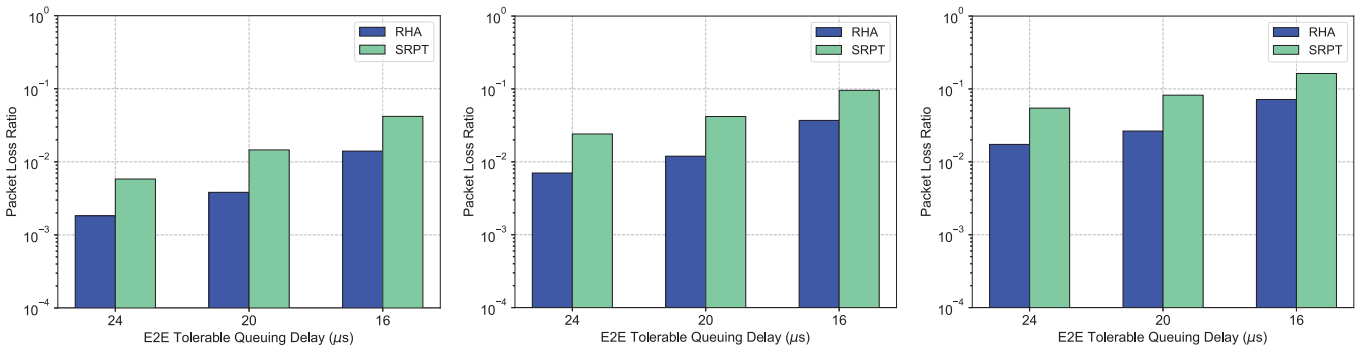
In order to further verify the effectiveness of our RHA packet scheduling scheme, we compare it with the packet scheduling scheme designed in SRPT [28], which ignores the remaining hops of each packet. Specifically, the experiments let *Host A* send E2E-latency-constrained flows with different E2E-TQDs to *Host C*, and measure the queuing delay that packets experience in *SW 1*. The experimental results are shown in Fig. 11, where *Range 1*, *Range 2* and *Range 3* are for (0, E2E-TQD/4], (E2E-TQD/4, E2E-TQD/2], and (E2E-TQD/2, E2E-TQD], respectively. Here, we generate congestion constantly over time, and define the state of congestion based on the results of SRPT. For example, if under the current congestion state, most packets of a flow using SRPT can still be delivered with short queuing delay, it is "slight congestion" for the flow, and it is "heavy congestion", otherwise. We find that both RHA and SRPT can avoid most packets experiencing long queuing delay when congestion is slight. However, as congestion aggravates, it becomes more difficult to maintain short queuing delay for E2E-latency-constrained packets, and thus the proportion of packets with relatively long queuing delay increases. RHA makes less packets experience long queuing delay than SRPT. This is because SRPT schedules packets only based on their remaining E2E-TQDs but ignores their remaining hops. As RHA makes more packets experience short queuing delay, it reduces the probability of these packets being dropped in subsequent hops, increasing the chance of forwarding them to *Host C* within their required E2E latency.

Finally, we compare the packet loss ratio of the scenarios with SRPT and RHA, when E2E-latency-constrained packets encounter different congestion probabilities on their path to their receiving end host. Specifically, we still let *Host A* send E2E-latency-constrained flows whose E2E-TQDs are selected from $\{16, 20, 24\}$ μs to *Host C*, introduce different degrees of congestion on each PDP-SW along their routing path with various probabilities to emulate the cases in a real-world network, and schedule packets with RHA and SRPT, respectively. The experimental results are shown in Fig. 12, and they are obtained by averaging over 10 independent experiments. As expected, the packet loss ratios from both RHA and SRPT increase with the congestion probability, and packets with stricter E2E latency requirements are more likely to be lost. RHA always outperforms SRPT to drop less packets in all the experimental scenarios, which verifies that RHA can schedule



(a) E2E-TQD is 24 μs (b) E2E-TQD is 20 μs (c) E2E-TQD is 16 μs

Fig. 11. Distribution of queuing delay for E2E-latency-constrained under different congestion levels.



(a) Congestion probability is 10% (b) Congestion probability is 20% (c) Congestion probability is 30%

Fig. 12. Packet loss ratio for delay-sensitive flows that have different E2E-TQDs with different congestion probabilities.

E2E-latency-constrained packets better than SRPT.

V. Conclusion

In this paper, we proposed an INT-assisted adaptive packet scheduling scheme to ensure the E2E latency of delay-sensitive flows, and optimized its implementation in P4-based hardware PDP-SWs. We first leveraged INT to catch realtime status of packets such that each PDP-SW can obtain their remaining E2E-TQDs accurately. Then, to avoid the starvation of packets, we adopted the WRR approach for the queues at each egress port of a PDP-SW. Next, we studied the relation between queue length and queuing delay, such that each PDP-SW can estimate queuing delay based on queue lengths obtained from its egress pipeline through packet recirculation. We also designed a queue length modification scheme to mitigate the impact of recirculation delay to make the obtained queue lengths more accurate. These efforts were combined to propose the INT-assisted adaptive packet scheduling system with RHA scheduling, which can schedule each E2E-latency-constrained packet based on its realtime status, remaining E2E-TQD, remaining hops, and the upper-bound queuing delay of queues in each PDP-SW along its routing path, to satisfy its QoS demand on E2E latency. Our implementation of the RHA packet scheduling was demonstrated experimentally in a real-world network testbed, and we evaluated it against representative benchmarks. Experimental results indicated that our proposal relieved the starvation of packets and ensured E2E latency for delay-sensitive packets better than the benchmarks.

Acknowledgments

This work was supported by the National Key R&D Program of China under Grant 2023YFB2903903.

References

[1] Cisco Annual Internet Report (2018-2023). [Online]. Available: https://www.cisco.com/c/en/us/solutions/collateral/executive-perspectives/annual-internet-report/white-papper-c11-741490.html.

[2] P. Lu *et al.*, "Highly efficient data migration and backup for Big Data applications in elastic optical inter-data-center networks," *IEEE Netw.*, vol. 29, pp. 36–42, Sept./Oct. 2015.

[3] P. Lu and Z. Zhu, "Data-oriented task scheduling in fixed- and flexible-grid multilayer inter-DC optical networks: A comparison study," *J. Lightw. Technol.*, vol. 35, pp. 5335–5346, Dec. 2017.

[4] V. Dukic *et al.*, "Beyond the mega-data center: Networking multi-data center regions," in *Proc. of ACM SIGCOMM 2020*, pp. 765–781, Aug. 2020.

[5] J. Liu *et al.*, "On dynamic service function chain deployment and readjustment," *IEEE Trans. Netw. Serv. Manag.*, vol. 14, pp. 543–553, Sept. 2017.

[6] A. Matencio-Escolar, Q. Wang, and J. Calero, "SliceNetVSwitch: Definition, design and implementation of 5G multi-tenant network slicing in software data paths," *IEEE Trans. Netw. Serv. Manag.*, vol. 17, pp. 2212–2225, Oct. 2020.

[7] R. Gour, G. Ishigaki, J. Kong, and J. Jue, "Availability-guaranteed slice composition for service function chains in 5G transport networks," *J. Opt. Commun. Netw.*, vol. 13, pp. 14–24, Jan. 2021.

[8] W. Lu, Z. Zhu, and B. Mukherjee, "On hybrid IR and AR service provisioning in elastic optical networks," *J. Lightw. Technol.*, vol. 33, pp. 4659–4669, Nov. 2015.

[9] Z. Zhu, W. Lu, L. Zhang, and N. Ansari, "Dynamic service provisioning in elastic optical networks with hybrid single-/multi-path routing," *J. Lightw. Technol.*, vol. 31, pp. 15–22, Jan. 2013.
[10] L. Gong *et al.*, "Efficient resource allocation for all-optical multicasting over spectrum-sliced elastic optical networks," *J. Opt. Commun. Netw.*, vol. 5, pp. 836–847, Aug. 2013.
[11] Y. Yin *et al.*, "Spectral and spatial 2D fragmentation-aware routing and spectrum assignment algorithms in elastic optical networks," *J. Opt. Commun. Netw.*, vol. 5, pp. A100–A106, Oct. 2013.
[12] L. Gong and Z. Zhu, "Virtual optical network embedding (VONE) over elastic optical networks," *J. Lightw. Technol.*, vol. 32, pp. 450–460, Feb. 2014.
[13] J. Case, M. Fedor, M. Schoffstall, and J. Davin, "A simple network management protocol (SNMP)," *RFC 1157*, May 1990. [Online]. Available: https://tools.ietf.org/html/rfc1157.
[14] P. Phaal, S. Panchen, and N. McKee, "InMon corporation's sFlow: A method for monitoring traffic in switched and routed networks," *RFC 3176*, Sept. 2001. [Online]. Available: https://tools.ietf.org/html/rfc3176.
[15] B. Claise, "Cisco systems NetFlow services export version 9," *RFC 3954*, Oct. 2004. [Online]. Available: https://tools.ietf.org/html/rfc3954.
[16] S. Ha, I. Rhee, and L. Xu, "Cubic: A new TCP-friendly high-speed TCP variant," *ACM SIGOPS Oper. Syst. Rev.*, vol. 42, pp. 64–74, Jul. 2008.
[17] M. Alizadeh *et al.*, "Date center TCP (DTCP)," in *Proc. of ACM SIGCOMM 2010*, pp. 63–74, Aug. 2010.
[18] G. Abbas, Z. Halim, and Z. Abbas, "Fairness-driven queue management: A survey and taxonomy," *IEEE Commun. Surveys Tuts.*, vol. 18, pp. 324–367, First Quarter 2016.
[19] P. Bosshart *et al.*, "P4: Programming protocol-independent packet processors," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 44, pp. 87–95, Jul. 2014.
[20] INT dataplane specification. [Online]. Available: https://github.com/p4lang/p4-applications/blob/master/docs/INT_v2_1.pdf.
[21] Z. Xu, S. Tang, and Z. Zhu, "Entropy-driven adaptive INT and its applications in network automation of IP-over-EONs," *IEEE J. Sel. Top. Quantum Electron.*, vol. 28, pp. 1–13, Mar. 2022.
[22] S. Tang, J. Kong, B. Niu, and Z. Zhu, "Programmable multilayer INT: An enabler for AI-assisted network automation," *IEEE Commun. Mag.*, vol. 58, pp. 26–32, Jan. 2020.
[23] F. Cugini *et al.*, "P4 in-band telemetry (INT) for latency-aware VNF in metro networks," in *Proc. of OFC 2019*, pp. 1–3, Mar. 2019.
[24] F. Paolucci, D. Scano, P. Castoldi, and E. Paoli, "Latency control in service chaining using P4-based data plane programmability," *Comput. Netw.*, vol. 216, p. 109227, Oct. 2022.
[25] A. Alcoz, A. Dietmuller, and L. Vanbever, "SP-PIFO: Approximating push-in first-out behaviors using strict-priority queues," in *Proc. of NSDI 2020*, pp. 1–19, Feb. 2020.
[26] Z. Yu *et al.*, "Programmable packet scheduling with a single queue," in *Proc. of ACM SIGCOMM 2021*, pp. 179–193, Aug. 2021.
[27] Z. Li, Y. Hu, L. Tian, and Z. Lv, "Packet rank-aware active queue management for programmable flow scheduling," *Comput. Netw.*, vol. 225, p. 109632, Feb. 2023.
[28] M. Alizadeh *et al.*, "pFabric: Minimal near-optimal datacenter transport," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 43, pp. 435–446, Aug. 2013.
[29] B. Niu *et al.*, "Visualize your IP-over-optical network in realtime: A P4-based flexible multilayer in-band network telemetry (ML-INT) system," *IEEE Access*, vol. 7, pp. 82 413–82 423, Jun. 2019.
[30] S. Tang *et al.*, "Sel-INT: A runtime-programmable selective in-band network telemetry system," *IEEE Trans. Netw. Serv. Manag.*, vol. 17, pp. 708–721, Jun. 2020.
[31] A. Sgambelluri *et al.*, "Exploiting telemetry in multi-layer networks," in *Proc. of ICTON 2020*, pp. 1–4, Jul. 2020.
[32] M. Anand, R. Subrahmaniam, and R. Valiveti, "POINT: An intent-driven framework for integrated packet-optical in-band network telemetry," in *Proc. of ICC 2018*, pp. 1–6, May 2018.
[33] P. Goyal, H. Vin, and H. Cheng, "Start-time fair queueing: A scheduling algorithm for integrated services packet switching networks," *IEEE/ACM Trans. Netw.*, vol. 5, pp. 690–704, Oct. 1997.
[34] F. Checconi, L. Rizzo, and P. Valente, "QFQ: Efficient packet scheduling with tight guarantees," *IEEE/ACM Trans. Netw.*, vol. 21, pp. 802–816, Oct. 2012.
[35] X. Li *et al.*, "RPQ: Resilient-priority queue scheduling for delay-sensitive applications," in *Proc. of HPSR 2022*, pp. 1–6, Jun. 2022.
[36] P4Runtime specification. [Online]. Available: https://p4.org/p4-spec/p4runtime/main/P4Runtime-Spec.html.