Information-Sensitive In-band Network Telemetry in P4-based Programmable Data Plane

Zichen Xu, Ziye Lu, and Zuqing Zhu, *Fellow, IEEE*

***Abstract*—With the development of programmable data plane (PDP), in-band network telemetry (INT) has become a promising network monitoring technique to visualize network operations in a fine-grained and real-time way. In this work, to better balance the tradeoff between INT overheads and monitoring accuracy, we design and optimize an information-sensitive INT system (namely, P4InfoSen-INT), which makes each PDP switch decide locally whether and what type(s) of telemetry data should be inserted in a packet based on the "information content" of the data, and implement it in P4-based PDP switches. We first realize the basic principle of P4InfoSen-INT with P4 programs. Then, we propose algorithms to estimate the information content of telemetry data accurately in a dynamic network and optimize the tradeoff between INT overheads and monitoring accuracy. Finally, we further optimize the implementation of P4InfoSen-INT by proposing table merging to reduce stage occupation in each switch. Experimental results verify that our proposed P4InfoSen-INT can balance the tradeoff between INT overheads and monitoring accuracy better than existing benchmarks.**

***Index Terms*—Programmable data plane (PDP), Tofino, P4, In-band network telemetry (INT), In-network computing.**

I. Introduction

Over past decades, network traffic has grown dramatically due to the fast development of data-centers (DCs) [1–4] and 5G networks [5, 6], and simulated various network technologies, *e.g.*, software-defined networking (SDN) [7, 8], virtual network slicing [9–11], and network function virtualization (NFV) [12–14]. Consequently, networks are becoming increasingly flexible and programmable at the cost of increased complexity, making network monitoring and troubleshooting more and more challenging [15]. Note that, in today's Internet, network operators need to not only provision highly-dynamic traffic but also satisfy more stringent and diversified quality-of-service (QoS) demands [16–18], and thus they have to monitor their networks in a fine-grained and real-time manner, such that network anomalies (*e.g.*, congestions and misconfigurations) can be detected, located and recovered promptly.

However, traditional polling-based network monitoring approaches (*e.g.*, SNMP [19], sFlow [20], and Netflow [21]) can hardly cope with the aforementioned emerging requirements. First, they normally need to place an agent on each monitored network element (NE) to collect and report status data, which consumes the NE's computing and forwarding resources and thus might impact its packet forwarding performance. Second, as these techniques let NEs report status data periodically or on demand in the out-of-band manner, they have difficulty precisely catching real-time status of a highly-dynamic network. Lastly but most importantly, they cannot reveal the end-to-end operation of packets in an arbitrary flow.

The rising and development of programmable data plane (PDP) [22, 23] provide new opportunities for network monitoring. Specifically, the programming protocol-independent packet processor (P4) [22] and the protocol oblivious forwarding (POF) [23] have been developed for realizing PDP, both of which can customize the packet processing in a switch to support new network monitoring schemes such as in-band network telemetry (INT) [24, 25]. INT lets the ingress switch of a flow insert INT instructions in packets, based on which each subsequent switch collects the network status that a packet sees and encodes the obtained telemetry data as INT fields in the packet. Then, the egress switch extracts the INT fields to recover how the packet gets processed hop-by-hop. Previous studies have implemented INT with both P4 [26] and POF [27], achieving flow-oriented and real-time monitoring.

Despite its benefits, INT increases packet lengths and brings in extra bandwidth and packet processing overheads. Therefore, researchers previously have proposed various techniques to reduce the overheads of INT, including sampling packets for INT field insertion [27, 28], distributing different types of INT fields among packets [26, 29], or probabilistically selecting telemetry data types to insert [30]. Although these sampling-based techniques have all shown their effectiveness, there are still unsolved problems. Specifically, sampling packets for INT can indeed relieve bandwidth overheads, but it would also reduce monitoring accuracy inevitably. Hence, it is still not clear how to determine and adjust the sampling rate of INT such that the tradeoff between INT overheads and monitoring accuracy can be optimized in a dynamic network environment. To the best of our knowledge, this issue is still under-explored.

In [31], we proposed the idea of information-sensitive INT, which is to let each switch determine whether and what type(s) of telemetry data should be inserted in a packet locally, based on the "information content" of telemetry data. In other words, the INT fields of various types of telemetry data are prioritized according to the amount of information that they can convey to the network monitoring system. For instance, the telemetry data indicating a sudden increase of packet processing latency can convey more information than the normally-used output port of a packet flow, and thus it should be reported with a higher priority. To demonstrate the idea quickly, we realized it with software-based POF switches (namely, EntropyINT [31]).

However, the study on information-sensitive INT in [31] is still preliminary from both the system and algorithm per-

Z. Xu, Z. Lu and Z. Zhu are with the School of Information Science and Technology, University of Science and Technology of China, Hefei, Anhui 230027, P. R. China (email: zqzhu@ieee.org).

Manuscript received on July 31, 2023.

spectives. In the system aspect, software switches perform significantly worse than hardware switches on packet processing. For example, a P4-based PDP switch with Tofino ASICs can deliver 12.8-Tbps throughput with line-rate up to 400 Gbps [32], while its cost is not prohibitively high [33]. Note that, it is still challenging to implement information-sensitive INT in such a hardware switch, even though the software-based approach has already been realized in [31]. This is because in order to do so, we have to address unique hardware restrictions properly and optimize the system design from the scratch. The hardware restrictions of PDP switch (*e.g.*, it can only store integer and does not support numerical operations like multiplication and division) prevent us from realizing the principle of information-sensitive INT directly. Furthermore, PDP switch only has limited memory, restricting the number of stages that a pipeline can include. This makes it vital for us to optimize the stage usage of information-sensitive INT with table merging, which is known to be challenging [33–35].

In the algorithm aspect, EntropyINT [31] did not address the problem of adapting information-sensitive INT to dynamic networks. Without the algorithm to determine when and how to update the status of information-sensitive INT in each switch adaptively, EntropyINT cannot estimate the information content of each telemetry data accurately in a dynamic network or balance the tradeoff between INT overheads and monitoring accuracy properly. Hence, the missing of algorithm design in [31] makes its system incomplete, and thus the benefits of information-sensitive INT were not fully explored to optimize the tradeoff between INT overheads and monitoring accuracy.

The aforementioned restrictions of EntropyINT motivate us to implement and optimize information-sensitive INT in P4-based hardware PDP switches. This work continues to optimize information-sensitive INT such that it can be implemented in PDP switches built with Tofino ASICs, and demonstrates an adaptive, efficient and real-time network monitoring system, namely, P4InfoSen-INT. The basic principle of information-sensitive INT is first realized with P4 programs. Then, we propose algorithms to help P4InfoSen-INT accurately estimate the information content of each telemetry data in a dynamic network, optimizing the tradeoff between INT overheads and monitoring accuracy. Finally, we further optimize the implementation of P4InfoSen-INT by proposing table merging techniques to reduce stage occupation in each switch. Experimental results verify that our proposed P4InfoSen-INT can balance the tradeoff between INT overheads and monitoring accuracy better than the existing benchmarks.

Our major contributions can be summarized as follows:

- We design and implement P4InfoSen-INT, which is an information-sensitive INT system that can realize flow-oriented, real-time and adaptive network monitoring.
- We optimize the data plane implementation of P4InfoSen-INT to reduce the stage occupation of its packet processing pipeline in each P4-based hardware PDP switch.
- We propose algorithms to facilitate the control plane implementation of P4InfoSen-INT, such that the information content of each telemetry data can be estimated accurately in a dynamic network and the tradeoff between INT overheads and monitoring accuracy can be balanced well.
- We build a small-scale but realistic network testbed to demonstrate P4InfoSen-INT experimentally and verify its benefits over the existing benchmarks.

The rest of the paper is organized as follows. Section II discusses the related work briefly. We present the system design and operation principle of P4InfoSen-INT in Section III. Our proposed algorithms for optimizing the data plane implementation and facilitating control plane operations are described in Sections IV and V, respectively. In Section VI, we show the experimental demonstrations for performance evaluation. Finally, Section VII summarizes this paper.

II. Related Work

Due to its advantages, INT has spurred extensive research activities and has been standardized in [24]. People have implemented INT in switches based on P4 [26] and POF [27], and deployed them to monitor various networks [36, 37]. However, the early day implementations of INT (*e.g.*, that in [38]) inserted INT fields without considering the overheads, which means that when a flow is selected for INT, each switch on its routing path will insert all the required INT fields in each of its packets. This will lead to extremely high bandwidth overheads, where some of them might not be necessary.

The studies in [39] and [40] have addressed how to reduce the bandwidth overheads when using the per-packet INT scheme. Specifically, they proposed algorithms to plan the routing paths of INT-enabled flows such that the coverage of network monitoring can be maximized and the collection of redundant telemetry data can be avoided. However, these approaches did not try to optimize the operation principle of INT to minimize its overheads. Researchers have also considered to filter out redundant telemetry data before sending it to the control plane in [41, 42]. Although they effectively relieved the data processing load in the control plane, they cannot reduce the bandwidth overheads of INT in networks.

Note that, sampling network status in the per-packet manner might not be necessary, because the working status of a switch usually would not change dramatically in a very short time, especially in metro or core networks where the line-rate is relatively high (*i.e.*, in Gbps or even higher [43]). Therefore, the studies in [26–30] considered to sample packets or/and types of telemetry data for INT field insertions to relieve the bandwidth overheads of INT. The approaches developed in [26–29] selected a portion of packets in each flow to insert INT fields, the work of [26, 29] tried to distribute different types of telemetry data among packets, and the authors of [30] proposed to sample different types of telemetry data with preset probabilities. Even though these sampling-based schemes did reduce the overheads of INT effectively, they have not addressed how to determine the sampling ratio adaptively in a dynamic network such that the tradeoff between INT overheads and monitoring accuracy can be balanced well. In [44, 45], we addressed the selection of sampling rate for INT adaptively, but it was from the perspective of ensuring that the extra bandwidth overheads caused by INT will not cause congestion on already heavy-loaded links.

We proposed EntropyINT in [31], implemented it on software switches, and demonstrated that it could balance the

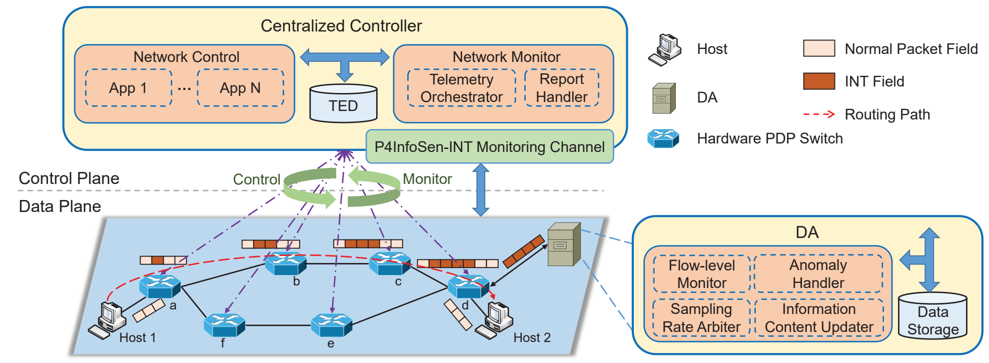


Fig. 1. System architecture of network with P4InfoSen-INT, TED: Traffic engineering database, DA: Data analyzer.

TABLE I
MAJOR ABBREVIATIONS

Abbrev.	Full Name	Abbrev.	Full Name
PDP	Programmable data plane	PMT	Priority mapping table
INT	In-band network telemetry data	TCAM	Ternary content addressable memory
QoS	Quality-of-service	SRAM	Static random-access memory
NE	Network element	PHV	Packet header vector
DA	Data analyzer	MPMT	Merged priority mapping table
TED	Traffic engineering database	PRMT	Probability mapping table
MAU	Match-action unit	TM	Table merging
MAT	Match-action table	TEM	Table entry merging

tradeoff between INT overheads and monitoring accuracy better than existing approaches. Nevertheless, as we have already explained, the packet processing performance of EntropyINT was severely limited by software switches, and algorithms have not been designed to enable the control plane operations such that the information content of each telemetry data can be estimated accurately in a dynamic network and the tradeoff between INT overheads and monitoring accuracy can be further optimized. Therefore, to the best of our knowledge, this is the first work to design and optimize a high-performance information-sensitive INT system based on hardware switches.

III. System Design and Operation Principle

This section will first give an overview of our design, and then describe the system architecture and operations in data and control planes. As several acronyms are frequently used in this paper, we list them in Table I for the readers' convenience.

A. Design Overview

As monitoring flow performance with per-packet INT is normally not necessary [27], we design P4InfoSen-INT to let each switch sample packets of a flow with a dynamic sampling rate, which reflects the interval between two consecutive packets that contain INT fields and is adjusted adaptively according to real-time network status, such that the tradeoff between INT overheads and monitoring accuracy can be balanced optimally. The ultimate goal of flow monitoring is to detect and locate network anomalies accurately and promptly. Meanwhile, it is known that network anomalies usually happen with a much lower probability than normal network state (*e.g.*, in normal network operation, congestion only occurs occasionally on a switch). This suggests that if the probability of occurrence of a telemetry data value is smaller, the data actually conveys more information to flow monitoring. Hence, we can quantify the importance of telemetry data (or its information content) based on its probability of occurrence and make each switch select and insert telemetry data in packets accordingly. Specifically, when a packet is selected for INT field insertion, the switch inserts the telemetry data with the largest information content. Then, P4InfoSen-INT is designed to realize such a mechanism automatically during flow monitoring.

B. System Architecture

The system architecture is shown in Fig. 1. There are three types of NEs in the data plane, *i.e.*, the end hosts, P4-based switches with Tofino ASICs, and data analyzers (DAs). The end hosts send/receive application traffic, the switches are programmed to support P4InfoSen-INT, and the DAs are designed to analyze collected telemetry data and report digested network status to the control plane. In a DA, the flow-level monitor parses and extracts the telemetry data encoded in the INT fields of INT packets[1], and records and analyzes the data in its data storage. When an anomaly is detected, it is sent to the anomaly handler, which will forward it to the control plane through the P4InfoSen-INT monitoring channel for further processing. Otherwise, the sampling rate arbiter and information content updater assist the control plane to adjust the sampling rate and information mapping scheme of telemetry data for P4InfoSen-INT. Note that, as INT packets are mirrored to DAs by the egress switches of flows, DAs are placed at the edge of the network. Therefore, the DAs actually conduct distributed data analytics for real-time and flow-level monitoring and troubleshooting, which offload a significant part of network monitoring tasks from the control plane.

The control plane is essentially a centralized SDN controller, for managing switches in the data plane. In the controller, the traffic engineering database (TED) records the service provisioning schemes of traffic flows and works with the network control and network monitor modules to facilitate network automation. We design two modules in the network monitor to realize self-adaptive monitoring and troubleshooting. Specifically, the report handler receives and analyzes the reports from DAs, while the telemetry orchestrator arranges the parameters of P4InfoSen-INT running in

[1]We refer to each packet that contains INT fields as an "INT packet".

switches. Therefore, both the report handler and telemetry orchestrator need to interact with the network control module. The report handler will forward the anomalies that it detects to the network control module, which will then adjust the provisioning schemes of affected flows to restore their services. On the other hand, the telemetry orchestrator consistently analyzes the working status of flows and their settings of P4InfoSen-INT, to determine whether the settings of P4InfoSen-INT need to be updated and how to update them accordingly. Then, if necessary, the telemetry orchestrator will suggest the network control module to update the settings of P4InfoSen-INT to adapt to dynamic network status and QoS demands of flows.

C. Data Plane Operations

1) Background on Switch with Tofino ASICs: Before describing the data plane operations, we first briefly introduce the architecture of a hardware switch with Tofino ASICs. In Fig. 2, such a switch leverages ingress and egress pipelines to process packets, where each pipeline consists of blocks such as a *Parser*, multiple *Match-Action Units (MAUs)*, and a *Deparser* [22]. Each MAU occupies a stage, which represents a unit of packet processing resources in one pipeline. The space of the static random-access memory (SRAM) and ternary content addressable memory (TCAM) in a pipeline is evenly allocated to each stage in it. As the *Parser*, *MAUs* and *Deparser* are all programmable with P4 language, we can customize packet processing in the switch with great flexibility.

When a packet arrives, it first enters the Parser in the ingress pipeline, which extracts its header fields, then passes through MAUs to experience lookup tables and the corresponding actions, and finally exits the ingress pipeline at Deparser, which encapsulates its header fields back. Similar operations are performed in the egress pipeline. The header fields and metadata (*e.g.*, intermediate results from table lookup) are stored in the packet header vector (PHV) containers in each stage, whose lengths can be $\{8, 16, 32\}$ bits and total space is fixed. Then, the parameter transfers between adjacent blocks in a pipeline can be realized with the PHV containers.

2) Principle of P4InfoSen-INT: We implement and optimize our information-sensitive INT (namely, P4InfoSen-INT) in hardware PDP switches. The idea of P4InfoSen-INT is to 1) sample packets in a flow to insert INT fields that contain telemetry data about the flow's forwarding status, and 2) introduce stateful processing in switches such that they can make local decisions to tell what type(s) of INT fields should be inserted in each INT packet based on the fields' information contents. Specifically, if we denote the probability that a type of discrete telemetry data Y (*e.g.*, *Out Port*) takes value y_i as

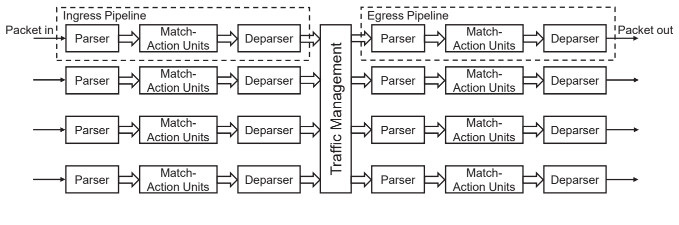


Fig. 2. Packet forwarding architecture of Tofino-based switch.

Match	Action	Action Data
data: y_1: exact	get_priority()	priority = p_1^*
default	set_priority()	priority = p_2^*

(a) MAT for a type of discrete telemetry data

Match	Action	Action Data
data: $[y_1', y_2')$: range	get_priority()	priority = p_1
data: $[y_2', y_3')$: range	get_priority()	priority = p_2
$\vdots$	$\vdots$	$\vdots$
data: $[y_{n-1}', y_n')$: range	get_priority()	priority = p_n

(b) MAT for a type of continuous telemetry data

Fig. 3. Designs of a PMT based on MATs for different telemetry data types.

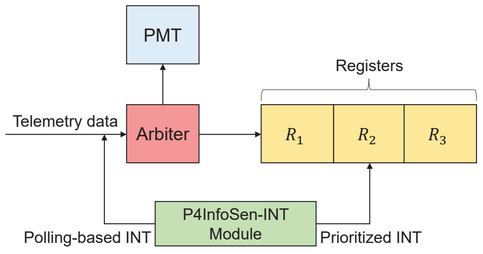


Fig. 4. Operation principle of P4InfoSen-INT.

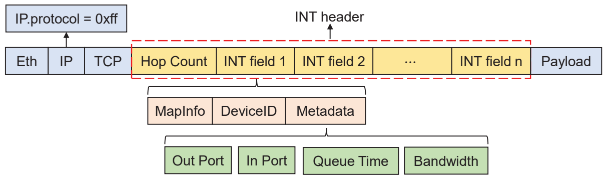
$P(Y = y_i)$, and that of a type of continuous telemetry data Y' (*e.g.*, *Bandwidth*) in range $[y_i', y_{i+1}')$ as $P(Y' \in [y_i', y_{i+1}'))$, their information contents are

$$\begin{cases} I_i = -\log_2[P(Y = y_i)], \\ I_i' = -\log_2[P(Y' \in [y_i', y_{i+1}'))]. \end{cases} \tag{1}$$

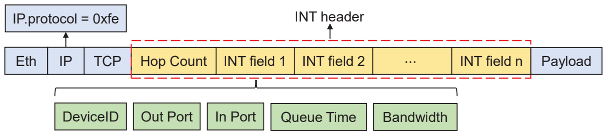
Eq. (1) provides us a method to quantitatively compare the importance of telemetry data in different types [31]. However, due to its limited arithmetic capability, Tofino ASIC cannot directly calculate or store information contents. Therefore, we design a priority mapping table (PMT) by leveraging the match-action tables (MATs), as shown in Fig. 3. Specifically, we use MATs that have *exact* match type for discrete telemetry data, and those with *range* match type for continuous telemetry data, while the "action" of each MAT denotes the priority of its telemetry data. Here, the ranges of the match key in Fig. 3(b) are in ascending order, *i.e.*, $y_i' < y_{i+1}', \forall i$. Then, we can let the SDN controller calculate and sort the information contents of all the data types in ascending order and incrementally assign integer priorities to them, forming the PMTs in switches.

Similar to the existing selective INT schemes [27, 30], P4InfoSen-INT also samples packets in each monitored flow for INT operations, *i.e.*, within each sampling cycle, only one packet in the flow will be selected to be inserted INT fields and become an INT packet. Then, we design two INT operations for P4InfoSen-INT, namely, **polling-based INT** and **prioritized INT**. The polling-based INT is introduced to avoid the situation where certain types of telemetry data will not be collected for a long time, and thus it just lets a switch encode all the telemetry data selected for a flow in one INT packet.

The procedure of the prioritized INT in P4InfoSen-INT can be explained with Fig. 4, where we allocate three registers (*i.e.*, R_1, R_2 and R_3) to each monitored flow. Here, R_2 will store



(a) Prioritized INT packet



(b) Polling-based INT packet

Fig. 5. Formats of INT packets in P4InfoSen-INT.

Algorithm 1: Collection of bandwidth usage

```
Input: Collection period $\Delta t$
Output: Bandwidth usage $B$
1  set up registers $R_4 = \{\tau\}$ (time) and $\widetilde{R}_1 = \{c_1, c_2\}$ (counters);
2  $\tau = c_1 = c_2 = 0$;
3  while one packet of the monitored flow arrives do
4      get current system time $t$ and packet length $L$;
5      if $t - \tau \geq \Delta t$ then
6          $\tau = t$, $c_1 = c_2$, $c_2 = L$;
7      else
8          $c_2 = c_2 + L$;
9      end
10     return $B = c_1$;
11 end
```

the value of the telemetry data that has the highest information-aware priority in the current sampling cycle, and R_1 and R_3 store the data's priority and type ID, respectively. Then, upon receiving each packet of a monitor flow, a switch collects all types of telemetry data selected for the flow, uses the arbiter to check the PMT and find the data with the highest priority, and sends the data and its parameters (*i.e.*, its priority and type ID) to the registers. The switch then compares the data's priority with that of the data currently stored in R_2, and updates the registers if the new data's priority is higher. Then, when the current sampling cycle ends, the switch encodes an INT field with the data in R_2 and inserts it in the INT packet. Therefore, prioritized INT always reports the telemetry data with the largest information content in each sampling cycle.

3) Packet Formats: The principle of P4InfoSen-INT determines that there will be three types of packets, *i.e.*, the **normal packets** that are not selected for INT operations, the **prioritized INT packets** that carry the telemetry data with the highest priority, and the **polling-based INT packets** that carry all types of selected telemetry data. The latter two are for INT packets and their formats are shown in Fig. 5, which are adapted from the standard INT packet format defined in [24] with only minor modifications.

In each INT packet, P4InfoSen-INT inserts an INT header in between its TCP/UDP header and payload at each switch on its flow's routing path. We leverage the "protocol" field in IP header to indicate the type of a packet. Specifically, the field is modified to *0xff* or *0xfe* by an ingress switch to mark a prioritized INT packet or polling-based INT packet, respectively, and is restored to its original value by an egress switch. Each INT header contains a field of *Hop Count* followed by a number of *INT Fields*, which is the same for prioritized and polling-based INT packets as shown in Fig. 5. *Hop Count* is a one-byte field whose value tells how many hops that the INT packet has experienced, and each *INT Field* contains the telemetry data collected on one switch.

Each *INT Field* in a prioritized INT packet contains three subfields: *MapInfo*, *DeviceID* and *Metadata*. *MapInfo* contains one byte to indicate the type of the telemetry data whose value is encoded in the 2-byte *Metadata*, and *DeviceID* stores the unique 4-byte ID of the switch that encodes the *INT Field*. We design P4InfoSen-INT to support four types of telemetry data, *i.e.*, *Queue Time*, *Bandwidth*, *In Port* and *Out Port*, each of which corresponds to a value of *MapInfo*. Here, *Queue Time* is the queuing time experienced by the packet, *Bandwidth* is the current bandwidth usage of the packet's output port, and *In Port* and *Out Port* tell the IDs of the input and output ports of the packet, respectively. On the other hand, each *INT Field* in a polling-based INT packet consists of five subfields: *DeviceID*, *Out Port*, *In Port*, *Queue Time* and *Bandwidth*, since it might encode all types of telemetry data in an INT packet.

4) Collection of Bandwidth Usage: As division is not well supported by Tofino ASICs, each switch can hardly calculate bandwidth usage directly. Hence, we let each switch count the data volume through an output port within a fixed period Δt, as the port's bandwidth usage, as explained in *Algorithm* 1. We initialize two registers R_4 and $\widetilde{R}_1$ for time and counters, respectively, in *Lines* 1-2. Here, as $\widetilde{R}_1$ can store two counters, it is different from the $R_1 \sim R_3$ in Fig. 4 and R_4, each of which only stores one value. When a packet arrives, *Line* 4 gets the current system time and the packet's length in bytes. Then, if the current collection period has expired, *Line* 6 updates the values stored in R_4 and $\widetilde{R}_1$. Otherwise, *Line* 8 only adds the packet length to c_2 (*i.e.*, the counter in $\widetilde{R}_1$ for the accumulated date volume in the current period). Finally, *Line* 10 returns c_1 (*i.e.*, the counter in $\widetilde{R}_1$ for the accumulated date volume in the previous period) as the current bandwidth usage.

5) Design of Pipelines in Switch: To implement P4InfoSen-INT, we make each switch support prioritized INT and polling-based INT simultaneously. As for prioritized INT, the switch collects different types of telemetry data, finds the one that has the highest priority within each sampling cycle, and encodes it in a prioritized INT packet. As for polling-based INT, the switch only encodes all types of telemetry data in a polling-based INT packet directly. Fig. 6 shows the design of the stages in ingress and egress pipelines of a switch. As the operations in the two pipelines can be performed simultaneously for different packets, their tables can be arranged in same stages.

In the ingress pipeline, *Ipv4_Tbl* determines the output port of a packet, *Hash_Tbl* calculates a hash value based on the packet's 5-tuple (*i.e.*, its source and destination IP addresses, source and destination layer-4 ports, and protocol) to identify its flow, and *INT_Tbl* sets $flag_1$ to 1 if the flow has been selected to be monitored. If we have $flag_1 = 1$, three registers R_4, R_5 and R_6 are allocated to process the

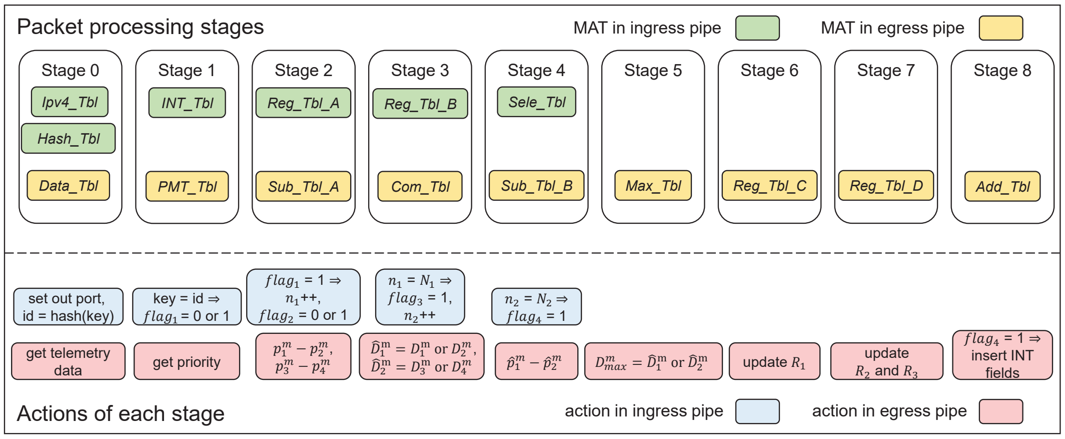

Packet processing stages
MAT in ingress pipe
MAT in egress pipe
Stage 0
Ipv4_Tbl
Hash_Tbl
Data_Tbl
Stage 1
INT_Tbl
PMT_Tbl
Stage 2
Reg_Tbl_A
Sub_Tbl_A
Stage 3
Reg_Tbl_B
Com_Tbl
Stage 4
Sele_Tbl
Sub_Tbl_B
Stage 5
Max_Tbl
Stage 6
Reg_Tbl_C
Stage 7
Reg_Tbl_D
Stage 8
Add_Tbl
set out port, id = hash(key)
key = id ⇒ $flag_1$ = 0 or 1
$flag_1 = 1 \Rightarrow$ n_1++, $flag_2$ = 0 or 1
$n_1 = N_1 \Rightarrow$ $flag_3 = 1$, n_2++
$n_2 = N_2 \Rightarrow$ $flag_4 = 1$
get telemetry data
get priority
$p_1^m - p_2^m$, $p_3^m - p_4^m$
$\widehat{D}_1^m = D_1^m$ or D_2^m, $\widehat{D}_2^m = D_3^m$ or D_4^m
$\widehat{p}_1^m - \widehat{p}_2^m$
$D_{max}^m = \widehat{D}_1^m$ or $\widehat{D}_2^m$
update R_1
update R_2 and R_3
$flag_4 = 1 \Rightarrow$ insert INT fields
Actions of each stage
action in ingress pipe
action in egress pipe

Fig. 6. Stage design for implementing P4InfoSen-INT in a Tofino-based hardware PDP switch.

Algorithm 2: Procedure of P4InfoSen-INT

```
   Input: INT sampling interval $N_1$, polling cycle $N_2$, and PMT.
1  initialize registers $R_1 = \{p\}$, $R_2 = \{v\}$, $R_3 = \{id\}$,
   $R_5 = \{n_1\}$ and $R_6 = \{n_2\}$;
2  $p = v = id = n_1 = n_2 = 0$;
3  while one packet of the monitored flow $f_m$ arrives do
4      apply Algorithm 1 for bandwidth collection;
5      get telemetry data and build $D^m$ by checking PMT;
6      find telemetry data $D^m_{\max}$ with the highest priority;
7      $n_1 = n_1 + 1$;
8      if $n_1 = 1$ then
9          $p = p^m_{\max}$, $v = v^m_{\max}$, $id = id^m_{\max}$;
10     else
11         if $p^m_{\max} \geq p$ then
12             $p = p^m_{\max}$, $v = v^m_{\max}$, $id = id^m_{\max}$;
13         end
14     end
15     if $n_1 = N_1$ then
16         mark the packet as an INT packet;
17         $n_1 = 0$, $n_2 = n_2 + 1$;
18         if $n_2 = N_2$ then
19             encode telemetry data in $D^m_1$, $D^m_2$, $D^m_3$ and $D^m_4$
               in the polling-based INT packet;
20             set the protocol field in IP header to 0xfe;
21             $n_2 = 0$;
22         else
23             encode telemetry data stored in $R_2$ and $R_3$ in the
               prioritized INT packet;
24             set the protocol field in IP header to 0xff;
25         end
26     end
27 end
```

packet. As explained in *Algorithm* 1, R_4 records the time when the current bandwidth collection period starts. R_5 stores the currently-accumulated number of packets in the flow. Upon receiving a packet, the operations on R_4 and R_5 are governed by *Reg_Tbl_A* and *Time_Tbl*, which increase the value of R_5 (n_1) by 1 and update R_4 according to *Algorithm* 1. Here, $flag_2$ indicates whether the current bandwidth collection period has expired or not. R_6 stores how many times that the prioritized INT has been performed on the flow (n_2), which is updated by *Reg_Tbl_B*. Finally, *Sele_Tbl* tells the egress pipeline to perform prioritized INT or polling-based INT based on $flag_4$.

In the egress pipeline, *Data_Tbl* gets the values of the four types of telemetry data. *PMT_Tbl* determines the priority of each piece of data by checking the PMT. Here, we define the *i*-th type of telemetry data of a monitored flow f_m as $D_i^m = \{v_i^m, p_i^m, id_i^m\}$, where v_i^m denotes the value of the telemetry data, and p_i^m and id_i^m are its priority and type ID, respectively. For each packet of flow f_m, a switch gets a set of telemetry data as $D^m = \{D_1^m, D_2^m, D_3^m, D_4^m\}$, denoting the related *Bandwidth*, *Queue Time*, *Out Port*, and *In Port*, respectively. Then, *Sub_Tbl_A*, *Com_Tbl_A* and *Com_Tbl_B* compare two types of telemetry data with Eq. (2), *i.e.*, applying Eq. (2) to D_1^m, D_2^m and D_3^m, D_4^m, to get $\widehat{D}_1^m$ and $\widehat{D}_2^m$, which are the data types with higher priorities and $\widehat{D}_i^m = \{\widehat{v}_i^m, \widehat{p}_i^m, \widehat{id}_i^m\}$.

$$\widehat{D}^m = \begin{cases} D_i^m, & p_i^m - p_j^m \geq 0, \\ D_j^m, & \text{otherwise.} \end{cases} \tag{2}$$

Next, *Sub_Tbl_B* and *Max_Tbl* compare $\widehat{D}_1^m$ and $\widehat{D}_2^m$ similarly to get the telemetry data $D^m_{\max} = \{v^m_{\max}, p^m_{\max}, id^m_{\max}\}$ whose priority is the highest among the four data types. *Reg_Tbl_C*, *Reg_Tbl_D* and *Reg_Tbl_E* read and update registers $R_1 \sim R_3$ if the priority of $D^m_{\max}$ is higher than that of the telemetry data currently stored in R_2. Finally, *Add_Tbl* selects to perform prioritized INT or polling-based INT based on the $flag_4$ from the ingress pipeline. Specifically, *Add_Tbl* performs prioritized INT with the values stored in registers R_2 and R_3 if we have $flag_4 = 0$, and it conducts polling-based INT, otherwise.

With the stage design in Fig. 6, the procedure of P4InfoSen-INT can be summarized with *Algorithm* 2, where N_1 is the INT sampling interval (*i.e.*, out of how many packets an INT packet is selected) and N_2 defines the cycle of polling-based INT (*i.e.*, out of how many INT packets a polling-based INT packet is selected). We have open-sourced the P4 codes for implementing *Algorithm* 2 on a Tofino-based switch in [46].

D. Control Plane Operation

The control plane overlooks switches in the network and performs global adjustments to further optimize the tradeoff between bandwidth overheads and monitoring accuracy of P4InfoSen-INT. Specifically, the control plane mainly handles two tasks: 1) adjusting the INT sampling interval N_1 of each monitored flow, and 2) updating the PMTs on switches for each monitored flow. In the following, we will briefly explain our design to handle the first task, while the optimization for the second task will be discuss in Section V.

For selective INT schemes, INT sampling interval is the key parameter to adjust the tradeoff between bandwidth overheads and monitoring accuracy [27, 28, 30]. However, to the best of our knowledge, the problem of how to select INT sampling interval adaptively is still under-explored. In our previous work [31], we proposed to adjust it based on the predicted peak bandwidth usage in the future, to avoid INT causing unnecessary congestions in a network. Nevertheless, the approach did not try to adjust INT sampling interval to optimize the tradeoff between bandwidth overheads and monitoring accuracy, and moreover, not all the traffic patterns are predictable. Therefore, in this work, we let the SDN controller cooperate with DAs to adjust INT sampling interval adaptively based on real-time collected telemetry data. Specifically, as shown in Fig. 2, the sampling rate arbiter in each DA analyzes and sends the telemetry data on bandwidth usage to the controller, which will update the sampling intervals in switches accordingly.

Algorithm 3: INT sampling interval adjustment

```
Input: Sampling interval $N_1$, bandwidth collection period $\Delta t$,
       threshold $\delta$.
1  $B^m_{last} = 0$;
2  while DA receives bandwidth data $b^m$ of flow $f_m$ do
3      calculate bandwidth usage $B^m = \frac{b^m \cdot 8}{\Delta t}$;
4      if $|B^m - B^m_{last}| \geq \delta$ then
5          get a new sampling interval $N_1'$ with Eqs. (3) and (4);
6          if $N_1' \neq N_1$ then
7              $N_1 = N_1'$, $B^m_{last} = B^m$;
8              update the sampling intervals in related switches;
9          end
10     end
11 end
```

To obtain an appropriate sampling interval, we first define the average INT header length $\overline{L}_{INT}$ of INT packets when they reach egress nodes as

$$\overline{L}_{INT} = \frac{(N_2 - 1) \cdot L_1 + L_2}{N_2}, \tag{3}$$

where L_1 and L_2 are the lengths of INT header in prioritized INT and polling-based INT packets, respectively, when the INT packets reach their last hops. Then, we get the new sampling interval N_1' for a monitored flow f_m as

$$N_1' = \left\lceil \frac{B^m \cdot \overline{L}_{INT}}{\eta \cdot (C^m - B^m) \cdot \overline{L_p}} \right\rceil, \tag{4}$$

where C^m denotes the maximum bandwidth capacity that f_m can use, $\eta \in (0, 1)$ is a preset parameter representing the ratio of the available bandwidth that can be used by INT, B^m is the bandwidth usage based on the historical telemetry data of f_m, and $\overline{L_p}$ is the average length of normal packets. Then, the bandwidth overheads introduced by P4InfoSen-INT will not make the flow use more bandwidth than that allocated to it, and the tradeoff between bandwidth overheads and monitoring accuracy of P4InfoSen-INT can be adjusted by changing η.

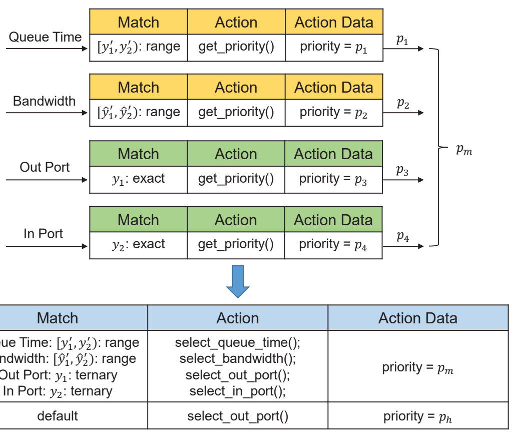

Queue Time
Bandwidth
Out Port
In Port
Match | Action | Action Data
$[y_1', y_2')$: range | get_priority() | priority = p_1
$[\hat{y}_1', \hat{y}_2')$: range | get_priority() | priority = p_2
y_1: exact | get_priority() | priority = p_3
y_2: exact | get_priority() | priority = p_4
p_1
p_2
p_3
p_4
p_m
Match | Action | Action Data
Queue Time: $[y_1', y_2')$: range; Bandwidth: $[\hat{y}_1', \hat{y}_2')$: range; Out Port: y_1: ternary; In Port: y_2: ternary | select_queue_time(); select_bandwidth(); select_out_port(); select_in_port(); | priority = p_m
default | select_out_port() | priority = p_h

Fig. 7. Example on merging MATs for different types of telemetry data.

Specifically, a larger η leads to a smaller sampling interval and thus better monitoring accuracy, and *vice versa*.

The procedure of sampling interval adjustment is shown in *Algorithm* 3. *Line* 3 calculates the actual bandwidth usage of a monitored flow f_m with the collected telemetry data on *Bandwidth*. Then, *Lines* 4-10 determine whether to update the sampling intervals of the flow in related switches. Specifically, to avoid updating switches too frequently, the controller gets a new sampling interval based on δ (*Line* 4), and will only update the sampling intervals when necessary (*Line* 6-9).

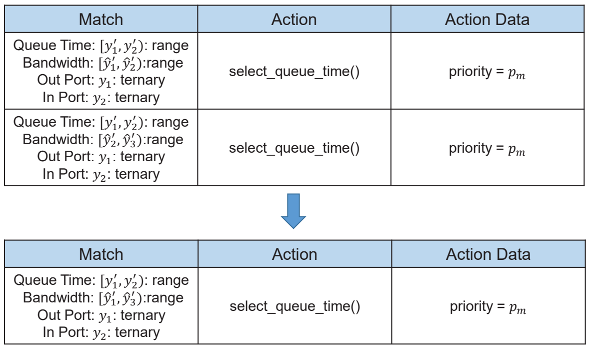
IV. Optimization of Data Plane Implementation

Although the design in Fig. 6 can realize P4InfoSen-INT on a hardware switch, it will occupy too many stages there and thus needs to be further optimized for better scalability. Specifically, using multiple MATs to find the telemetry data with the highest priority is resource consuming. In this section, we propose a novel scheme to merge MATs such that the related operations can be realized with less stages in a switch.

A. Merging of Tables

In Fig. 6, the process of finding the telemetry data with the highest priority occupies four stages. This can be optimized by redesigning the PMT. In Section III, we build the PMT by customizing an MAT for each type of telemetry data (as shown in Fig. 3), and then by checking the MATs, a switch can find the telemetry data with the highest priority. This can actually be simplified through table merging, *i.e.*, designing an MAT to handle all types of telemetry data. Fig. 7 shows an example, where the switch looks up MATs to get the priorities of telemetry data and then find the one whose priority p_m is the highest. This can also be done with a merged priority mapping table (MPMT) that takes the four types of telemetry data as match keys, *i.e.*, there is no need to perform repeated comparisons anymore and *Stages* 2-5 in Fig. 6 can be released.

To build MPMT based on the original PMT, we introduce an entry set for each type of telemetry data. For *Bandwidth* or *Queue Time*, the entry set can be defined as $E_i^S = \{e^s_{i,1}, e^s_{i,2}, \cdots, e^s_{i,n}\}$, where we have $e^s_{i,j} = \{v^l_{i,j}, v^u_{i,j}, a, p_{i,j}\}$, and $[v^l_{i,j}, v^u_{i,j})$ denotes the range of the match key, a denotes the action (*i.e.*, "get_priority" in Fig. 3(a)), and $p_{i,j}$



Match	Action	Action Data
Queue Time: $[y'_1, y'_2)$: range Bandwidth: $[\hat{y}'_1, \hat{y}'_2)$:range Out Port: y_1: ternary In Port: y_2: ternary	select_queue_time()	priority = p_m
Queue Time: $[y'_1, y'_2)$: range Bandwidth: $[\hat{y}'_2, \hat{y}'_3)$:range Out Port: y_1: ternary In Port: y_2: ternary	select_queue_time()	priority = p_m

Match	Action	Action Data
Queue Time: $[y'_1, y'_2)$: range Bandwidth: $[\hat{y}'_1, \hat{y}'_3)$:range Out Port: y_1: ternary In Port: y_2: ternary	select_queue_time()	priority = p_m

Fig. 8. An example of merging entries of MPMT.

is its action data (*i.e.*, priority). For *In Port* or *Out Port*, we only need to define one entry $e_i^{ns} = \{v_i, a, p_i^*\}$, where v_i is the match key value that denotes the expected port number, and p_i^* is its action data. Then, the entry set of MPMT becomes $E^P = \{e_1^p, e_2^p, \cdots, e_n^p\}$. Here, we have $e_j^p = \{v_1, m_1, v_2, m_2, v_{1,k_j}^l, v_{1,k_j}^u, v_{2,k_j}^l, v_{2,k_j}^u, a_j, mp_j, p_{m_j}\}$, where v_1, v_2, $[v_{1,k_j}^u, v_{1,k_j}^d)$, and $[v_{2,k_j}^u, v_{2,k_j}^d)$ are the values and ranges of match keys, respectively, m_1 and m_2 are the "masks" of v_1 and v_2, respectively, a_j can be set to one of the actions in the merged table in Fig. 7, mp_j denotes the "match priority" of the entry, and p_{m_j} is the action data that denotes the highest priority of the telemetry data. E^P can be obtained with *Algorithm* 4, which will run in the control plane.

Algorithm 4: Table merging

```
    Input: Entry sets $E_1^S$ and $E_2^S$, entries $e_1^{ns}$ and $e_2^{ns}$, and $p_h$
    Output: Entry set of MPMT $E^P$
1   $E^P = \emptyset$, $m_1 = 0xffff$, $m_2 = 0x0000$;
2   get $v_1$, $v_2$, $p_1^*$ and $p_2^*$ from entries $e_1^{ns}$ and $e_2^{ns}$;
3   for each entry $e_{1,i}^s$ in $E_1^S$ do
4       get $v_{1,i}^u$, $v_{1,i}^d$ and $p_{1,i}$ from entry $e_{2,i}^s$;
5       for each entry $e_{2,j}^s$ in $E_2^S$ do
6           get $v_{2,j}^u$, $v_{2,j}^d$ and $p_{2,j}$ from entry $e_{1,j}^s$;
7           get the highest priority $p_m$ from $p_1^*$, $p_2^*$, $p_{1,i}$ and $p_{2,j}$;
8           set $a$ based on $p_m$;
9           $e^p = \{v_1, m_1, v_2, m_1, v_{1,i}^u, v_{1,i}^d, v_{2,j}^u, v_{2,j}^d, a, 0, p_m\}$;
10          store $e^p$ in $E^P$;
11      end
12  end
13  $e^{p_1} = \{v_1, m_1, 0, m_2, 0, m_1, 0, m_1, 3, 1, p_h - 1\}$;
14  $e^{p_2} = \{0, m_2, v_2, m_1, 0, m_1, 0, m_1, 4, 1, p_h\}$;
15  store $e^{p_1}$ and $e^{p_2}$ in $E^P$;
16  set action of default entry according to merged table in Fig. 7;
17  return entry set $E^P$;
```

In *Algorithm* 4, the for-loop in *Lines* 3-12 traverses all the entries to get their match key values and action data. *Lines* 3-8 get the highest priority of the telemetry data and determine the corresponding action. For example, if $p_{1,i}$ is the highest one, a is set to 1. *Lines* 9-10 generate an entry of the merged table and store it in E^P. Note that, for the *exact* type MAT in Fig. 3(a), we can use a default entry to match the unexpected port numbers. However, in the merged table, one default entry might not be enough due to the existence of multiple match keys. Hence, *Lines* 13-16 are introduced to generate the entries

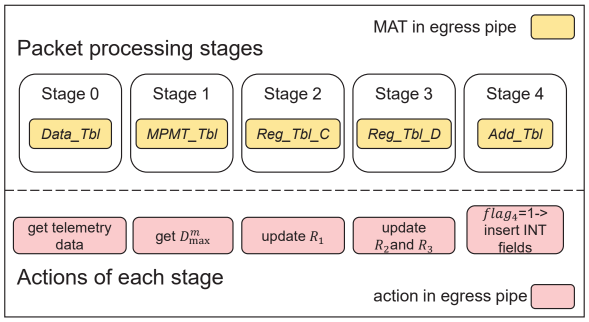


Fig. 9. Implementation of P4InfoSen-INT in a switch with MPMT.

for the unexpected port numbers. The *ternary* match type enables us to set a mask to select the effective bit(s) of match values. Specifically, we set the masks of the entries for *In Port* and *Out Port* to *0xffff* and *0x0000* to match to expected and unexpected port numbers, respectively. However, the telemetry data that can match to the entry generated by *Line* 9 will also match to the entries generated by *Lines* 13-14, and thus we use the match priority to determine which entry should be selected. Specifically, when the data can match to multiple entries, the entries are prioritized according to their match priorities. The default entry is used to deal with the case where *In Port* and *Out Port* are both unexpected values.

B. Merging of Table Entries

Next, we try to merge the entries in the MPMT to further save memory resources in a switch. Fig. 8 provides an example on entry merging, where two entries with the same action(s) are merged. Moreover, a merged entry may be further merged with other entries. Hence, a well-designed approach for merging the entries in the MPMT will allow P4InfoSen-INT to monitor more flows, further improving the scalability of our proposal. However, as the approach is too specific on the hardware-related implementation of P4InfoSen-INT, we omit it here but explain it as the *Algorithm* 5 in Appendix A.

After optimizing the PMT with MPMT, we design the egress pipeline of P4InfoSen-INT in a switch as that in Fig. 9, where *MPMT_Tbl* can get $D_{\max}^m$ directly without doing any comparison. Therefore, stages can be effectively saved over the scheme in Fig. 6. Note that, the scheme in Fig. 9 is just the ideal case, but due to the limited memory resources in each stage, *MPMT_Tbl* might occupy multiple stages due to the longer match key and more entries in the merged table.

V. Status Update in Control Plane

To ensure that P4InfoSen-INT can adapt to dynamic network changes, the control plane needs to update the PMT on each switch to achieve flexible adjustments of the INT scheme of a monitored flow. As we have explained in Section III-B, each PMT stores the priorities of the telemetry data of a monitored flow, which are obtained by calculating the information content of each data value with Eq. (1). However, as Eq. (1) actually determines the information content of a data value based on the statistical distribution of the data, which is normally unknown at the initialization of a P4InfoSen-INT

Data Type	Probability			
Queue Time	$r_{1,1}$: $\theta_{1,1}$	$r_{1,2}$: $\theta_{1,2}$	$\cdots$	r_{1,M_1}: θ_{1,M_1}
Bandwidth	$r_{2,1}$: $\theta_{2,1}$	$r_{2,2}$: $\theta_{2,2}$	$\cdots$	r_{1,M_2}: θ_{2,M_2}
Out Port	$r_{3,1}$: $\theta_{3,1}$		$r_{3,2}$: $\theta_{3,2}$	
In Port	$r_{4,1}$: $\theta_{4,1}$		$r_{4,2}$: $\theta_{4,2}$	

Fig. 10. Layout of PRMT.

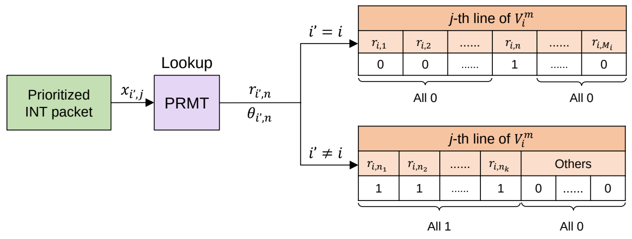
system, we can only initialize each PMT with an estimated distribution (*e.g.*, the uniform distribution) and update it according to the distribution of collected telemetry data during network operation [31]. Then, the question is how to update each PMT to ensure that the priorities stored in it can converge to represent the actual distribution of the PMT's telemetry data.

With P4InfoSen-INT, a switch checks the PMT for a monitored flow and inserts the telemetry data whose priority is the highest accordingly. Therefore, the more accurate the PMT is, the more effective P4InfoSen-INT is to balance the tradeoff between bandwidth overheads and monitoring accuracy. The update of each PMT is taken care of by the control plane, by building and maintaining a probability mapping table (PRMT) for each PMT. Fig. 10 shows the layout of a PRMT, which stores the probability distribution of each type of telemetry data, and the probability distribution is estimated by checking the telemetry data collected at a DA. Specifically, each row of the PRMT corresponds to a type of telemetry data, and we denote the row for the i-th type of telemetry data as $S_i = \{s_{i,1}, s_{i,2}, \cdots, s_{i,M_i}\}$, where each element in the row is $s_{i,n} = \{r_{i,n}, \theta_{i,n}\}$. $r_{i,n}$ is a range or value of the i-th type of telemetry data and $\theta_{i,n}$ is its probability. $\theta_{i,n}$ is initialized with a preset value, and then, according to the analysis done by the DA, its value is updated after the collection of every N_3 INT packets (an update cycle). Hence, the priorities of the telemetry data in PMT are updated based on the PRMT.

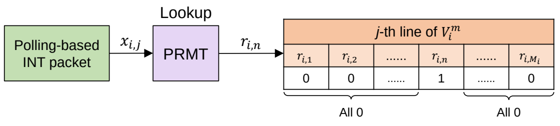
The update of PRMT can be regarded as a parameter estimation problem. We assume that $\alpha_{i,n}^t$ is the true probability of the i-th type of telemetry data falling in $r_{i,n}$ in the t-th update cycle, which is the parameter that we need to estimate, and thus the distribution of the i-th type of telemetry data in the t-th update cycle can be denoted as $\mathcal{A}_i^t = \{\alpha_{i,1}^t, \alpha_{i,2}^t, \cdots \alpha_{i,M_i}^t\}$. As the DA receives N_3 INT packets in each update cycle, we denote the telemetry data that the DA collects in the update cycle for a hop as $X = \{x_1, x_2, \cdots x_{N_3}\}$, each of which is a data set that contains four types of or one type of data value(s) for a polling-based or prioritized INT packet, respectively. For the data set x_j, we define the probability that it can be collected under the distribution $\mathbf{A}^t = \{\mathcal{A}_i^t, i \in [1,4]\}$ as $P\left(x_j \mid \mathbf{A}^t\right)$. Approximately, we can assume that the data collection of each INT packet is independent, and thus the likelihood function of $X = \{x_1, x_2, \cdots x_{N_3}\}$ is

$$P(X \mid \mathbf{A}^t) = \prod_{j=1}^{N_3} P\left(x_j \mid \mathbf{A}^t\right). \tag{5}$$

At the end of each update cycle t, X is known and then we need to use it to calculate the maximum likelihood estimation (MLE) of $\mathbf{A}^t$ according to Eq. (5), which denotes



(a) The j-th INT packet is a prioritized INT packet



(b) The j-th INT packet is a polling-based INT packet

Fig. 11. Procedure for updating matrix $\mathbf{V}_i^m$ after receiving an INT packet.

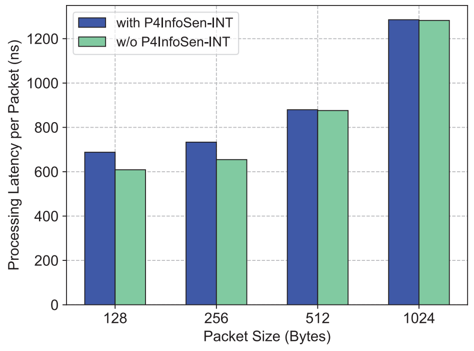
the probability distribution of four types of telemetry data in the current update cycle, *i.e.*,:

$$\hat{\mathbf{A}}^t = \underset{\mathbf{A}^t}{\operatorname{argmax}} \left[L(\mathbf{A}^t)\right], \tag{6}$$

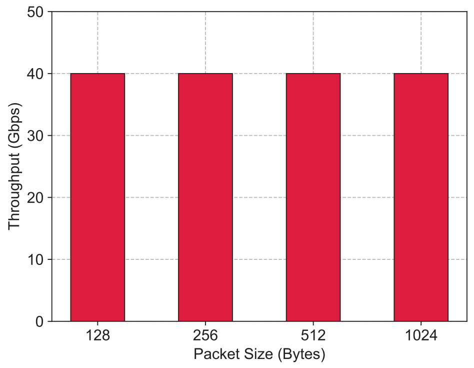
where we define $L(\mathbf{A}^t) = \log\left[P\left(X \mid \mathbf{A}^t\right)\right]$ because the values of $P\left(X \mid \mathbf{A}^t\right)$ can be hard to differentiate and $\log(\cdot)$ is a monotonically increasing function to amplify the difference. In addition, since $\mathbf{A}^t$ is a multi-dimensional variable, it is still complicated to express $P(X \mid \mathbf{A}^t)$ and calculate $\hat{\mathbf{A}}^t$ in Eq. (6). Therefore, we try to introduce extra parameters to denote the true values/ranges of all types of telemetry data at the sampling of an INT packet, which are unknown for us. Specifically, we define a set of hidden variables $Z_i = \{z_{i,1}, z_{i,2}, \cdots z_{i,N_3}\}$ for the i-th type of telemetry data, where $z_{i,j} \in [1, M_i]$ represents the index of the column that the value/range of the i-th type of telemetry data falls in at the sampling of the j-th INT packet. Thus the hidden variables of four types of telemetry data at the sampling of the j-th INT packet can be denoted as a quadruple $\mathbf{z_j} = \{z_{1,j}, z_{2,j}, z_{3,j}, z_{4,j}\}$. For example, when the first INT packet of an update cycle arrives at a switch, the value of the first type of telemetry data falls in the range of the second column $r_{1,2}$, we have $z_{1,1} = 2$. We assume that the distribution of hidden variables for all the types of telemetry data are independent of each other.

In order to estimate the distribution of hidden variables conveniently, we introduce a matrix for each type of telemetry data of a monitored flow f_m, *i.e.*, for the i-th type of telemetry data, it gets an $N_3 \times M_i$ matrix $\mathbf{V}_i^m$, to record its possible values or ranges. Thus we can get a matrix set $\mathbf{V}^m = \{\mathbf{V}_1^m, \mathbf{V}_2^m, \mathbf{V}_3^m, \mathbf{V}_4^m\}$, where $\mathbf{V}_1^m$, $\mathbf{V}_2^m$, $\mathbf{V}_3^m$ and $\mathbf{V}_4^m$ denote the matrices of the telemetry data of *Queue Time*, *Bandwidth*, *Out Port*, and *In Port* in Fig. 10, respectively.

Fig. 11 explains how the DA updates the the j-th row of $\mathbf{V}_i^m$ when it receives the j-th INT packet in an update cycle. If the INT packet is a prioritized INT packet and we assume that it contains the i'-th type of telemetry data about a hop, the DA can denote the data value as $x_{i',j}$, find the $z_{i',j} = n$ for it from PRMT, and set the value of element $\mathbf{V}_{i'}^m(j, n)$ to 1. For the telemetry data other than the i'-th one, the information content of its value about the hop should be smaller than that of $x_{i',j}$ according to the principle of P4InfoSen-INT, since it is



(a) Processing latency per packet



(b) Packet processing throughput

Fig. 12. Packet processing performance of switch with P4InfoSen-INT.

not encoded in the INT packet. Therefore, the DA can derive the possible values or ranges of the type of the telemetry data from PRMT and update the j-th row of $\mathbf{V}_i^m$ as

$$\mathbf{V}_i^m(j,n) = \begin{cases} 1, & \theta_{i,n} > \theta_{i',n},\ \forall n \in [1, M_i], \\ 0, & \text{otherwise}. \end{cases} \tag{7}$$

On the other hand, if the INT packet is a polling-based INT packet, the DA can get all the four types of telemetry data $\{x_{1,j}, x_{2,j}, x_{3,j}, x_{4,j}\}$, and the corresponding $\{z_{i,j}\}$, and the matrices $\mathbf{V}^m$ are updated as shown in Fig. 11(b). By updating $\mathbf{V}^m$, we can mark the possible values/ranges of all the types of telemetry data at the sampling of an INT packet, thereby estimating the distribution of hidden variables.

With the hidden variables, we get the probability that data set x_j can be obtained under distribution $\mathbf{A}^t = \{\mathcal{A}_i^t, i \in [1,4]\}$

$$\begin{aligned} P\left(x_j \mid \mathbf{A}^t\right) &= \sum_{z_{1,j}=1}^{M_1} \cdots \sum_{z_{4,j}=1}^{M_4} P\left(x_j, z_{1,j}, \cdots, z_{4,j} \mid \mathbf{A}^t\right) \\ &= \sum_{\mathbf{z_j}} P(x_j, \mathbf{z_j} \mid \mathbf{A}^t), \end{aligned} \tag{8}$$

where $P\left(x_j, \mathbf{z_j} \mid \mathbf{A}^t\right)$ is the probability of getting x_j and $\mathbf{z_j}$.

Then, to get $\hat{\mathbf{A}}^t$ in Eq. (6) and update the PRMT and PMT, we propose an update approach based on the expectation maximization (EM) algorithm, which is effective for parameter estimation with hidden variables [47]. The detailed procedure of the approach is explained in *Algorithm* 6 in Appendix B.

VI. Experimental Demonstrations

In this section, we show real-world experimental demonstrations to verify the effectiveness of our P4InfoSen-INT system.

A. Experimental Setup

We implement our proposed P4InfoSen-INT system in a real-world testbed with the configuration as shown in Fig. 1. The data plane includes two end hosts, six Tofino-based switches, and a DA, which are connected through 10GbE ports. Each end host is emulated with a software traffic generator/analyzer [48] running on a Linux server, which can generate/receive packets at a data-rate up to 10 Gbps. The DA is home-made and also runs on a Linux server, and it can achieve a packet processing rate of over 2 Mpps. The experiments let *Host* 1 send dynamic traffic flows whose data-rates range within $[2, 10]$ Gbps according to the traces[2] in [49], to *Host* 2, and the flows are routed through switches *a*-*b*-*c*-*d* in sequence. In the control plane, we have a controller running on a server to manage switches with P4Runtime [50].

The experiments evaluate P4InfoSen-INT against two representative benchmarks, *i.e.*, the PINT in [30] and the adaptive INT (AINT) in [44]. PINT randomly selects a type of telemetry data to insert in each INT packet according to preset probabilities, and AINT polls each type of telemetry data to insert in INT packets. For fair comparisons, we make PINT and AINT use the packet formats of P4InfoSen-INT in Fig. 5.

B. Throughput of Switch with P4InfoSen-INT

We first measure the packet processing latency and throughput of a switch implemented with P4InfoSen-INT, to verify that our implementation of P4InfoSen-INT does not cause noticeable degradation on packet processing performance. Specifically, we send 40 Gbps traffic with different packet sizes to a switch with P4InfoSen-INT, and Fig. 12 shows the results averaged over more than 2 million packets. In Fig. 12(a), we can see that the processing latency per packet in the switch with P4InfoSen-INT is still extremely low even though it is larger than that in the switch without P4InfoSen-INT. The results on throughput in Fig. 12(b) further prove the packet processing performance of P4InfoSen-INT, as the line-rate of 40 Gbps can be achieved for all the tested packet sizes.

C. Resource Consumptions of Implementations

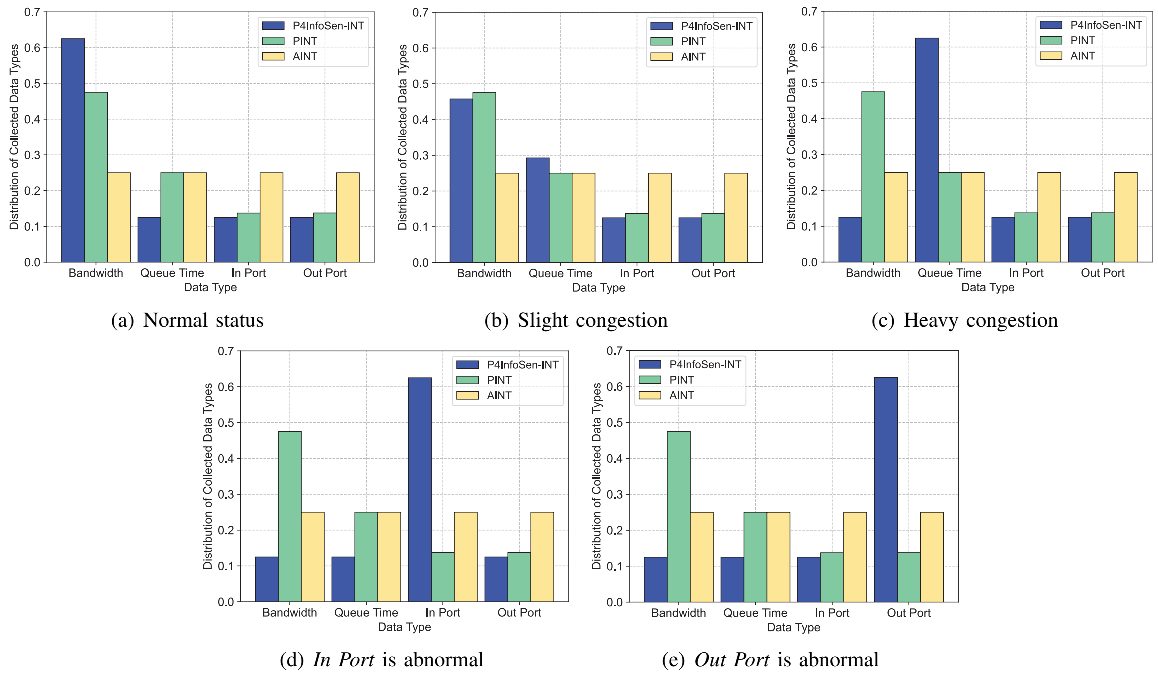
Table II summarizes the resource usages of our implementations of PINT, AINT, and various versions of P4InfoSen-INT in a Tofino-based switch. Here, the SRAM/TCAM usages denote the number of occupied SRAM/TCAM blocks, each of which contains a fixed amount of corresponding type of memory resources. Note that, PINT needs to leverage an MAT with *range* match type, which uses TCAM, to implement the probabilistic telemetry data collection, while AINT needs a counter and an MAT with *exact* match type, which uses SRAM, to poll various types of telemetry data. Hence, PINT consumes more TCAM while AINT uses more SRAM.

As P4InfoSen-INT has more complex operations, it uses more SRAM blocks, TCAM blocks, and stateful arithmetic and logic units (ALUs) than PINT and AINT. We can see

[2]The traces were published online and collected in realistic networks of research institutions, enterprises and Internet service providers. Specifically, 14 reasonable sources were considered, each of which records traffic traces in real time with a collection interval of a few minutes.

TABLE II
RESOURCE CONSUMPTION IN A TOFINO-BASED SWITCH

Scheme	Stages	SRAM Blocks	TCAM Blocks	Stateful ALUs	PHV Containers		
					8-bit	16-bit	32-bit
PINT	5	16	4	4	17	21	8
AINT	5	21	2	5	17	20	8
P4InfoSen-INT w/o TM or TEM	9	31	9	7	26	24	8
P4InfoSen-INT with TM	7	24	59	7	14	23	8
P4InfoSen-INT with TM and TEM	5	22	26	7	14	23	8



(a) Normal status (b) Slight congestion (c) Heavy congestion

(d) *In Port* is abnormal (e) *Out Port* is abnormal

Fig. 13. Distributions of telemetry data types collected by INT schemes.

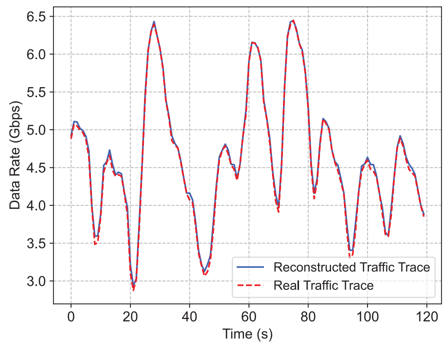


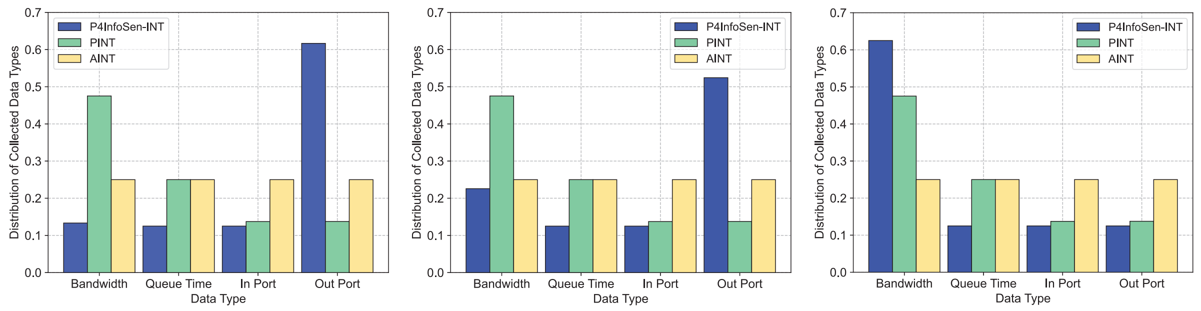
Fig. 14. Traffic trace reconstructed with data collected by P4InfoSen-INT.

that without the optimizations by TM and TEM algorithms, P4InfoSen-INT occupies 9 stages and more SRAM/TCAM blocks and PHV containers, affecting the scheme's scalability especially when it needs to be implemented with other network functions. This justifies the necessity of TM and TEM algorithms. As for various versions of P4InfoSen-INT, although the P4InfoSen-INT with TM can reduce the usage of stages, SRAM and PHV containers, it significantly increases the usage of TCAM since the MPMT uses longer match keys and more table entries. The issue can be relieved by the TEM algorithm as it reduces the number of table entries in the MPMT.

D. Feature Validation

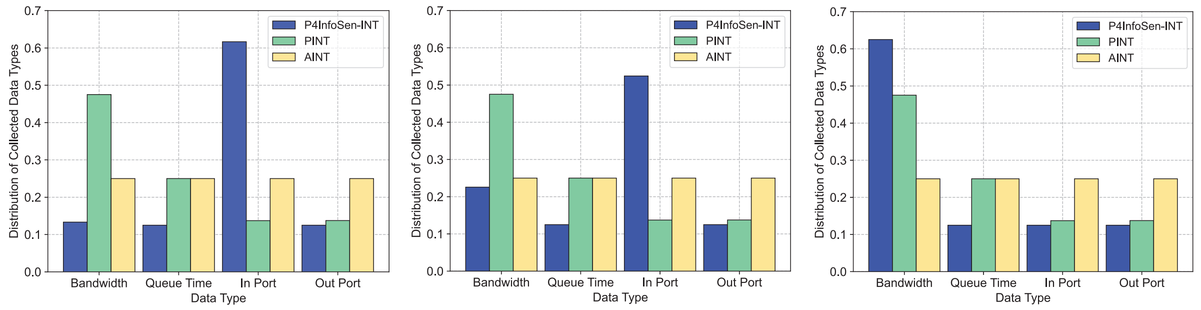
We then conduct experiments to show the feature of telemetry data collection of PINT, AINT and P4InfoSen-INT. Here, we set the INT sampling rate of the schemes to be the same such that they encounter the same INT bandwidth overheads. The distributions of telemetry data types collected by the INT schemes under normal network status are shown in Fig. 13(a). It can be seen that PINT prioritizes *Bandwidth* and *Queue Time* as these two types of telemetry data is time-variant while the data of *Out Port* and *In Port* is constant when the network status is normal. Meanwhile, compared with PINT, P4InfoSen-INT collects more data on *Bandwidth*. This is because the data of *Queue Time* does not change as significantly as that of *Bandwidth* under normal network status, and thus according to the principle of P4InfoSen-INT, the data samples of *Bandwidth* possess larger information contents and should be given higher priorities for telemetry data collection.

Next, we introduce anomalies regarding *Queue Time*, *In Port* and *Out Port* and redo the experiments, and Fig. 13(b)-13(e) show the distributions of collected telemetry data types, respectively. For each anomaly, we can see that P4InfoSen-INT accurately selects the abnormal data type to focus more on. This is because the values/ranges of the abnormal data type appear rarely in historical data and thus lead to relatively large information content according to Eq. (1). On the other hand,



(a) Distribution when load balancing begins (b) Distribution when PMT is being updated (c) Distribution when PMT has been updated

Fig. 15. Distributions of telemetry data types collected by ingress switch when the network starts to perform load balancing.



(a) Distribution when load balancing begins (b) Distribution when PMT is being updated (c) Distribution when PMT has been updated

Fig. 16. Distributions of telemetry data types collected by egress switch when the network starts to perform load balancing.

as PINT collects telemetry data based on preset probabilities, it still prioritizes *Bandwidth* and *Queue Time* as in Fig. 13(a), while AINT always treats all the data types equally. The results in Fig. 13 verify that P4InfoSen-INT can make smart and local decisions to adjust its data collection scheme adaptively.

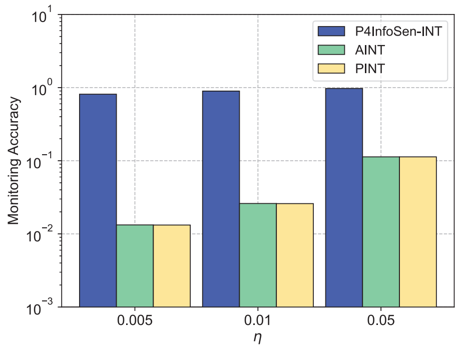
Then, we reconstruct the traffic trace of the monitored flow based on the data on *Bandwidth* collected by P4InfoSen-INT when *Out Port* is abnormal, and the results are plotted in Fig. 14. We observe that although the data on *Bandwidth* is collected much less frequently than the case under normal network status, the reconstructed trace still approximates the real one well. This verifies the effectiveness of the coexistence of prioritized and polling-based INT and *Algorithm* 1.

Finally, we test the scenario where the network starts to do load balancing after sending a flow for a while. Specifically, the flow is first routed over the path a-b-c-d in Fig. 1 only, and when the load balancing starts, its packets are routed over the paths a-b-c-d and a-f-e-d alternately. We plot the distributions of telemetry data types collected by the ingress and egress switches (switches a and d, respectively) in Figs. 15 and 16, respectively. Figs. 15(a) and 16(a) show the distributions when the load balancing begins, which indicate that the ingress and egress switches respectively collect more *Out Port* and *In Port* telemetry data. This is because when the load balancing has just started, the new output/input port at the ingress/egress switch is unexpected from the perspective of the switch's current PMT. Then, when the load balancing continues, the control plane obtains new telemetry data distributions from the DA and updates the PMTs on the ingress and egress switches accordingly. Therefore, although the ingress or egress switch still pays more attention to *Out Port* or *In Port*, respectively, the proportion of *Out Port* or *In Port* among all the telemetry data types decreases because the corresponding PMT is being updated (as shown in Figs. 15(b) and 16(b)). Next, after the PMTs have been updated, they can capture the changes of *Out Port* or *In Port* on ingress or egress switch accurately, and Figs. 15(c) and 16(c) indicate that the distributions of telemetry data types become similar to that in Fig. 13(a), which was collected before the load balancing starts. The results in Figs. 15 and 16 further confirm that P4InfoSen-INT allows each switch to make smart and local decisions for adaptive flow monitoring.

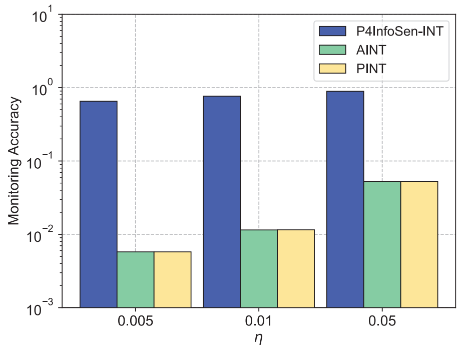
E. Performance Benchmarking

To further demonstrate the effectiveness of INT sampling rate adjustment in P4InfoSen-INT based on *Algorithm* 3, we introduce anomalies regarding *Queue Time*, set η in Eq. (4) as different values to change the bandwidth allocated for P4InfoSen-INT, and compute the percentage of collected abnormal data on *Queue Time*, which can be treated as monitoring accuracy. We still compare P4InfoSen-INT with PINT and AINT, and also consider the scheme of adjusting INT sampling rate based on peak bandwidth usage [31, 45].

The results on monitoring accuracy and bandwidth overheads are shown in Figs. 17 and 18, respectively. In Fig. 17, we can see that the monitoring accuracy increases with the bandwidth allocated to INT, and P4InfoSen-INT achieves higher monitoring accuracy than PINT and AINT. This is because P4InfoSen-INT adjusts its telemetry data collection scheme adaptively according to the information contents of collected data values, and thus can balance the tradeoff between monitoring accuracy and bandwidth overheads the best.

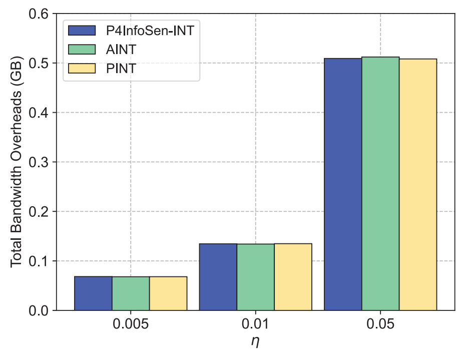


(a) Adjusting sampling rate based on *Algorithm* 3

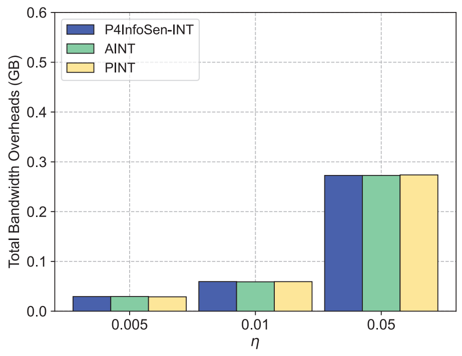


(b) Adjusting sampling rate based on peak bandwidth

Fig. 17. Results on monitoring accuracy.



(a) Adjusting sampling rate based on *Algorithm* 3



(b) Adjusting sampling rate based on peak bandwidth

Fig. 18. Results on bandwidth overheads.

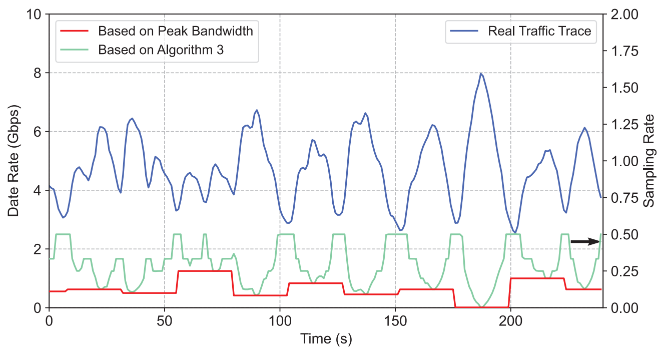


Fig. 19. Sampling rate of P4InfoSen-INT.

Meanwhile, compared with adjusting INT sampling rate based on peak bandwidth usage, the method based on *Algorithm* 3 provides higher monitoring accuracy. As *Algorithm* 3 can use more available bandwidth for INT without exceeding the capacity allocated to a monitored flow, the method based on it provides more bandwidth overheads in Fig. 18.

We average the sampling rates of P4InfoSen-INT in each second and show the results in Fig. 19, when different INT sampling adjustment schemes are used. We can see that compared with that from the method based on *Algorithm* 3, the sampling rate adjusted based on peak bandwidth usage is much smaller and change less frequently, which explains why its monitoring accuracy is smaller in Fig. 17(b). We also show the bandwidth usage on the link between switches c and d when P4InfoSen-INT adjusts the INT sampling rate based on the two schemes with $\eta = 0.05$ in Fig. 20. Compared with the scheme that adjusts the sampling rate based on peak bandwidth usage, the method based on *Algorithm* 3 does consume more bandwidth overheads, but it does not make the total bandwidth usage exceed the allocated capacity. Hence, *Algorithm* 3 can adjust the INT sampling rate to achieve better monitoring accuracy without violating the bandwidth constraint.

Finally, we conduct experiments to further verify the effectiveness of TEM algorithm. Specifically, we first generate different numbers of monitored flows to get the original entries for them in PMT without TM or TEM, then apply TM algorithm to get the entries in MPMT, and finally use TEM algorithm to further optimize the entries in MPMT. The results are shown in Fig. 21, where each data point is obtained by averaging the results from $1,000$ independent experiments. As expected, it takes more table entries to monitor more flows with P4InfoSen-INT. Compared with the original case without TM or TEM (the green curve), the one with TM but without TEM (the blue curve) does increase the number of entries significantly. However, the table entry increase can be effectively compensated with TEM, making the results of the case with TM and TEM (the red curve) similar to those of the original case. Therefore, the results in Fig. 21 verify that TEM algorithm can effectively improve the scalability of the implementation of P4InfoSen-INT in Tofino-based switches.

VII. Conclusion

In this paper, we implemented and optimized P4InfoSen-INT, which is an information-sensitive INT system based on P4-based hardware switches that are built with Tofino ASICs, and demonstrated that the proposed system could leverage agile local decisions on switches to achieve adaptive,

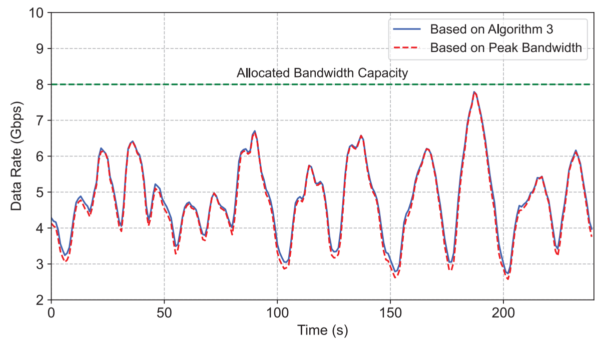


Fig. 20. Bandwidth usage of P4InfoSen-INT.

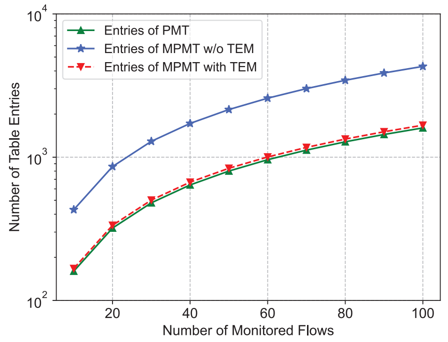


Fig. 21. Table entries in MPMT *versus* number of monitored flows.

efficient and real-time network monitoring. We first leveraged MATs in switches to design a PMT for denoting the priorities of different types of telemetry data, which were set based on the information content of telemetry data. Then, we designed the operation principle of P4InfoSen-INT such that it can be realized with the stages in a switch, and also proposed algorithms to adjust the INT sampling rate and PMTs adaptively according to the network status. Next, to improve the scalability of P4InfoSen-INT, we designed the TM and TEM algorithms, to reduce the stage usage of P4InfoSen-INT effectively. Our implementation of P4InfoSen-INT was demonstrated experimentally in a real-world network testbed, and evaluated against two representative benchmarks, showing that P4InfoSen-INT was more adaptive to network changes than the benchmarks, and achieved the best tradeoff between monitoring accuracy and bandwidth overheads.

Acknowledgments

This work was supported by the NSFC project 62371432.

Appendix A
Approach for Merging Table Entries in MPMT

Algorithm 5 explains our proposed approach to merge the entries in the MPMT to further save the memory resources in a switch. *Line* 1 is for the initialization. The for-loop of *Lines* 2-6 finds the entries in MPMT E^P whose action data is different, and insert their data and actions in sets P and A, respectively. Then, we merge the entries whose action data is the same with *Lines* 7-32. *Lines* 8-12 select the entries whose action data is the same and put them in set E. Then, *Lines* 13-30 merge the entries in E based on their actions. Specifically, if the actions of these entries are to select *Queue Time*, we determine whether two entries can be merged based on their range keys of *Bandwidth* (*Lines* 16-21). Otherwise, we merge the entries based on their range keys of *Queue Time* (*Lines* 22-30). If an entry cannot be merged with others, we insert it in set ME^P (*Line* 31). Finally, *Line* 33 returns the result.

Algorithm 5: Table entry merging

```
Input: Entry sets in $E^P$
Output: Optimized entry sets in $ME^P$
1  $ME^P = \emptyset$, $E = \emptyset$, $P = \emptyset$, $A = \emptyset$, $L = 0$, $g = 2$;
2  for each entry $e_i^p$ in $E^P$ do
3      if $p_i \notin P$ then
4          inset $p_i$, $a_i$ and $e_i^p$ in $P$, $A$ and $ME^P$, respectively;
5      end
6  end
7  for each $\{p_i, a_i\}$ in $P$ and $A$ do
8      for each entry $e_j^p$ in $E^P$ do
9          if $p_j = p_i$ then
10             insert $e_j^p$ in $E$;
11         end
12     end
13     $L = |E|$, $e_1^p = e_1$;
14     while $g \leq L$ do
15         $e_2^p = e_g$;
16         if $a_i = 1$ then
17             if $v_{2,k_1}^u = v_{2,k_2}^l$ then
18                 $v_{2,k_1}^u = v_{2,k_2}^u$;
19             else
20                 insert $e_1^p$ in $ME^P$, $e_1^p = e_2^p$;
21             end
22         else
23             if $v_{1,k_1}^u = v_{1,k_2}^l$ then
24                 $v_{1,k_1}^u = v_{1,k_2}^u$;
25             else
26                 insert $e_1^p$ in $ME^P$, $e_1^p = e_2^p$;
27             end
28         end
29         $g = g + 1$;
30     end
31     insert $e_1^p$ in $ME^P$, $E = \emptyset$, $g = 2$;
32 end
33 return entry set $ME^P$;
```

Appendix B
Updating PRMT and PMT with EM Algorithm

The analysis in Section V indicates that the problem of updating PRMT and PMT is essentially to solve the maximum likelihood estimation (MLE) in Eq. (6) to get $\hat{\mathbf{A}}^t$. This can be done by leveraging the EM algorithm [47] that can find the M-LE of a probabilistic model with hidden variables. Specifically, EM algorithm obtains the maximum of an objective function by increasing its lower bound with an iterative approach that consists of two major steps, *i.e.*, the expectation and maximization steps. The expectation step first estimates the distribution of hidden variables based on the sampled telemetry data X, and then calculates the expectation of the aforementioned lower bound accordingly. Next, the maximization step increases the lower bound and recalculates the corresponding parameters. At the beginning, $\mathbf{A}^t$ is initialized as

$$\alpha_{i,n}^t(0) = \theta_{i,n}^t, \tag{9}$$

Algorithm 6: Procedure of updating PRMT and PMT

```
Input: Current PRMT, matrix set $\mathbf{V}^m$, INT packets in an
       update cycle $N_3$, maximum iterations $N_4$.
1  $t = 1$, $j = 0$, $\mathbf{V}_1^m = \mathbf{V}_2^m = \mathbf{V}_3^m = \mathbf{V}_4^m = \mathbf{0}$;
2  while receive an INT packet of monitored flow $f_m$ do
3      $j = j + 1$;
4      update the $j$-th rows of $\mathbf{V}_1^m$, $\mathbf{V}_2^m$, $\mathbf{V}_3^m$, and $\mathbf{V}_4^m$;
5      if $j = N_3$ then
6          for each type of telemetry data do
7              initialize $\mathbf{A}^t$ with Eq. (9);
8              for each $k \in [1, N_4]$ do
9                  perform expectation step with Eq. (13);
10                 perform maximization step with Eq. (17);
11             end
12             update PRMT with Eq. (18);
13         end
14         update corresponding PMT based on new PRMT;
15         $j = 0$, $t = t + 1$;
16     end
17 end
```

where $\theta_{i,n}^t$ denotes the value of $\theta_{i,n}$ at the beginning of the current (*i.e.*, the t-th) update cycle.

As EM algorithm ensures that the lower bound of likelihood function $L(\mathbf{A}^t)$ will increase after each iteration, it eventually converges to the maximum value [47]. We denote the $\mathbf{A}^t$ in the k-th iteration as $\mathbf{A}^t(k)$ and find the lower bound of $L(\mathbf{A}^t(k))$ according to the Jensen's inequality [51] as

$$\begin{aligned} L\left(\mathbf{A}^t(k)\right) &= \sum_{j=1}^{N_3} \log \left[\sum_{\mathbf{z_j}} P\left(x_j, \mathbf{z_j} \mid \mathbf{A}^t(k)\right)\right] \\ &= \sum_{j=1}^{N_3} \log \left\{\sum_{\mathbf{z_j}} \left[Q\left(\mathbf{z_j}\right) \cdot \frac{P\left(x_j, \mathbf{z_j} \mid \mathbf{A}^t(k)\right)}{Q\left(\mathbf{z_j}\right)}\right]\right\} \\ &\geqslant \sum_{j=1}^{N_3} \sum_{\mathbf{z_j}} \left\{Q\left(\mathbf{z_j}\right) \cdot \log \left[\frac{P\left(x_j, \mathbf{z_j} \mid \mathbf{A}^t(k)\right)}{Q\left(\mathbf{z_j}\right)}\right]\right\}, \end{aligned} \tag{10}$$

where $Q\left(\mathbf{z_j}\right)$ is the posterior distribution of $\mathbf{z_j}$ that satisfies

$$Q\left(\mathbf{z_j}\right) = P\left(\mathbf{z_j} \mid x_j, \mathbf{A}^t(k-1)\right). \tag{11}$$

Since $Q\left(\mathbf{z_j}\right)$ is independent of $\mathbf{A}^t(k)$, we can ignore the parts in the right side of Eq. (10) that are unrelated to $\mathbf{A}^t(k)$ and get the lower bound $B(\mathbf{A}^t(k))$ as

$$B(\mathbf{A}^t(k)) = \sum_{j=1}^{N_3} \sum_{\mathbf{z_j}} \left\{Q\left(\mathbf{z_j}\right) \cdot \log \left[P\left(x_j, \mathbf{z_j} \mid \mathbf{A}^t(k)\right)\right]\right\}. \tag{12}$$

Then, in each iteration, EM algorithm works as follows. In the **expectation step**, the DA calculates the distribution of hidden variables. We denote the probability of the i-th type of telemetry data that matches to value/range $r_{i,n}$ based on the data in the j-th INT packet as $\mu_{j,n}^i$ (*i.e.*, $\mu_{j,n}^i = Q(z_{i,j} = n)$). Hence, the probability distribution of $Q(z_{i,j})$ can be denoted as $\{\mu_{j,1}^i, \cdots, \mu_{j,M_i}^i\}$. In the k-th iteration, $\mu_{j,n}^i$ is obtained as

$$\mu_{j,n}^i(k) = \frac{\mathbf{V}_{i,t}^m(j,n) \cdot \alpha_{i,n}^t(k-1)}{\sum\limits_{n'=1}^{M_i} \left[\mathbf{V}_{i,t}^m(j,n') \cdot \alpha_{i,n'}^t(k-1)\right]}, \tag{13}$$

where $\mathbf{V}_{i,t}^m$ represents $\mathbf{V}_i^m$ at the t-th update cycle. In the **maximization step**, we find the $\mathbf{A}^t(k)$ that maximizes $B(\mathbf{A}^t(k))$. The principle of P4InfoSen-INT suggests that the scheme of telemetry data collection is fixed. Then, for any hidden variable distribution $\mathbf{z_j}$, we can get its unique data collection scheme and know whether there is a conflict between $\mathbf{z_j}$ and x_j, *i.e.*,

$$P\left(x_j, \mathbf{z_j} \mid \mathbf{A}^t(k)\right) = \begin{cases} 0, & Q(\mathbf{z_j}) = 0, \\ P\left(\mathbf{z_j} \mid \mathbf{A}^t(k)\right), & \text{otherwise.} \end{cases} \tag{14}$$

Therefore, Eq. (12) can be converted into

$$\begin{aligned} B(\mathbf{A}^t(k)) &= \sum_{j=1}^{N_3} \sum_{\mathbf{z_j}} \left\{Q\left(\mathbf{z_j}\right) \cdot \log \left[P\left(\mathbf{z_j} \mid \mathbf{A}^t(k)\right)\right]\right\} \\ &= \sum_{j=1}^{N_3} \left\{\sum_{n_1=1}^{M_1} \cdots \sum_{n_4=1}^{M_4} \left[\prod_{i=1}^{4} \mu_{j,n_i}^i(k) \cdot \log \left(\prod_{i=1}^{4} \alpha_{i,n_i}^t(k)\right)\right]\right\}, \end{aligned} \tag{15}$$

where we have

$$\begin{cases} \sum\limits_{j=1}^{N_3} \left\{\sum\limits_{n_1=1}^{M_1} \cdots \sum\limits_{n_4=1}^{M_4} \left[\prod\limits_{i=1}^{4} \mu_{j,n_i}^i(k)\right]\right\} = N_3, \\ \sum\limits_{n_1=1}^{M_1} \cdots \sum\limits_{n_4=1}^{M_4} \left[\prod\limits_{i=1}^{4} \alpha_{i,n_i}^t(k)\right] = 1. \end{cases} \tag{16}$$

Then, it can be easily verified that when $B(\mathbf{A}^t(k))$ reaches its maximum, the value of $\alpha_{i,n}^t(k)$ is

$$\alpha_{i,n}^t(k) = \frac{\sum\limits_{j=1}^{N_3} \mu_{j,n}^i(k)}{N_3}, \tag{17}$$

which is the update equation of the maximization step.

The two steps are repeated for N_4 iterations. Then, we get the probability distribution of each type of telemetry data in the current (the t-th) update cycle, and update the PRMT as

$$\theta_{i,n}^{t+1} = \frac{t-1}{t} \cdot \theta_{i,n}^t + \frac{1}{t} \cdot \alpha_{i,n}^t(k), \tag{18}$$

where $\theta_{i,n}^t$ is the distribution estimated in previous $t-1$ cycles.

The procedure of updating PMT and PRMT is shown in *Algorithm* 6. We update PRMT and the corresponding PMT when the DA have received and processed all the INT packets in an update cycle (*Lines* 2-17). *Line* 4 updates the j-th rows of $\mathbf{V}^m$ with the procedure in Fig. 11. If an INT packet is the last one in the current cycle, *Lines* 6-13 use EM algorithm to update the PRMT, and *Line* 14 updates the priorities in the corresponding PMT based on the new PRMT. With *Algorithm* 6, the information content updater in each DA updates PRMTs to the controller, enabling the P4InfoSen-INT system to adapt to dynamic network changes and optimize the tradeoff between bandwidth overheads and monitoring accuracy accordingly.

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] W. Lu *et al.*, "AI-assisted knowledge-defined network orchestration for energy-efficient data center networks," *IEEE Commun. Mag.*, vol. 58, pp. 86–92, Jan. 2020.

[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] A. Matencio-Escolar, Q. Wang, and J. Alcaraz 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.

[6] 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.

[7] Z. Zhu *et al.*, "Demonstration of cooperative resource allocation in an OpenFlow-controlled multidomain and multinational SD-EON testbed," *J. Lightw. Technol.*, vol. 33, pp. 1508–1514, Apr. 2015.

[8] S. Li *et al.*, "Improving SDN scalability with protocol-oblivious source routing: A system-level study," *IEEE Trans. Netw. Serv. Manag.*, vol. 15, pp. 275–288, Mar. 2018.

[9] L. Gong and Z. Zhu, "Virtual optical network embedding (VONE) over elastic optical networks," *J. Lightw. Technol.*, vol. 32, pp. 450–460, Feb. 2014.

[10] H. Jiang, Y. Wang, L. Gong, and Z. Zhu, "Availability-aware survivable virtual network embedding (A-SVNE) in optical datacenter networks," *J. Opt. Commun. Netw.*, vol. 7, pp. 1160–1171, Dec. 2015.

[11] L. Gong, H. Jiang, Y. Wang, and Z. Zhu, "Novel location-constrained virtual network embedding (LC-VNE) algorithms towards integrated node and link mapping," *IEEE/ACM Trans. Netw.*, vol. 24, pp. 3648–3661, Dec. 2016.

[12] M. Zeng, W. Fang, and Z. Zhu, "Orchestrating tree-type VNF forwarding graphs in inter-DC elastic optical networks," *J. Lightw. Technol.*, vol. 34, pp. 3330–3341, Jul. 2016.

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

[14] Z. Xu and Z. Zhu, "On establishing and task scheduling of data-oriented vNF-SCs in an optical DCI," *J. Opt. Commun. Netw.*, vol. 14, pp. 89–99, Mar. 2022.

[15] R. Govindan *et al.*, "Evolve or die: High-availability design principles drawn from Google's network infrastructure," in *Proc. of ACM SIGCOMM 2016*, pp. 58–72, Aug. 2016.

[16] Z. Pan *et al.*, "Advanced optical-label routing system supporting multicast, optical TTL, and multimedia applications," *J. Lightw. Technol.*, vol. 23, pp. 3270–3281, Oct. 2005.

[17] 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.

[18] 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.

[19] 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.

[20] 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.

[21] B. Claise, "Cisco systems NetFlow services export version 9," *RFC 3954*, Oct. 2004. [Online]. Available: https://tools.ietf.org/html/rfc3954.

[22] P. Bosshart *et al.*, "P4: Programming protocol-independent packet processors," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 44, pp. 87–95, Jul. 2014.

[23] S. Li *et al.*, "Protocol oblivious forwarding (POF): Software-defined networking with enhanced programmability," *IEEE Netw.*, vol. 31, pp. 12–20, Mar./Apr. 2017.

[24] INT dataplane specification. [Online]. Available: https://github.com/p4lang/p4-applications/blob/master/docs/INT_v2_1.pdf.

[25] C. Kim *et al.*, "In-band network telemetry via programmable dataplanes," in *Proc. of ACM SIGCOMM 2015*, pp. 1–2, Aug. 2015.

[26] 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.

[27] 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.

[28] Y. Kim, D. Suh, and S. Pack, "Selective in-band network telemetry for overhead reduction," in *Proc. of CloudNet 2018*, pp. 1–3, Oct. 2018.

[29] 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.

[30] B. Basat *et al.*, "PINT: Probabilistic in-band network telemetry," in *Proc. of ACM SIGCOMM 2020*, pp. 662–680, Aug. 2020.

[31] 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.

[32] Intel Tofino 2. [Online]. Available: https://www.intel.com/content/www/us/en/products/network-io/programmable-ethernet-switch/tofino-2-series.html.

[33] T. Pan *et al.*, "Sailfish: Accelerating cloud-scale multi-tenant multi-service gateways with programmable switches," in *Proc. of ACM SIGCOMM 2021*, pp. 194–206, Aug. 2021.

[34] C. Zhang *et al.*, "HyperV: A high performance hypervisor for virtualization of the programmable data plane," in *Proc. of ICCCN 2017*, pp. 1–9, Sept. 2017.

[35] P. Zheng, T. Benson, and C. Hu, "P4Visor: Lightweight virtualization and composition primitives for building and testing modular programs," in *Proc. of ACM CoNEXT 2018*, pp. 98–111, Dec. 2018.

[36] A. Sgambelluri *et al.*, "Exploiting telemetry in multi-layer networks," in *Proc. of ICTON 2020*, pp. 1–4, Jul. 2020.

[37] 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.

[38] 100G in-band network telemetry with Netcope P4. [Online]. Available: https://www.netcope.com/Netcope/media/content/100G-In-band-Network-Telemetry-With-Netcope-P4.pdf.

[39] G. Simsek, D. Ergenc, and E. Onur, "Efficient network monitoring via in-band telemetry," in *Proc. of DRCN 2021*, pp. 1–6, Apr. 2021.

[40] A. Castro *et al.*, "Near-optimal probing planning for in-band network telemetry," *IEEE Commun. Lett.*, vol. 25, pp. 1630–1634, May 2021.

[41] J. Vestin *et al.*, "Programmable event detection for in-band network telemetry," in *Proc. of CloudNet 2019*, pp. 1–6, Nov. 2019.

[42] E. Song *et al.*, "INT-filter: Mitigating data collection overhead for high-resolution in-band network telemetry," in *Proc. of GLOBECOM 2020*, pp. 1–6, Dec. 2020.

[43] S. Bhattacharyya, C. Diot, and J. Jetcheva, "Pop-level and access-link-level traffic dynamics in a Tier-1 POP," in *Proc. of ACM SIGCOMM IMW 2001*, pp. 39–53, Nov. 2001.

[44] S. Tang *et al.*, "Self-adaptive network monitoring in IP-over-EONs: When multilayer telemetry is flexible and driven by data analytics," in *Proc. of OFC 2021*, pp. 1–3, Jun. 2021.

[45] S. Tang, S. Zhao, X. Pan, and Z. Zhu, "How to use in-band network telemetry wisely: Network-wise orchestration of Sel-INT," *IEEE/ACM Trans. Netw.*, vol. 31, pp. 421–435, Jul. 2023.

[46] P4InfoSen-INT. [Online]. Available: https://github.com/USTC-INFINITELAB/P4InfoSen-INT.

[47] C. Do and S. Batzoglou, "What is the expectation maximization algorithm," *Nat. Biotechnol.*, vol. 26, pp. 897–899, Aug. 2008.

[48] pktgen-DPDK. [Online]. Available: https://git.dpdk.org/apps/pktgen-dpdk/.

[49] S. Liu and Z. Zhu, "Generating data sets to emulate dynamic traffic in a backbone IP over optical network," *Tech. Rep.*, 2019. [Online]. Available: https://github.com/lsq93325/Traffic-creation/blob/master/README.md?tdsourcetag=s_pctim_aiomsg.

[50] P4Runtime specification. [Online]. Available: https://p4.org/p4-spec/p4runtime/main/P4Runtime-Spec.html.

[51] D. Mitrinovic, E. Barnes, D. Marsh, and J. Radok, *Elementary Inequalities*. Noordhoff Groningen, 1964.