LINT: Accuracy-adaptive and Lightweight In-band Network Telemetry

Shihabur Rahman Chowdhury*, Raouf Boutaba*, and Jérôme François†
*David R. Cheriton School of Computer Science, University of Waterloo
{sr2chowdhury | rboutaba}@uwaterloo.ca
†INRIA - Nancy Grand Est, France
jerome.francois@inria.fr

***Abstract*—In-band Network Telemetry (INT) has recently emerged as a means of achieving per-packet near real-time visibility into the network. INT capable network devices can directly embed device internal state such as packet processing time, queue occupancy and link utilization information in each passing packet. INT is enabling new network monitoring applications and is currently being used in production for providing fine-grained feedback to congestion control mechanisms. The microscopic network visibility facilitated by INT comes at the expense of increased data plane overhead. INT piggybacks telemetry information on user data traffic and can significantly increase packet size. A direct consequence of increasing packet size for carrying telemetry data is a substantial drop in network goodput. This paper aims at striking a balance between reducing INT data plane overhead and the accuracy of network view constructed from telemetry data. To this end, we propose LINT, an accuracy-adaptive and Lightweight INT mechanism that can be implemented on commodity programmable devices. Our evaluation of LINT using real network traces on a fat tree topology demonstrates that LINT can reduce INT data plane overhead by ≈25% while ensuring more than 0.9 recall for monitoring queries trying to identify congested flows and switches in the network.**

***Index Terms*—Network telemetry, Programmable networks**

I. Introduction

Network monitoring is fundamental to network management and is the basis of many network Operations, Administration and Management (OAM) activities such as fault management [1], traffic engineering [2], threat detection and mitigation [3], [4], and capacity planning, accounting and billing [5], among others. Traditionally, network monitoring has been *pull-based*, *i.e.*, a centralized control plane or network management system periodically reads the counters from the network devices [2], [6]. A drawback of pull-based monitoring is the coarse time granularity of monitoring the network, typically in the order of seconds or minutes. In response to the limitations of pull-based monitoring, push-based *streaming telemetry* has been recently emerging as a paradigm where network devices directly stream telemetry information to data collection and analytics engines [7]–[9], providing near real-time and microscopic visibility into the network.

Along the same vein of streaming telemetry, *In-band Network Telemetry (INT)* [10] has recently emerged as a means to obtain per-packet real time view of the network. INT is an outstanding effort to enable network devices (*e.g.*, software and hardware switches, Network Interface Cards (NICs)) to embed device internal state such as packet processing latency, queue depth and link utilization into each passing packet, consequently, facilitating a real-time and microscopic view into network traffic. Such fine-grained telemetry capability is enabling new use-cases such as pin-pointing the root cause of congestion and packet drops through switch queue profiling [11] and per-packet fine-grained feedback to support low-latency data center transport [12], which are otherwise difficult to perform with traditional network monitoring. As of today, INT is supported by commodity hardware such as fixed function and programmable switches [11], [13], and SmartNICs [14], [15], and is being deployed in production telecommunications and data center networks [12], [16].

The microscopic telemetry capabilities enabled by INT come at the expense of increased data plane overhead [17], [18]. This overhead is attributed to each INT capable network device on a packet's path augmenting the packet with telemetry data, thus increasing the packet's size in proportion to the path length. For instance, collecting three telemetry data items on a 5-hop path in a data center results in more than 40% additional bits (compared to the original packet size) added to a packet (details in Section II). Packet size increase for carrying telemetry data can have several negative consequences. First, it reduces data plane goodput since a lesser fraction of the bits in a packet now become usable for transporting user data traffic [18]. Furthermore, packet size increase beyond the Maximum Transport Unit (MTU) fragments the packet, adding the overhead of packet reassembly and potentially increasing latency. In this context, we set out to answer the following question: *can we reduce INT data plane overhead without severely degrading the quality of network monitoring queries that rely on telemetry data collected through INT?*

In this paper, we aim at reducing the data plane overhead of INT while reaping its benefits as much as possible. Our objective is to identify and filter the *less interesting* observations of telemetry data directly in the data plane without negatively impacting the quality of collected telemetry data. In this context, we use quality of results produced by different network monitoring queries as an indicator of the quality of the telemetry data. To this end, we propose LINT, an accuracy-adaptive and **L**ightweight **INT** mechanism that runs in the data plane. Specifically, we make the following contributions:

- We analyze several publicly available real network traces

978-3-903176-32-4

to quantify INT data plane overhead. Our study complements the one presented in [18] that uses synthetic traffic.
- We present LINT, an accuracy-adaptive and lightweight INT mechanism for programmable data plane. Network devices employing LINT independently decide on selectively reporting telemetry data on passing packets, without any explicit coordination and intervention from a control plane.
- We evaluate LINT using a combination of network emulation and simulation, and publicly available real network traces. Our simulation results show that LINT can reduce INT data plane overhead by about 25% while achieving more than 90% recall (compared to regular INT) for monitoring queries trying to identify flows with high latency and congested switches in the network.

The rest of the paper is organized as follows. We first briefly describe how INT operates in Section II. In the same section, we also present an empirical study demonstrating the extent of incurred INT data plane overhead using real network traces. Then, we present our solution in Section III followed by the evaluation of LINT using real network traces in Section IV. We discuss the related works and contrast our work with state-of-the-art in Section V. Finally, we conclude with some future research directions in Section VI.

II. Background and Motivation

A. In-band Network Telemetry (INT)

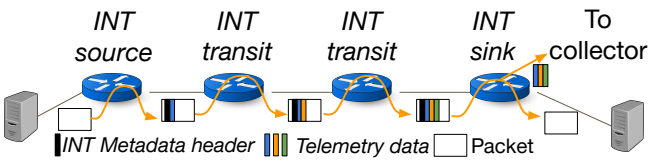


Fig. 1. INT operation

INT is a standardization effort from the P4 applications working group for gaining near real-time microscopic visibility into the network [10]. As mentioned earlier, INT enables network devices to embed telemetry information directly into the passing packets. The INT specification makes the following conceptual classification of network devices:

INT source: The first network device on a packet's path that initiates INT and embeds telemetry data in the packet. This INT source can be a software switch, a SmartNIC, or an INT capable top-of-rack switch or a border router.

INT transit: The network devices on a packet's path that embed telemetry data in the packets.

INT sink: The last INT capable device on a packet's path. An INT sink strips off all the telemetry information from a packet, constructs an *INT report* according to the INT specification [10] and sends the INT report to a collector. The INT sink can be configured to send all or selectively some of the reports to the collector based on pre-defined policies.

The operation of INT is summarized in Fig. 1. An INT source can be configured to initiate telemetry data collection for each packet or for packets matching a watchlist. The

TABLE I
Example of telemetry data

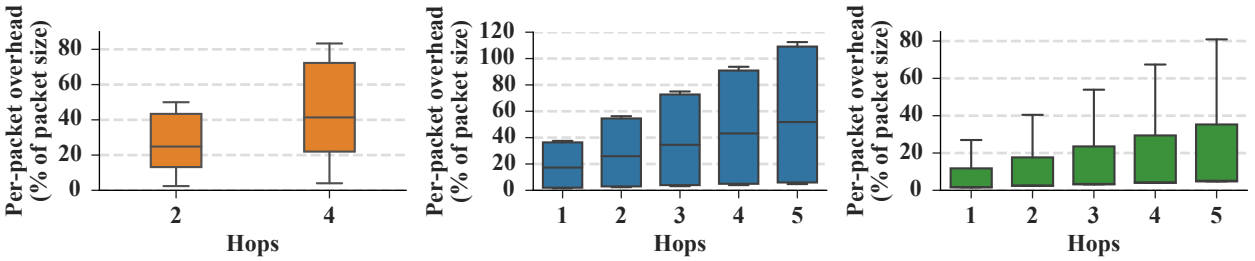
Telemetry data	Description
Switch ID	Identifier associated with a device
Ingress Port ID	Identifier of the packet's arriving port
Egress Port ID	Identifier of the packet's outgoing port
Ingress/Egress Timestamp	The packet's time of arrival/departure
Hop Latency	Time spent by the packet in the device
Egress port TX utilization	Utilization of the packet's output port
Queue occupancy	The packet's observed outgoing queue size

INT source initiates the telemetry data collection process by encapsulating the packet using one of the protocols described in the INT specification and inserting an *INT metadata header* (12 bytes) into a packet. An INT header contains control information such as the maximum number of INT capable devices on the packet's path, the encapsulation protocol to use for INT and the set of telemetry data that each INT transit device should add to the packet. Then, each INT transit device adds telemetry data to the packets carrying INT metadata header. Finally, the INT sink strips the telemetry information from the packets, removes the encapsulation and INT metadata headers, and restores the original packet before sending it to its destination. The sink can be configured to send all or a subset of reports to a collector based on pre-defined policies (*e.g.*, report only when total path latency exceeds a threshold).

The current INT specification defines a set of telemetry data items (4 bytes each) as presented in Table I [10]. However, programmable switches and SmartNICs enabled by the Protocol Independent Switch Architecture (PISA) [22] can be programmed using the P4 programming language [23] for computing other functions on the packets and the flows (*e.g.*, mean packet size, moving average of queue occupancy), and augment the packets with the result. Telemetry data collected through INT can be used for answering network monitoring queries that require per-packet information (*e.g.*, identifying flows that have a congested switch on its path, computing per-packet end-to-end latency distribution, per-switch queue profiling for identifying root cause of congestion or increased tail latency, among others) or provide feedback to control and management applications (*e.g.*, congestion control [12]).

B. INT Overhead

There are several sources of overhead in INT. First, the INT source adds a 12 byte INT metadata header to each packet [10]. Then, each INT transit node adds one or more telemetry data items to the packet according to the instruction embedded in the INT metadata header. The INT specification reserves 4 bytes for each telemetry data item to be added [10]. Therefore, a switch reporting 3 data items such as its SwitchID, the hop latency and the queue occupancy will add 12 bytes to the passing packets. Since each transit node on a packet's path adds telemetry data to the packet, INT overhead increases linearly with the path length of the packet. For instance, if a packet goes through 5 INT switches (including the INT source) and each switch is adding 3 telemetry data items, then the packet will have 72 bytes added to it when it reaches the INT sink. For MTU size packets (1500 bytes



(a) FB-WS [19] (b) UNIV1 [20] (c) CAIDA (Equinix 2016) [21]

Fig. 2. Packet size increase due to INT

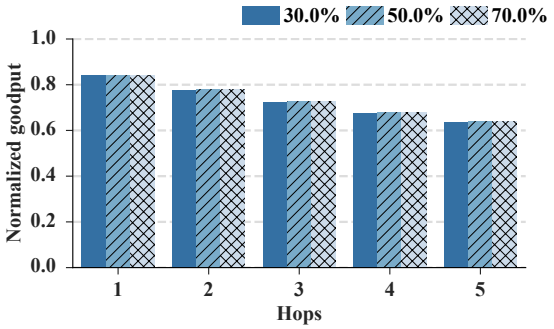


Fig. 3. Mean normalized goodput of a link by varying the INT hops (*i.e.*, the per-packet overhead) and link utilization (considering packet sizes from the UNIV1 trace [20])

for Ethernet), the bit overhead can translate to ≈5% increase in packet size. However, the mean and median packet sizes in most networks are typically much smaller than the MTU (*e.g.*, median packet size is close to 250 bytes in data centers [19], [20]), resulting in higher per-packet bit overhead due to INT.

More recently, an empirical study in [18] sheds light on the extent of network goodput degradation due to INT's bit overhead in the data plane. This study used ns3 simulation on a fat-tree data center network [24] with synthetic traffic generated from a web search workload. The results show that INT's bit overhead in the data plane can reduce network goodput by as much as ≈ 20%. In this paper, we present a complementary study to that presented in [18], demonstrating the extent of per-packet bit overhead on some publicly available real network traces. Specifically, we analyze the traces from Facebook's production web service cluster (FB-WS) [19], a campus data center network (UNIV1 data set from [20]) and a wide-area network traffic capture from CAIDA (Equinix 2016) [21]. Unlike the UNIV1 and CAIDA traces, the FB-WS trace does not contain packet captures. Rather, it contains meta-data extracted from sampled packets (*e.g.*, packet size, source and destination racks and pods) from the end-hosts.

The result of our analysis is presented in Fig. 2. For this analysis, we assumed collecting three telemetry data items from each switch on a packet's path. We varied the number of INT devices on a packet's path (*i.e.*, Hops in the figure) from 1 to 5. To provide some context, in a fat-tree data center network [24], a path between servers within the same pod and different pods is 2 and 4 hops, respectively. For wide area networks such as the autonomous system level graph, the median path length lies between 5 and 6 [25]. Note that for the FB-WS trace, we leveraged the meta-data describing a packet's source and destination pod and rack, and correlated that with Facebook's data center network architecture [26] to identify the number of hops. All the clusters in Facebook's production network exhibited similar behavior, except for the Hadoop cluster, where almost all the packets were MTU sized. For the other traces, no such meta-data was made available, hence, we experimented with a range of hop counts. In Fig. 2, we plot the distribution of per-packet overhead (the box represents the quartiles and the bars' end points represent the extremes) in terms of percent increases from original packet size.

For the FB-WS, UNIV1, and CAIDA traces, the median packet size increase is ≈40%, ≈40%, and ≈10% compared to the original packet size, respectively, for collecting three telemetry data items on a 4-hop path. On the higher end of the distribution, we observe the 75-th percentile overhead under the same setting to be ≈70%, ≈90%, and ≈30%, respectively, which is significant. A direct consequence of packet size increase for transporting telemetry data is reduced network goodput since lesser fraction of the bits in a packet remain available for transporting the original network traffic.

We also conduct another analytical study to measure the impact of INT data plane overhead on network goodput. For this study, we consider a 10 Gbps network link carrying traffic similar to that of the UNIV1 trace [19]. We assume the packet sizes to be uniform and equal to the median packet size in UNIV1 trace. With this assumption, we compute that link's goodput considering different levels of utilization and normalize the result with that from the case when INT is not used. We can see from the results in Fig. 3 that even for one hop, INT data plane overhead can reduce network goodput by ≈20%. Moreover, on a typical pod-to-pod path in a fat-tree topology (*i.e.*, 4 hops), the goodput can reduce by ≈30% due to packet size increase. These analytical results motivate the need for mechanisms that can strike a balance between INT data plane overhead and the quality of collected telemetry data for answering network monitoring queries.

III. The LINT Algorithm

We can draw an analogy between INT and sensor networks. INT capable devices are similar to sensors that measure and report certain metrics from the environment to a collector or sink. Sensors are typically resource constrained (*e.g.*, limited battery life and limited network bandwidth), therefore, need to carefully measure and report without abusing the constrained resources. Albeit not constraint, however, INT needs to work

in a way to not incur significant data plane overhead, thereby, negatively impacting regular network operations. Given the similarity, we leverage techniques from *model-driven data acquisition*, a well-studied topic in the sensor networking literature for reducing data transmission from sensor nodes to the sinks [27]. We propose LINT, an accuracy-adaptive and lightweight INT mechanism that can run in programmable data plane. LINT selectively reports telemetry data items on passing packets by estimating accuracy loss at the collector. LINT can be implemented within the constraints of commodity programmable PISA devices, and can work without any global coordination and intervention from a control plane.

In the following, we first give a brief overview of model-driven data acquisition that forms the basis of our solution (Section III-A) followed by the description of LINT (Section III-B). We also present LINT-flow (Section III-C), an extension of LINT that takes the flow-context of the packets into consideration. Finally, we conclude this section with some implementation considerations for LINT (Section III-D).

A. Overview of Model-driven Data Acquisition

In the model-driven data acquisition paradigm, a prediction model is used to determine if sensors should be queried for new data or to filter the data at the sensor. The prediction model is devised for capturing the pattern of the measurements or to correlate measurements of different metrics. At one extreme of the approach is the one presented in [27], where the model is solely used for determining if a query engine should query the sensor nodes for measurements or not. The query engine queries a sensor only when it determines that the model output is not sufficient for maintaining a satisfactory level of accuracy. In contrast, the Spanish Inquisition Protocol (SIP) [28] is on the other extreme where a predictor is used to solely determine if a sensor reading should be transmitted to a sink or can be dropped at the sensor. The sensors using SIP use a predictor function to forecast what the sink is expecting next to receive. When the estimation indicates that the current sensor reading is far off from what the sink is expecting, only then the sensors send their readings to the sink. Our problem is close to the latter, *i.e.*, determining from the data plane if telemetry data items should be reported. Hence, we will use SIP as a basis for our solution. A comprehensive survey of model-driven data acquisition techniques can be found in [29].

B. LINT: Accuracy-adaptive and Lightweight INT

1) Overview: While designing LINT, our goal is to keep it lightweight. In other words, LINT should be capable of running within the constraint of commodity PISA devices such as no floating point operations, limited to no multiplication and no division, no loops or recursion, limited number of match-action stages and limited match-action entries per-stage, and only one stateful register memory read-modify-write per packet processing stage [22], [30]. Furthermore, executing LINT should not consume substantial amount of device resources and should leave enough resources for running other applications (*e.g.*, [31]–[34]) on a PISA device.

Algorithm 1: LINT Algorithm

Input: p = The current packet; $\mathcal{D}$ = metrics to monitor; α = weight parameter of EWMA; δ = error threshold

1 **function** LINT($p, \mathcal{D}, \alpha, \delta$)
2 **foreach** $d \in \mathcal{D}$ **do**
3 $val_d \leftarrow$ current observation of d
4 $s \leftarrow d_D^{next}$, $t \leftarrow d_C^{next}$
5 $d_D^{next} \leftarrow \alpha val_d + (1-\alpha)s$
6 $deviation \leftarrow |d_D^{next} - t|$
7 **if** $deviation > \delta d_D^{next}$ **then**
8 $p.add\_telemetry\_observation(d, val_d)$
9 $d_C^{next} \leftarrow \alpha val_d + (1-\alpha)t$

Therefore, to keep LINT simple, we build on SIP presented in [28]. For each packet with an INT metadata header that arrives at a PISA device, LINT makes the decision of reporting a telemetry data item according to Algorithm 1 as follows. A device running LINT tries to estimate the amount of error that can be introduced at the collector if the requested telemetry data items are not piggybacked on the current packet. For estimating this error, the device uses a predictor function for each telemetry data item of interest. The predictor function for a telemetry data item d is used for computing the following:

- d_D^{next}: the predictor function applied on all past observations of d in this device.
- d_C^{next}: the predictor function applied on the observations of d reported to the collector.

Essentially, the quantity d_C^{next} denotes what the collector will predict about the observation of d if the current observation is not reported. The device decides to report the currently observed value of d if the difference between d_D^{next} and d_C^{next} is within an operator defined fraction δ of d_D^{next}, *i.e.*, $|d_D^{next} - d_C^{next}| \leq \delta \times d_D^{next}$ (line 4 – 7). In other words, when the device estimates that the prediction error at the collector can go above an acceptable threshold, it reports the current observation. Otherwise, the device skips reporting the current observation of d. In this way, LINT adapts telemetry data reporting to estimated error. Note that we can choose the parameter δ to be in the form 2^{-m}, in this way replace the multiplication operation by a bit shift operation.

2) Device and collector coordination: The value of d_D^{next} is updated whenever a packet arrives with INT metadata header instruction for reporting d. However, d_C^{next} is updated only when the device reports an observation of d piggybacking on a packet. The collector replaces any missing telemetry data item d not reported by a device on the packet's path by using the same predictor function as the device. In this way, both the device and the collector stay in sync about the extent of the error due to not reporting an observation of a telemetry data item d. Also, each device independently makes their own decision. Indeed, additional information about the error estimate can improve the quality of decision making for a device. However, that would require coordination between the devices and is not a desirable for keeping LINT lightweight.

3) Choice of predictor function: The concrete realization of LINT requires deciding on a predictor function that can be computed within the constraints of PISA devices. In this

regard, we chose from the moving average family of predictor functions since they have a constant memory footprint, have less number of parameters to tune and are computationally lightweight. We leave the exploration of more computationally demanding predictor such as machine learning based prediction [35] for a future exploration. Specifically, we chose to use Exponentially Weighted Moving Average (EWMA) [36] for LINT. EWMA computes moving average of a data stream by applying exponentially decaying weights to the items in the stream according to the order they appear. As time progresses, observations further in the past have lesser and lesser impact on EWMA. We can recursively compute EWMA for a stream of observations $\tilde{x} = \langle x_0, x_1, \ldots x_t \rangle$ as follows:

$$S_0 = x_0$$
$$S_t = \alpha x_t + (1 - \alpha) S_{t-1} \quad (0 < \alpha < 1)$$

Here, x_t is the observation at the current time t, S_t is the EWMA at time t computed from x_t and S_{t-1}. The weight α determines how much importance will be given to the past observations. The multiplication term involving the fraction α can be avoided by choosing α of the form 2^{-m} (for some integer $m > 0$) [30]. By doing so, we can rewrite the EWMA computation equation as follows:

$$S_t = S_{t-1} + 2^{-m}(x_t - S_{t-1}) \quad (m > 0) \tag{1}$$

The multiplication by 2^{-m} $(m > 0)$ in (1) can be performed by shifting bits to the right m times, which is supported by commodity programmable hardware.

C. LINT-Flow: Flow-context aware LINT

Very often packets from the same network flow exhibit similar behavior (*e.g.*, often due to packets of the same flow belonging to the same application) and are subjected to same operational policies (*e.g.*, packets from the same flow sent to the same output queue based on flow priority). Therefore, applying the predictor function with a packet's flow context in consideration has the potential to reduce errors. In this regard, we propose LINT-flow, an extension of LINT that also takes a packet's flow context into consideration while applying the predictor function.

In contrast to maintaining a pair of EWMA values for each telemetry data item (*i.e.*, EWMA of all observations in a device and EWMA of the observations reported to the collector from the device), we maintain a pair of EWMA values for each observed flow in a hash table. Without loss of generality we assume the network flows are identified by the five tuple (source IP, destination IP, network protocol, source port, destination port). When a packet with INT metadata header arrives at a device, LINT-flow identifies the hash table entries corresponding to the flow that the packet belongs to and updates the EWMA values accordingly. Subsequently, LINT-flow considers the difference between the EWMA values corresponding to the packet's flow while deciding which telemetry data item(s) should be reported on a packet.

D. Implementation Considerations

Realizing LINT on programmable data plane will require changes to the INT protocol message formats. One key issue that must be addressed is how to communicate to the collector that only a subset of the originally requested telemetry data items have been reported. One solution is to embed a bitmap at each INT transit node, representing the telemetry data items that the node is reporting. To avoid consuming more bits, this bitmap can share unused space in other fields such as the SwitchID. Devising a robust solution for this issue requires further investigation and we leave it for future exploration.

Our ongoing implementation effort is mostly simulation-centric and in part is around bmv2, the P4 reference software switch. A full-fledged implementation on a programmable PISA hardware is yet to be done. In this section, we briefly describe the potential data plane resource requirements for LINT. For LINT, we need two stateful registers per telemetry data item for maintaining the EWMA of the observations at the device and the EWMA of the observations sent to the collector. Therefore, a total of $2|\mathcal{D}|$ processing stages and $2|\mathcal{D}|$ register entries will be needed for dealing with a set telemetry data items $\mathcal{D}$. For instance, to selectively report hop latency and queue occupancy (*i.e.*, $\mathcal{D} = \{\text{hop latency, queue occupancy}\}$), we will require 4 processing stages and 8 register entries in total. The parameters α and δ are not expected to change very frequently and we can specify them as constants during pipeline configuration.

For LINT-flow, we will need $4|\mathcal{D}|$ processing stages considering a flow cache implementation similar to that in [34]. However, at each processing stage we will require a hash table that can be implemented using a register array for keeping track of the active set of flows and their corresponding EWMA values. Indeed, keeping track of all flows per processing stage will be impractical. However, one observation is that most network flows are short-lived, especially in data centers [19], [20]. Therefore, the active set of flows will be changing fast, which creates the opportunity for applying cache eviction policies to track only a subset of flows at a time. We present a simulation to study demonstrating the impact of tracking a limited number of flows per processing stage in Section IV-E.

IV. Evaluation

We employ a combination of network emulation and simulation to evaluate the effectiveness of LINT and LINT-flow. Before describing the evaluation results we first briefly describe the methodology. The goal of our evaluation is to contrast between different aspects of LINT with that of performing INT for each packet in the network. To accomplish this, we first deploy a network consisting of bmv2 switches (P4 software switch) using Mininet. The bmv2 switches run a P4 program that implements INT (a modified version of the *int.p4* implementation provided with the ONOS SDN controller). After subjecting the deployed network with traffic through different hosts, we collect the generated INT reports by passively capturing packets on relevant INT sink switch interfaces. These INT reports provide us with the ground truth

to compare against. Then, we simulate LINT and LINT-flow on the captured INT reports to obtain modified INT reports that LINT and LINT-flow would generate when deployed in the network. These generated INT reports are then used for executing several network monitoring queries and the results are compared against the query results obtained using the ground truth. Since network emulation does not provide predictable and reproducible timing behavior, it is difficult to reproduce the same per-packet latency and queue occupancy in the switches in successive runs and compare between approaches, hence, our hybrid approach.

In the following, we first describe the setup (Section IV-A) and the evaluation metrics (Section IV-B). Then we present our evaluation results focusing on the following scenarios: (i) evaluation of INT data plane overhead reduction by using LINT (Section IV-C); (ii) evaluation of the impact of selectively reporting telemetry data items by LINT on the result of network monitoring queries (Section IV-D); and (iii) evaluation of LINT-flow, including studying the impact of limiting the memory for tracking flows (Section IV-E).

A. Setup

1) Topology and Workload: We used a 4-port fat-tree data center network topology (20 switches, 32 links) for our evaluation. Each top-of-the-rack switch in each pod was connected with a traffic generating host. We enabled jumbo Ethernet frames on all the interfaces to avoid packet fragmentation during our experiments. For the workload, we used the packet capture from UNIV1 trace [20]. We divided and distributed the capture files to the Mininet hosts, and replayed them using the *tcpreplay* tool. We used ONOS controller for path setup and for configuring the bmv2 switches to embed SwitchID, hop latency and queue occupancy on all packets. Out of the 4 pods in the topology, the hosts from pod 0 and pod 3 sent traffic to the hosts in pod 1 and pod 2, creating an aggregation traffic pattern (similar to partition–aggregate or reduce workload). This pod-to-pod path consists of 4 INT hops.

2) Network Monitoring Queries: We used the INT reports for answering the following questions about the network:

- **(Q_{Tail}) Tail latency [11]**: Which flows have at least one packet with total hop latency in the tail latency zone? For our experiment, we use the 95-th percentile of total hop latency from all collected INT reports as the threshold for tail latency zone.
- **($Q_{Congestion}$) Congested switch identification**: Which flows have a congested switch on their path? We define a congested switch to be a switch where a packet is experiencing more than $x\%$ of its path's total hop latency. We set this threshold to 40% in our experiments.
- **($Q_{Latency}$) Path latency**: What is the total hop-latency experienced by each of the packets?
- **(Q_{Queue}) Queue profiling [11]**: Obtain the time series of queue occupancy in a switch within a given time window.

3) Parameter Selection: We experimented with different values of α (in the form 2^{-m}) and consistently obtained the best results for $\alpha = 2^{-1}$, hence, used this value for reporting all the results. We varied δ between 2^{-6} and 2^{-1} in multiples of 2 in the experiments.

B. Evaluation Metrics

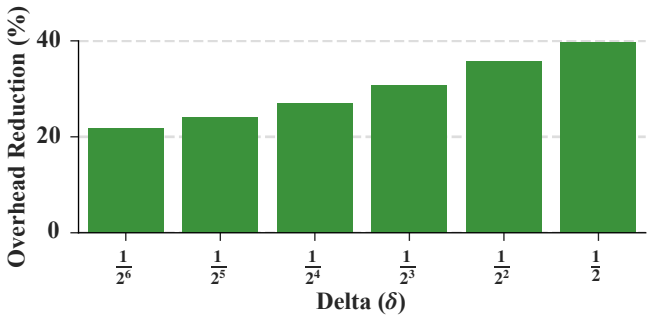
1) Recall: We evaluate queries Q_{Tail} and $Q_{Congestion}$ using recall, *i.e.*, the fraction of identified flows that are also identified by the ground truth. For these queries, we are interested in measuring the fraction of the culprit flows that can still be identified by LINT, hence, the choice of using recall.

2) Normalized Root Mean Squared Error (NRMSE): We evaluate $Q_{Latency}$ and Q_{Queue} by measuring the deviation from the ground truth using NRMSE. For the metrics of interest in these queries, we first compute the root mean squared error (RMSE) across all the collected INT reports. Then we normalize the RMSE by the range of values of that metric and express as percentage.

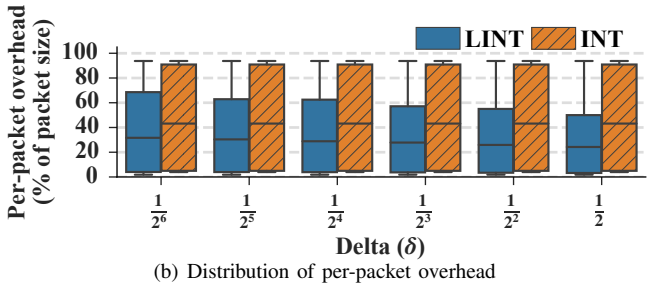
3) Overhead reduction: We express overhead reduction as the ratio of the number of telemetry data item observations that were not reported by LINT (and LINT-flow) to the total number of observations collected in the ground truth.

4) Per-packet overhead: Per-packet overhead is computed as the percent increase in packet size due to INT.

C. Data plane Overhead Reduction



(a) Overhead reduction compared to per-packet INT



(b) Distribution of per-packet overhead

Fig. 4. Overhead comparison between LINT and INT

Our first set of results demonstrate the effectiveness of LINT in reducing data plane overhead compared to regular INT. In Fig. 4(a), we present the percentage overhead reduction by LINT compared to regular INT for different values of δ. The parameter δ provides a tuning knob in our algorithm to increase or decrease overhead while having an opposite effect on how often telemetry data items are reported from the data plane. As we can see, even with a very small δ ($= 2^{-6}$), LINT can reduce 20% data plane overhead compared to regular INT. We can clearly see that a higher δ also increases the gain in

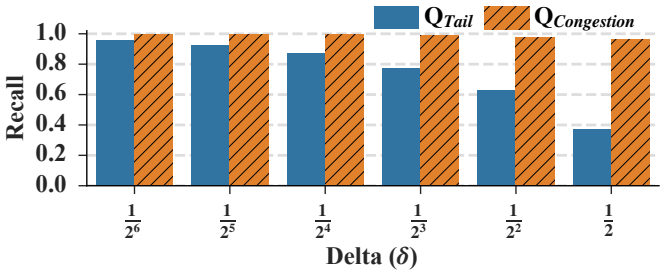


Fig. 5. Q_{Tail} and $Q_{Congestion}$ Recall

overhead reduction. However, this savings in overhead comes at the cost of degrading the quality of results of the monitoring queries as we will discuss in Section IV-D.

We also present the distribution of per-packet overhead incurred by both LINT and INT in Fig. 4(b). The boxes in this figure represent the quartiles of the distribution while the bars' endpoints represent the extremes of the distribution. Although LINT's overhead in the extreme can be as bad as INT, however, the higher quartiles are significantly smaller for LINT. For instance, for $\delta = 2^{-4}$, LINT reduces the 75th-percentile overhead of INT from ≈90% to ≈60%.

D. Query Performance

We demonstrate LINT's capability of retaining useful information even after deciding not to report some observations of the telemetry data items. We evaluate Q_{Tail} and $Q_{Congestion}$ using recall and $Q_{Latency}$ and Q_{Queue} using NRMSE.

1) Q_{Tail} and $Q_{Congestion}$: In Fig. 5, we present the recall of queries Q_{Tail} and $Q_{Congestion}$ computed using the telemetry data obtained through LINT and compared against the ground truth. As noted earlier, δ is a tuning knob to find a trade-off between overhead and accuracy, which is also evident in this figure. Increasing δ causes recall of both of the queries to degrade. For Q_{Tail}, the recall starts to degrade slowly and then falls sharply. This is because errors due to selectively reporting telemetry data items causes the tail latency threshold to diverge further from that computed using the ground truth.

However, $Q_{Congestion}$ can tolerate more noise in the data as long as the ratios of latency values collected from different switches remain similar. As a result, even after estimating some of the missing values with EWMA, $Q_{Congestion}$ retains a very high recall. For instance, for the highest δ used in our experiments ($\delta = 2^{-1}$) $Q_{Congestion}$'s recall is still more than 0.95, only a few points below its best recall (for $\delta = 2^{-6}$). However, the precision of $Q_{Congestion}$ degrades with higher δ (not shown), increasing the chances of raising false alarms.

Comparing between the quality of the query results and the overhead reduction, we find $\delta = 2^{-5}$ to be a good trade-off in our experiment setting. With $\delta = 2^{-5}$, we still have a substantial overhead reduction of about 25% while maintaining a recall above 0.9 for both of the queries.

2) $Q_{Latency}$ and Q_{Queue}: We present the NRMSE of the results of $Q_{Latency}$ and Q_{Queue} in Fig. 6 and Fig. 7, respectively. For $Q_{Latency}$, we compute the NRMSE of the total hop latency for all INT reports collected through LINT (Fig. 6). For Q_{Queue}, we compute the NRMSE of the stream of queue occupancy

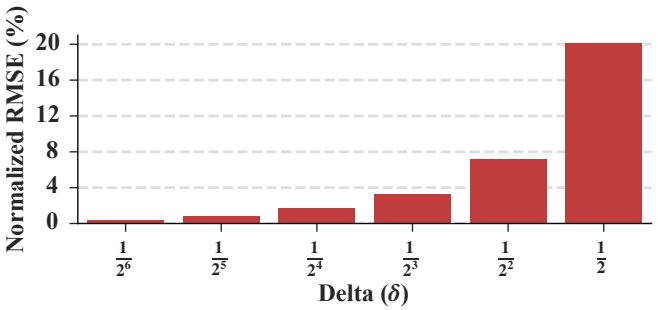


Fig. 6. NRMSE of total hop latency

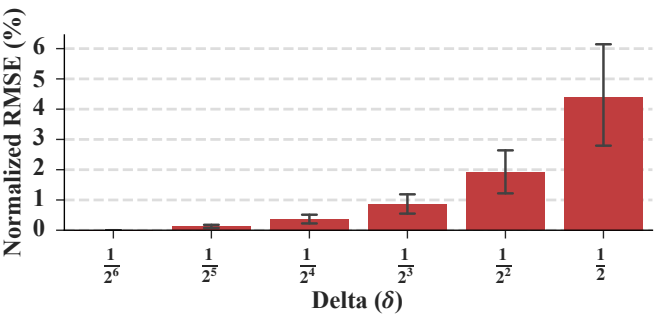


Fig. 7. Mean NRMSE of queue occupancy for all switches

observations for each of the switches collected through LINT and present the mean along with standard deviation across the switches in Fig. 7. We observe a similar trend as the previous queries, *i.e.*, a higher δ degrades the quality of the query results. However, for the previously identified operating point, $\delta = 2^{-5}$, the NRMSE is minute for both $Q_{Latency}$(less than 2%) and Q_{Queue}(less than 0.25%).

E. LINT-flow Performance and Trade-offs

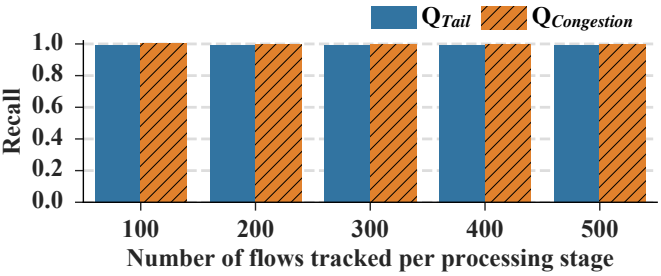


Fig. 8. Impact of number of track flows per-stage on Q_{Tail} and $Q_{Congestion}$

We demonstrate the impact of having limited memory on LINT-flow by fixing the number of simultaneously tracked flows per processing stage (between 100 and 500) and employing least recently used (LRU) eviction policy for the old flows. We present the results on Q_{Tail} and $Q_{Congestion}$ recall in Fig. 8. For these results we set δ to 2^{-5}. Our first observation is that considering flow-context while selectively reporting telemetry data items substantially improves the recall for both queries. Although not shown here for space constraints, the same holds for higher δ as well. However, the number of flows that can be simultaneously tracked at each stage had very little impact on the recall. We also observed similar behavior for overhead reduction. This behavior can be attributed to a combination of

factors such as the short-lived nature of the flows, the use of LRU policy to exclude the old flows, and the reduction in IP address entropy due to rewriting the IP addresses in the trace with the ones of the Mininet hosts in the network. However, we plan to investigate further to identify the root cause.

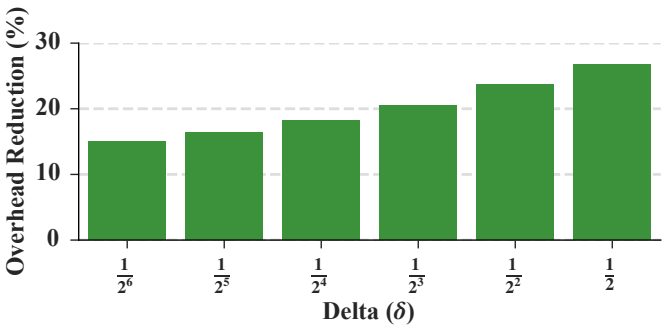


Fig. 9. Overhead reduction by LINT-flow

Even for LINT-flow, the overhead reduction is dominated by the δ parameter. We present results on overhead reduction in Fig. 9 by fixing the number of flows simultaneously tracked at each stage to 100 and varying δ. Indeed, considering flow-context leads to reporting more telemetry data items for keeping the per-flow error estimate within bounds, consequently, reducing the gain. However, the overhead reduction still remains within 15% –25% range.

V. Related Works

INT promises to provide unprecedented visibility into the network that was not possible with traditional network monitoring technologies. Since the release of initial specification many applications of INT for network operations and management have been proposed, including for failure detection [37], [38], congestion control [12] and tracking the data plane rules matched by the flows [11], [39], among others. Although INT was initially proposed for IP networks, several extensions of INT have been proposed such as for wireless networks [40] and multi-layer IP-over-Optical networks [41]–[43]. It is worth mentioning that INT is one of several concurrent efforts towards performing in-band network telemetry using live network traffic (*cf.* In-situ Operations, Administration, and Maintenance (IOAM) standardization effort within the IETF [44]).

A classic issue in network monitoring, also applicable to INT is the trade-off between monitoring accuracy and the overhead of monitoring. Several research works have attempted to address this issue for INT from different perspectives. For instance, some research works have proposed to use INT only for specially crafted probe packets instead of for live network traffic [45], [46]. Probing the network in this way requires carefully crafting the probe packets and planning the probe paths for maximum network coverage. Pan *et al.*, addresses this problem by proposing an optimization based approach in [45]. However, probe packets are often not subjected to the same treatment as the live network traffic, therefore, can obtain an incorrect view of the network.

Several approaches have been proposed to reduce INT overhead for live network traffic. For instance, Marques *et al.*, have proposed an offline optimization approach for INT in [47]. Their approach assumes the knowledge of all network flows and devises an offline schedule for what telemetry data item should be collected by packets of which flows while considering constraints such as MTU limitations. An online approach for reducing INT overhead is presented by Tang *et al.*, in [17]. They implement INT capabilities in Open vSwitch and employ sampling for deciding which packets should be subjected to INT along the way. A central controller adjusts the sampling rate at the end hosts and configures a watchlist of flows to monitor. In contrast to these aforementioned approaches, we propose an online mechanism that works completely in the data plane without the intervention of a centralized controller. Also, each switch independently decides on which telemetry data items to report without any global coordination. In contrast to the sampling-based approaches such as those presented in [17], [37], we propose to adapt telemetry data reporting based on error estimates computed within the data plane devices.

Very recently PINT [18] proposed a randomized algorithm for INT. PINT fixes the bit overhead allowed on a packet for INT. Then, each network device makes a random decision for embedding INT data. Since switches randomly decide on embedding INT data, therefore, the requested telemetry data items can be reported across multiple packets. PINT also proposes mechanisms for minimizing the number of packets required to collect all required telemetry data items. PINT is effective for executing network monitoring queries that work with aggregate data and when network flows are not short-lived. In contrast to PINT, we propose a complimentary approach for supporting network monitoring queries that rely on per-hop telemetry data and is oblivious to flow duration.

VI. Conclusion

In this paper, we presented LINT, an accuracy-adaptive and lightweight INT mechanism. LINT operates entirely in the data plane without any control plane intervention and without any global co-ordination. We also proposed LINT-flow, an extension of LINT that takes each packet's flow-context into consideration for selectively reporting telemetry data. We evaluated LINT using a real data-center traffic trace. Our evaluation results demonstrated the effectiveness of LINT in reducing data plane overhead by 25% while maintaining more than 0.9 recall for network monitoring queries trying to identify flows with high latency and flows with congested switches in the network. We plan to have a full-fledged implementation of LINT on commercially available PISA devices. We also plan to investigate the use of information theory measures for quality of information aware adaptive telemetry collection.

Acknowledgement

This work was supported in part by an NSERC Discovery grant and in part by the INRIA International Chair in Network Softwarization program.

REFERENCES

[1] T. Holterbach, E. C. Molero, M. Apostolaki, A. Dainotti, S. Vissicchio, and L. Vanbever, "Blink: Fast connectivity recovery entirely in the data plane," in *USENIX NSDI*, 2019, pp. 161–176.

[2] I. F. Akyildiz, A. Lee, P. Wang, M. Luo, and W. Chou, "A roadmap for traffic engineering in sdn-openflow networks," *Computer Networks*, vol. 71, pp. 1–30, October 2014.

[3] V. Sekar, N. G. Duffield, O. Spatscheck, J. E. van der Merwe, and H. Zhang, "Lads: Large-scale automated ddos detection system." in *USENIX ATC*, 2006, pp. 171–184.

[4] J. Boite, P. Nardin, F. Rebecchi, M. Bouet, and V. Conan, "Statesec: Stateful monitoring for ddos protection in software defined networks," in *IEEE NetSoft*, 2017, pp. 1–9.

[5] B. Claise and R. Wolter, *Network Management: Accounting and Performance Strategies*. Cisco Press, 2006.

[6] W. Stallings, *SNMP, SNMPv2, and CMIP: The practical guide to network management*. Addison-Wesley Longman Publishing Co., 1993.

[7] A. Gupta, R. Harrison, M. Canini, N. Feamster, J. Rexford, and W. Willinger, "Sonata: Query-driven streaming network telemetry," in *ACM SIGCOMM*, 2018, p. 357–371.

[8] F. Paolucci, A. Sgambelluri, F. Cugini, and P. Castoldi, "Network telemetry streaming services in sdn-based disaggregated optical networks," *Journal of Lightwave Technology*, vol. 36, no. 15, pp. 3142–3149, 2018.

[9] H. Song, F. Qin, P. Martinez-Julia, L. Ciavaglia, and A. Wang, "Network telemetry framework," Working Draft, IETF Secretariat, Internet-Draft draft-ietf-opsawg-ntf-04, September 2020. [Online]. Available: http://www.ietf.org/internet-drafts/draft-ietf-opsawg-ntf-04.txt

[10] T. P. A. W. Group, "In-band Network Telemetry (INT) data plane specification," June 2020. [Online]. Available: https://github.com/p4lang/p4-applications/blob/master/docs/INT_v2_1.pdf

[11] "Barefoot deep insight™– solution brief," Barefoot Networks, White paper, 2018. [Online]. Available: https://www.barefootnetworks.com/static/app/pdf/DI-UG42-003ea-ProdBrief.pdf

[12] Y. Li, R. Miao, H. H. Liu, Y. Zhuang, F. Feng, L. Tang, Z. Cao, M. Zhang, F. Kelly, M. Alizadeh, and M. Yu, "HPCC: High precision congestion control," in *ACM SIGCOMM*, 2019, p. 44–58.

[13] "Cisco streaming telemetry." [Online]. Available: https://developer.cisco.com/docs/ios-xe/#!streaming-telemetry-quick-start-guide/streaming-telemetry

[14] "Int using netronome agilio cx smartnic." [Online]. Available: https://www.netronome.com/blog/in-band-network-telemetry-its-not-rocket-science/

[15] Y. Feng, S. Panda, S. G. Kulkarni, K. K. Ramakrishnan, and N. Duffield, "A smartnic-accelerated monitoring platform for in-band network telemetry," in *IEEE LANMAN*, 2020, pp. 1–6.

[16] "Making the switch: Disruptive telecom white box collaboration accelerates and opens the platform, powering unprecedented network performance and insights," April 2017. [Online]. Available: https://about.att.com/story/white_box_collaboration.html

[17] S. Tang, D. Li, B. Niu, J. Peng, and Z. Zhu, "Sel-int: A runtime-programmable selective in-band network telemetry system," *IEEE Transactions on Network and Service Management*, vol. 17, no. 2, pp. 708–721, 2020.

[18] R. Ben Basat, S. Ramanathan, Y. Li, G. Antichi, M. Yu, and M. Mitzenmacher, "Pint: Probabilistic in-band network telemetry," in *ACM SIGCOMM*, 2020, p. 662–680.

[19] A. Roy, H. Zeng, J. Bagga, G. Porter, and A. C. Snoeren, "Inside the social network's (datacenter) network," in *ACM SIGCOMM*, 2015, p. 123–137.

[20] T. Benson, A. Akella, and D. A. Maltz, "Network traffic characteristics of data centers in the wild," in *ACM IMC*, 2010, p. 267–280.

[21] "The CAIDA UCSD anonymized internet traces - 2016 - 2016/04/06 13:19:00 utc." [Online]. Available: https://www.caida.org/data/passive/passive_dataset.xml

[22] P. Bosshart, G. Gibb, H.-S. Kim, G. Varghese, N. McKeown, M. Izzard, F. Mujica, and M. Horowitz, "Forwarding metamorphosis: Fast programmable match-action processing in hardware for sdn," in *ACM SIGCOMM*, 2013, pp. 99–110.

[23] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese, and D. Walker, "P4: Programming protocol-independent packet processors," *ACM SIGCOMM Computer Comm. Rev.*, vol. 44, no. 3, pp. 87–95, 2014.

[24] M. Al-Fares, A. Loukissas, and A. Vahdat, "A scalable, commodity data center network architecture," in *ACM SIGCOMM*, 2008, p. 63–74.

[25] K. Bakhshaliyev, M. A. Canbaz, and M. H. Gunes, "Investigating characteristics of internet paths," *ACM Trans. Model. Perform. Eval. Comput. Syst.*, vol. 4, no. 3, September 2019.

[26] "Introducing data center fabric, the next-generation facebook data center network," November 2014. [Online]. Available: https://engineering.fb.com/production-engineering/introducing-data-center-fabric-the-next-generation-facebook-data-center-network/

[27] A. Deshpande, C. Guestrin, S. R. Madden, J. M. Hellerstein, and W. Hong, "Model-driven data acquisition in sensor networks," in *VLDB*, 2004, pp. 588–599.

[28] D. Goldsmith and J. Brusey, "The spanish inquisition protocol—model based transmission reduction for wireless sensor networks," in *IEEE SENSORS*, 2010, pp. 2043–2048.

[29] G. M. Dias, B. Bellalta, and S. Oechsner, "A survey about prediction-based data reduction in wireless sensor networks," *ACM Comput. Surv.*, vol. 49, no. 3, Nov. 2016.

[30] N. K. Sharma, A. Kaufmann, T. Anderson, A. Krishnamurthy, J. Nelson, and S. Peter, "Evaluating the power of flexible packet processing for network resource allocation," in *USENIX NSDI*, 2017, pp. 67–82.

[31] A. C. Lapolli, J. Adilson Marques, and L. P. Gaspary, "Offloading real-time ddos attack detection to programmable data planes," in *IFIP/IEEE IM*, 2019, pp. 19–27.

[32] X. Jin, X. Li, H. Zhang, R. Soulé, J. Lee, N. Foster, C. Kim, and I. Stoica, "Netcache: Balancing key-value stores with fast in-network caching," in *ACM SOSP*, 2017, p. 121–136.

[33] V. Sivaraman, S. Narayana, O. Rottenstreich, S. Muthukrishnan, and J. Rexford, "Heavy-hitter detection entirely in the data plane," in *ACM SOSR*, 2017, pp. 164–176.

[34] J. Sonchack, A. J. Aviv, E. Keller, and J. M. Smith, "Turboflow: Information rich flow record generation on commodity switches," in *ACM EuroSys*, 2018, pp. 1–16.

[35] Z. Xiong and N. Zilberman, "Do switches dream of machine learning? toward in-network classification," in *ACM HotNets*, 2019, p. 25–33.

[36] J. S. Hunter, "The exponentially weighted moving average," *Journal of quality technology*, vol. 18, no. 4, pp. 203–210, 1986.

[37] T. Pan, E. Song, C. Jia, W. Cao, T. Huang, and B. Liu, "Lightweight network-wide telemetry without explicitly using probe packets," in *IEEE INFOCOM Workshops*, 2020, pp. 1354–1355.

[38] C. Jia, T. Pan, Z. Bian, X. Lin, E. Song, C. Xu, T. Huang, and Y. Liu, "Rapid detection and localization of gray failures in data centers via in-band network telemetry," in *IEEE/IFIP NOMS*, 2020, pp. 1–9.

[39] S. Wang, Y. Chen, J. Li, H. Hu, J. Tsai, and Y. Lin, "A bandwidth-efficient int system for tracking the rules matched by the packets of a flow," in *IEEE GLOBECOM*, 2019, pp. 1–6.

[40] P. Janakaraj, P. Pinyoanuntapong, P. Wang, and M. Lee, "Towards in-band telemetry for self driving wireless networks," in *IEEE INFOCOM Workshops 2020*, 2020, pp. 766–773.

[41] M. Anand, R. Subrahmaniam, and R. Valiveti, "Point: An intent-driven framework for integrated packet-optical in-band network telemetry," in *IEEE ICC*, 2018, pp. 1–6.

[42] S. Tang, J. Kong, B. Niu, and Z. Zhu, "Programmable multilayer int: An enabler for ai-assisted network automation," *IEEE Communications Magazine*, vol. 58, no. 1, pp. 26–32, 2020.

[43] B. Niu, J. Kong, S. Tang, Y. Li, and Z. Zhu, "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, 2019.

[44] F. Brockners, S. Bhandari, and T. Mizrahi, "Data fields for in-situ oam," Working Draft, IETF Secretariat, Internet-Draft draft-ietf-ippm-ioam-data-10, July 2020. [Online]. Available: http://www.ietf.org/internet-drafts/draft-ietf-ippm-ioam-data-10.txt

[45] T. Pan, E. Song, Z. Bian, X. Lin, X. Peng, J. Zhang, T. Huang, B. Liu, and Y. Liu, "Int-path: Towards optimal path planning for in-band network-wide telemetry," in *IEEE INFOCOM*, 2019, pp. 487–495.

[46] Y. Lin, Y. Zhou, Z. Liu, K. Liu, Y. Wang, M. Xu, J. Bi, Y. Liu, and J. Wu, "Netview: Towards on-demand network-wide telemetry in the data center," *Computer Networks*, vol. 180, p. 107386, 2020.

[47] J. A. Marques, M. C. Luizelli, R. I. T. da Costa Filho, and L. P. Gaspary, "An optimization-based approach for efficient network monitoring using in-band network telemetry," *Journal of Internet Services and Applications*, vol. 10, no. 1, p. 12, 2019.