A SmartNIC-Accelerated Monitoring Platform for In-band Network Telemetry

Yixiao Feng[‡], Sourav Panda[†], Sameer G Kulkarni[†], K. K. Ramakrishnan[†], Nick Duffield[‡]
[†]University of California, Riverside, [‡]Texas A&M University

***Abstract*—Recent developments in In-band Network Telemetry (INT) provide granular monitoring of performance and load on network elements by collecting information in the data plane. INT enables traffic sources to embed telemetry instructions in data packets, avoiding separate probing or infrequent management-based monitoring. INT sink nodes track and collect metrics by retrieving INT metadata instructions appended by different sources of INT information. However, tracking the INT state in packets arriving at the sink is both compute intensive (requiring complex operations on each packet), and challenging for the standard P4 match-action packet processing pipeline to maintain line-rate. We propose a network telemetry platform in which the INT sink is implemented using distinct (C-based) algorithms on a SmartNIC in the monitoring host, complementing the P4 packet processing pipeline. This design accelerates packet processing and handles complex INT-related operations more efficiently than P4 match-action processing alone. While the P4 pipeline parses INT headers, a general-purpose Micro-C algorithms performs complex INT tasks (*e.g.* aggregation, event-detection, notification, *etc.*). We demonstrate that partitioning of INT processing significantly reduces processing overhead vs. a P4-only implementation, providing accurate, timely and almost loss-free event notification.**

***Index Terms*—In-band network Telemetry, Network Monitoring, P4**

I. Introduction

Network monitoring provides information crucial for the control, operation and management of computer networks. Over the last few years, In-band Network Telemetry (INT) [1] has represented an important development in network monitoring that supports collecting and reporting network state in the data plane. INT enables traffic source to embed telemetry instructions in data packets. This capability will improve on long-standing practices of management based monitoring via relatively infrequent SNMP-based polling or event detection via RMON [2], or performance measurement using probe packets, by providing timely and precise measurements from user packets in the network data plane. In addition to enabling detailed and granular monitoring of network performance, the information provided through INT can potentially improve congestion control and alert to the occurrence of complex events defined by the cumulative information of a packets traversal of network elements.

The advent of programmable switches has been a significant development, both for enabling acquisition of information from packets, and to enable both control *and* data paths to adapt the dynamic conditions in the network informed through monitoring. In particular, the P4 [3] language provides a framework with which to program switches to process packet flows more flexibly with a richer match/action functionality than was previously possible. In the context of INT, P4 is able to leverage recent advances in the compute capabilities of the networking devices such as switches, routers and end host network interface cards (NICs) to enhance monitoring capability. Specifically, additional packet fields can be introduced for each switch (we use the generic term, switch, to refer to both layer-2 switches and layer 3 routers) in the network to provide monitoring information. These fields are embedded within the data packets and switches are able to use them to insert granular telemetry information concerning the performance and functioning of the network switches and links in the data path. Thus, to provide the most actionable information to an end system requires processing all INT packet fields populated by switches in a packet's path and providing that information in a timely manner can be a significant challenge.

With the need for ever higher link bandwidths in datacenters (DC), servers increasingly depend on smartNICs (sNIC) to offload network packet processing[1] because of the performance boost they can provide. The main challenge that our work addresses is *how to design a cost-efficient traffic measurement and analysis infrastructure that can act as a monitoring sink platform for In-Network Telemetry (INT) to provide loss-free and timely INT notification of complex events dependent on entire packet paths to the monitoring host by leveraging the sNIC.* Our work addresses a gap in current approaches, which do not achieve scalable cost-efficient INT packet processing at the monitoring sink. Specifically, most of the existing solutions [4], [5], [6] rely on the host-CPU to perform INT packet processing and event detection and typically are unable to achieve high-performance (line-rate) INT event processing.

In this work, we present the architecture, design and implementation of a high-performance INT processing and INT event detection framework, in which we partition INT processing between the sNIC and the monitoring host. The key advantage of this architecture, compared to existing solutions, is the ability of the sNIC to provide key processing functions – transformation and application level steering – that are much more difficult to support in a programmable switch, while relieving the host CPU of the requirement to process packets in software at full line rate. We illustrate the scheme to collect and aggregate INT information across the packets of different network flows (*i.e.,* network flows identified by

[1]SmartNICs are the programmable and extensible network interface cards (NICs) that provide the network traffic processing capabilities within the NIC, allowing the CPU to program and offload certain packet processing to be performed within the NIC

978-1-7281-8154-7/20/$31.00

unique IP 5-tuple) within the sNIC and export the INT metrics to the monitoring host. This allows us to improve packet processing rates by a factor of around 3 compared with rates reported in the literature [6] by having the sNIC relieve the packet processing demands on the host CPU. Furthermore, we propose mechanisms to program the INT event parameters on the monitoring framework to have the sNIC detect and export INT events to the monitoring host in a timely manner. The host can then process these events to take necessary data/control plane measures, readjust/reprogram the INT event thresholds on the sNIC, and also timestamp and store the event information (time-series data) in externalized database (Redis).

Our work incorporates the most recent advances and specifications of INT v2.0 [7]. Our frameworks support all INT metrics (packet path information: switch id, hop latency, queue occupancy, congestion *etc.*) listed in the current INT specification and is also extensible to support additional metrics as necessary. Our approach not only enables us to achieve line-rate packet processing, but also drastically reduces the host CPU consumption for INT-related processing.

To summarize, the key contributions of our work include:

- **Flexible partitioning and INT processing offload**: The INT monitoring functionality is judiciously split between the sNIC and host to cooperatively process the INT packets and INT events at full line rate. With a large cohort of much less powerful microengines (MEs), the sNIC performs INT packet processing in-path at line rate (with the MEs performing the processing in parallel) to extract the key flow information, while intelligently avoiding slowdowns from locks used for coordination;
- **INT data and Event Aggregation**: The sNIC processes each flow to collect and aggregate INT information across different packets in a hash table and also transforms the INT packet data to INT events and notifies the events in a timely manner to the monitoring host;
- **Adaptable and Mutable INT data and event support**: Our platform provides a data structure to facilitate platform specific INT data collection that is adaptable to custom requirements and allows for filtering the INT events based on configured thresholds, while allowing us to mutate threshold parameters dynamically based on the INT event processing rate and characteristics of the monitoring platform.

II. Background And Related Work

A. SmartNIC architecture overview and capabilities

In this work, we used the Netronome SmartNIC [8] for offloading critical INT processing functions. In addition to network processing capabilities of traditional NICs, *e.g.,* checksum, segmentation and reassembly, the sNIC also has 60 flow processing cores (that run at 633MHz), also known as Microengines (MEs), for implementing the more complex network data plane functionality. This may include encryption/decryption, flow table (match-action) processing, traffic shaping, security and traffic analysis functions, *etc.* The Netronome sNIC supports P4 for programmable parsing (header extraction), which is conducted by a subset of MEs, known as the packet processing cores. We dedicate 54 MEs for packet processing pipeline and INT header processing. A global load balancer on the sNIC distributes the incoming packets among these 54 PMEs based on a credit-based scheme to provide considerable parallelism for high throughput packet processing. The rest of the MEs run dedicated MicroC programs *i.e.,* separate functions executed asynchronously to enable stateful packet processing within the sNIC. (e.g., stateful event processing, micro-burst detection, *etc.*).

The sNIC has a hierarchical memory sub-system, providing a large external memory $2GB$, which we leverage to implement custom data structures (user configured) for caching flow specific INT data (called a FlowCache) for several flows, while aggregating INT metrics across packets of a flow. We also use the memory to store the P4/SDN match action tables.

B. INT and telemetry report specification

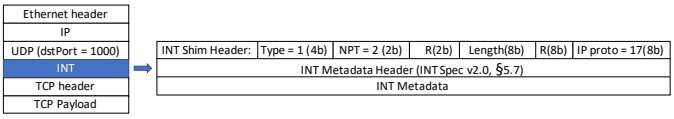


Fig. 1: INT over TCP packets

INT [7] provides real-time, fine-grained, end-to-end network monitoring in the dataplane [9]. Our platform currently supports INT processing and monitoring for the INT-MD type where the INT data is carried along with the packet.

INT Header format and location: The current specification [7] has multiple ways to place the INT header and metadata fields. To allow support for INT even with SSL *i.e.,* encrypted TCP data, we choose the INT with new UDP header encapsulation approach for TCP packets where we insert a new UDP header, INT header and INT data in between the original Layer-3 (IP) and Layer-4 (TCP) headers. However, for UDP packets we leverage the original UDP header, as per the specification. For both cases, we set the destination port of the outer UDP header to INT_PORT (0x1000) to indicate the presence of INT data. Although this approach requires the sink node to perform different processing for TCP vs. UDP packets (which we feel is not ideal), we seek to be consistent with the current specification. The primary motivation for the different treatment for UDP packets in the specification appears to be to try and minimize the overhead for inserting the INT headers with UDP packets. Fig. 1 shows the INT shim header, which is 4 bytes long, followed by the INT metadata header of 12 bytes. After the INT shim and metadata header, each INT hop adds the same length of metadata[7] as set in the metadata header. In our experiment settings, every packet traverses a fixed path of 5 INT intermediate switches.

INT Event and Telemetry Reporting At the INT sink, the INT metadata is extracted and telemetry reports are generated and communicated to the monitoring host based on the guidelines detailed in P4 telemetry report specification[7]. Further, we incorporate additional mechanisms to distinctly report the aggregate information for the regular INT telemetry data, and

to provide specific INT event notifications along with the associated INT data to the monitoring host.

C. Related Work

A large number of works [10], [11], [12], [13], [14], [15] have addressed different aspects of processing and collection of INT packets. Here, we focus primarily on INT monitors.

INT Monitors and Event Processing: IntMon [16] implements an INT monitoring service on the Open Networking Operating System (ONOS). However, it achieves very low processing rates and high cpu utilization. IntCollector [9] also uses UDP encapsulation for INT packets and the monitor reports INT change events based on a predefined threshold. However the INT packet processing and event detection are implemented on the host CPU, splitting INT packet processing into a fast path (accelerated by an eXpress Data Path (XDP)) and a slow path for exporting and inserting INT events into a database. Because of the packet processing being done on the host CPU, performance is limited. The work in [5] is closest to ours. They implement the INT packet processing and INT event detection using the sNIC P4 pipeline. But, they only report simple threshold-crossing INT events to the stream processor (running on the host CPU) using the kernel bypass technique AF XDP. However, the use of P4 pipeline restricts the per-flow state information to simple registers and counters only, and does not give us the ability to maintain complex per flow state that are required by most server-based networking applications [17]. Also, additional miscellaneous functions such as timeouts *etc.* are not easily implementable using P4 [17]. By using callable C functions and P4, we design a highly efficient INT monitoring platform that not only supports notification of INT events, but also exports the basic INT telemetry report for every INT packet.

III. Design And Implementation

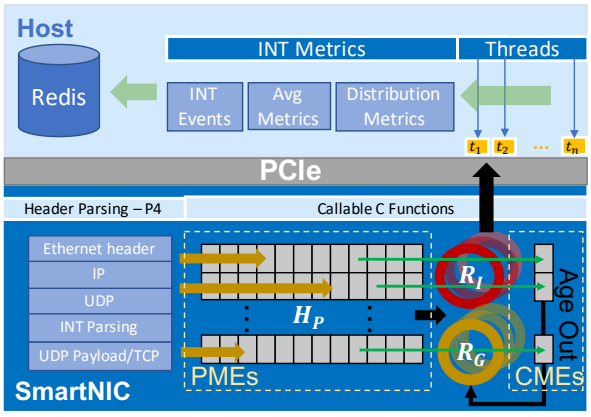


Fig. 2: Architecture

A. Architecture and Data Structure

Fig. 2 presents the high-level architecture of our INT monitoring platform. While generally an INT sink is the 'last' switch and the monitoring node is a host connected over a link to that switch, we propose an architecture that utilizes a combination of a sNIC and host (commercially available off-the-shelf server) to serve as combined INT sink and monitoring node. Packets flow through the monitor (which acts as a line-rate bump-in-the-wire) to the destination. The packet processing pipeline in the sNIC takes advantage of P4 packet parsing, match-action rules and Micro-C algorithms to achieve fine-grained INT traffic analysis at line rate. The sNIC aggregates the INT metrics across multiple packets of a flow and collects the INT metrics for a large number ($\sim 6Million$) of flows in the INT Flowcache. We utilize the general-purpose Micro-C algorithms to perform more complex INT tasks (*e.g.,* averaging across INT messages per flow, event detection, and notification). We describe below the key components, data structure and algorithms that seek to achieve an accurate, timely, and close to loss-free INT monitoring platform.

B. sNIC Data Structure and operations

Fig. 2 also shows the data structure on sNIC and the corresponding operations for INT packet processing. We allocate a large hash table (a.k.a. INT FlowCache, H_P), with 2^{19} rows on the sNIC memory and 12 buckets per row to accommodate hash collisions. The IP 5-tuple of the incoming packet header is hashed to identify the row index in the hash table. Each bucket in the hash table includes a flow key (*i.e.,* 5-tuple), packet count, the timestamp of the most recent update and the INT metrics. The INT metrics consists of the most recent as well as the average statistics of the INT metadata (*i.e.,* switch ID, hop latency, queue occupancy, link utilization) for each INT transit node. We allocate 52 micro engines for the packet processing pipeline (PMEs) and dedicate 2 micro engines for custom processing (CMEs). We leverage the CMEs to age out the entries in H_P that have not been updated over a predefined time period. Based on the available credits at each of the PMEs, a global load balancer on the sNIC distributes incoming packets among these 52 PMEs. This allows all PMEs to process packets in parallel. (Each ME has 4 threads and each of these threads may process the packets concurrently). Hence, to guarantee consistency and accurate operations on primary hash table, each thread locks the corresponding row entry to update or evict to host.

Ring buffers: We leverage the ring buffers to export necessary INT information to the monitoring host. Each ring buffer is configured to hold 64K entries. We use a total of 8 general ring buffers (R_G) to export an evicted bucket (INT entry of a flow) to the host (*i.e.,* upon collision and no free bucket in the row, we evict one bucket (least recently updated) to make space for a new flow.) We also use 8 INT event ring buffers (R_I) to export the INT events that are generated by the packet processing pipeline. The collected INT flow metrics would be inaccurate if the ring buffer overflows. We found that 64K entries for each ring buffer is sufficient space to prevent evictions from overflow on the ring buffer.

Packet Processing and key operations: An incoming flow might result in one of the two operations in H_P: 1) Update operation: the incoming INT flow matches one of the keys in H_P or, has no match but has an empty bucket in the row in H_P. 2) Eviction: all the buckets of a row in INT FlowCache

are occupied and the incoming INT flow does not match any keys in H_P. In this scenario, one existing entry (LRU) is evicted to the corresponding general ring buffer R_G. In either case (1 or 2), the PME thread will update the data structure and update the most recent as well as the average statistics of the INT metadata (*i.e.,* switch ID, hop latency, queue occupancy, link utilization) for each INT transit node. In addition, an INT data update may result in one or more INT event notification operations, as described below.

C. INT Data and Event Notification

Our INT Flowcache aggregates and collects INT information carried in each packet of the flow on the sNIC. For every packet, we extract the INT metrics (*i.e.,* the four metadata fields: switch ID, hop latency, queue occupancy, link utilization) embedded by each INT transit node. Further, when the incoming packet's INT data for a metric at any transit node exceeds a predefined threshold–configurable for each INT metric for each of the $\mathcal{T}$ and $\mathcal{C}$ events–an INT event (*i.e.,* event $\mathcal{T}$ and/or event $\mathcal{C}$ described below) is generated and notified to the monitoring host through the INT event ring buffer R_I. For each INT event, we store the corresponding INT fields (*i.e.,* switch ID, measurement value at that switch, bitmap indicating the event type and the timestamp) in the ring buffer R_I. In the flow entry in H_P, we also track the average for each of the metrics for each of the switches in the path.

We specify two broad classes of INT events: 1) Change events $\mathcal{C}$, and 2) Threshold-crossing events $\mathcal{T}$. Change events, $\mathcal{C}$, occur when the difference between the previous and the current INT metadata value (*i.e.,* hop latency, queue occupancy, or link utilization) exceed a predefined threshold. Threshold-crossing events, $\mathcal{T}$, occur when the current INT metadata value exceeds a predefined threshold. We further categorize Event set for $\mathcal{C}$ and $\mathcal{T}$ into a) Per switch and b) End-to-end.

Per switch: Since we store the INT metadata value for each of the INT transit nodes (switches), we can account for i) the current INT metadata value that exceeds the predefined threshold; ii) the change in INT metadata value for these switches by comparing the previous and current reported values for the same flow. We generate an event when current absolute value or the difference between two values of the same metric exceeds a predefined threshold.

End-to-end The use of Micro-C allows us to also compute the aggregate (end-to-end measure) across all the INT transit node reported values, especially for the hop latency and also to build the path information by concatenating the IDs of each of the transit nodes. We generate an event when either the current aggregate value exceeds the threshold or the difference (previous and current) for the aggregated hop latency exceeds the predefined threshold.

Tracking Distribution for INT Metrics Each switch reports metrics in INT packets, and it is desirable to collect the distribution of the INT metadata by storing the occurrence for each possible value, or a range of values, over time. We allocate three arrays for the three INT measurements (*i.e.,* hop latency, queue occupancy, link utilization) per switch in the sNIC memory. Each entry in the array has a counter for a certain range of values (bin) of an INT metric, and the counter is incremented by 1 each time the observed metadata value corresponds to the bin's range. The distribution of each measurement per switch can provide important information on switch and path status (*e.g.,* determine the bottleneck by identifying the switch with the worst hop latency).

D. Host Data Structure and operations

The monitoring host reads the exported INT information including the flow statistics from sNIC. The INT notificationsfrom the sNIC are also periodically flushed to the Redis database at the host. We dedicate a number of host threads (up to 10 CPU cores/threads) for reading from the sNIC rings R_G, R_I and the distribution arrays plus a CPU core to flush the INT metrics to the Redis [18] database.

IV. Evaluation

A. Evaluation Setup

1) Testbed: We evaluate our monitoring platform on a Linux (kernel version 4.4.0-142) server with 10 Intel Xeon 2.20GHz CPU cores and 256GB memory and Netronome Agilio 4000 CX Dual-Port 10 Gigabit sNICs.

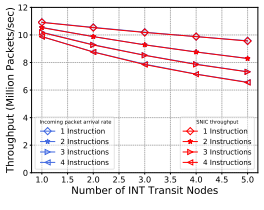
2) Evaluation Trace: We use a publicly available packet trace from CAIDA 2019 [19] containing about 186 millions packets over a 5 minute interval. We speed up the trace by reducing the packet size to 64 bytes, to achieve the highest packet arrival rate. The INT headers are inserted into every packet using the UDP encapsulation option, and emulate each transit node and the number of metrics (INT instructions) by creating a version of the trace with the corresponding number of INT metadata fields for each packet. We simulate a number of INT transit nodes, varying from 1 to 5 and a number of instructions varying from 1 to 4 for each packet.

3) Synthetic INT metadata: Assuming for example that there are 5 INT transit nodes and 4 instructions on each node, the values embedded inside each packet are randomly generated according to a exponential distributed probabilities across all possible values for each measurements (*i.e.,* hop latency, queue occupancy, link utilization). For example, the queue occupancy takes values from 0 to 300 with exponential decreasing probabilities. The INT layer and corresponding metadata are inserted into the 64-byte CAIDA 2019 trace to be used in our evaluation. For our experiments, we use MoonGen [20] as pcap trace replay tool.

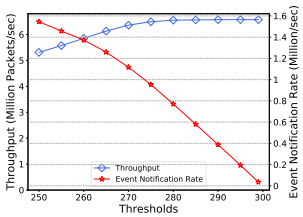
B. Throughput and INT Events Rate

When traffic arrives at the sNIC, we process the packet through the packet processing pipeline and update the primary hash table H_P with the flow information, count, timestamp as well as the INT metadata. Fig. 3(a) shows the throughput achieved through our system (packets being processed and then routed back to the MoonGen packet generator. The throughput essentially overlaps the incoming packet arrival rate at sNIC, for different numbers of INT transit nodes and varying number of INT metrics reported (instructions). The

highest incoming packet arrival rate is around 11 Mpps with 1 instruction on 1 INT transit node (64 byte packets plus the INT headers and meta-data). As the number of INT transit nodes and the number of instructions increases, the size of each packet would also increase, reflected in the lower per-packet throughput in Fig. 3(a). We are able to maintain full throughput across all of the configurations tested, and match the incoming rate even when processing multiple (4) instructions per packet at each of the 5 transit nodes. Thus, our monitoring node can fully function as a 'bump-in-the-wire'.



(a) Packet Throughput (processed by sNIC)



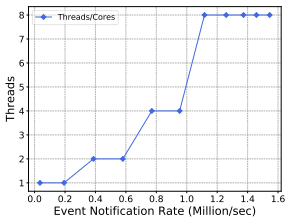
(b) Tput vs. Event Notif. Rate, varying Q len. threshold

Fig. 3: sNIC & complete system Tput vs. event notifications

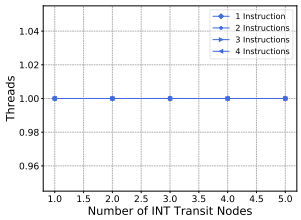
Fig. 3(b) shows the throughput achieved by our complete INT monitoring platform as the rate of INT event notifications posted successfully to the host changes when we vary the threshold for when the metric reported by an INT packet is detected as an event. The operation on receiving a packet with INT meta-data in it is to first update the primary hash table H_P (*i.e.,* update the latest INT information and update the cumulative statistics) for every packet. We also have to evict entries to the general ring buffer R_G on a hash collision, to accommodate a new flow. When the threshold for a metric is smaller, an arrival of an INT packet is more likely to generate an INT event notification to the host. While the sNIC can receive INT packets at the full line rate for different configurations of transit nodes and metrics reported in each INT packet (as shown in Fig. 3(a)), the throughput of INT packets processed through the complete platform (including notifications delivered up to the host without loss) varies depending on the threshold used to generate an event. We show the result for varying INT threshold-crossing events, since they generate a much higher notification rate than change events, thus stressing our platform more. Fig. 3(b) is for varying the queue occupancy threshold, generating a varying amount of INT notification events to R_I. Our monitoring platform can maintain full line rate (*i.e.,* 6.57 Mpps as shown in Fig. 3(a) with 5 switches and 4 instructions each) even when the notification rate is around 0.6-0.8 Million INT events per second. The throughput decreases when there is a higher rate of notifications, as this introduces more overhead on the packet processing pipeline. However, our monitoring platform can still maintain a throughput of around 5.3 Mpps when there are 1.6 Million INT event notifications to R_I.

C. Host Thread Usage

Fig. 4 shows the number of CPU threads used by the host to read the two ring buffers R_I and R_G. Fig. 4(a) shows the



(a) Host Thread Usage: R_I



(b) Host Thread Usage: R_G

Fig. 4: Number of Host Threads Required

thread usage for reading the INT ring buffers R_I at different event notification rates. The maximum threads/cores we need to allocate on the host for each ring buffer is 8, which can support a notification rate of 1.6 Million INT events per second. The host only needs 1 thread(core) to read the general ring buffer across all of the experiment scenarios.

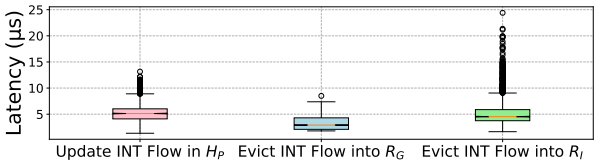


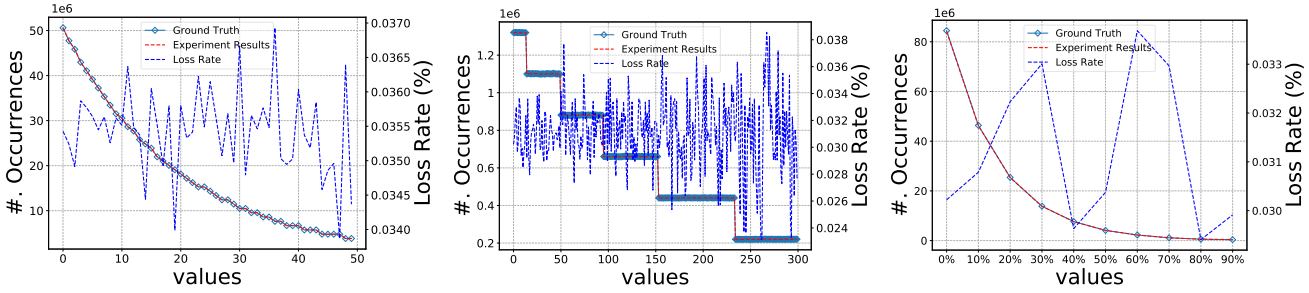
Fig. 5: INT Processing Latency

D. Latency

Fig. 5 shows the processing latency for INT flows through the different components of the packet processing pipeline. There are three major operations: updating the INT and flow information in the primary hash table H_P; eviction of a flow entry (including INT meta-data) to the general ring buffer R_G when required; the notification of a flow's INT information to the INT event ring buffer to deliver the information to the host. The eviction to R_G takes the least amount of time. The averaging of the INT metric when writing the entry into the ring R_G takes less time than updating the complete path's information, which includes up to 20 measurements (5 switches and 4 measurements each). On the other hand, the first step of updating the INT and flow information in H_P is to write the latest INT metrics received for each of the INT transit node, which takes additional processing. The latency for notification of the INT event into R_I depends on the event notification threshold and the resulting rate. A higher event notification rate has a significant impact on the latency because of the need to have a lock on R_I. We configure the experiment to have around 0.4 million INT events sent to R_I per second. A lock is necessary to ensure correctness and to avoid a race among the different threads of 52 PMEs that can concurrently access and update the ring buffer. Nonetheless, we can observe that the median latency within the sNIC for the three operations of INT processing and event notification are less than 5μs and a maximum of 25μs which is orders of magnitude lower than processing INT on the host CPU.

E. Accuracy

Fig. 6 shows the histogram of the metrics collected across the INT transit nodes and their accuracy relative to the ground



(a) End-to-end Hop Latency (b) Queue Occupancy at Switch 1 (c) Link Utilization at Switch 1

Fig. 6: Accuracy in collecting INT metric distribution. Loss rate = 1−(#Observed sNIC Occurrences)/(#Occurrences in trace).

truth. Collecting each metric requires extracting the INT metrics for each switch in the path. Our experiments consider three metrics (*i.e.,* end-to-end hop latency, queue occupancy and link utilization for the first switch in the path) with the notification rate configured to be around 0.4 Million INT events/sec. Each metric value is randomly generated according to an exponential distribution. Data for Fig. 6(a) is collected by summing hop latency across all switches, while for Fig. 6(b) and (c), we consider data from the first switch. Fig. 6 shows that the inaccuracy at the host for each of the metrics reported to the host is less than 0.04% at the full line rate. This is significantly better than what has been observed with other efforts reported in the literature. We are currently designing methods to robustly identify the bottleneck link in the network (and in the path of individual flows). Flows with the highest end-end latency can be identified, with the host using exported statistics to determine the switch in the path of those flows with the highest queue occupancy/link utilization.

V. Conclusion

Efficiently collecting INT traffic statistics at an INT sink, in a loss-free manner and generating notifications without impacting throughput is crucial for an INT network monitoring platform. To strike a balance between having more complex INT metric collection and maintaining a high throughput, we proposed a smartNIC-based host as an INT sink and monitoring platform. Unlike using a P4 switch as an INT sink, the sNIC is able to perform INT-packet processing at high rates as well as the complex statistics collection tasks in a loss-free manner. The packet processing pipeline in the sNIC combines the packet header extraction using P4, and callable C functions running on micro-engines to achieve the high performance we desire. While INT traffic is aggregated on a large hash table on the sNIC, the INT events are exported to a set of ring buffers retrieved by the monitoring host. The history of INT events and aggregated INT metrics are stored using in-memory database in the monitoring host. Our evaluations show that the monitoring platform achieves full line-rate throughput through the sNIC, high event notification rates, and good accuracy for traffic statistics collection. The latency incurred by our INT-sink and monitoring platform is quite low–a few μsecs–so the platform can function as a 'bump-in-the-wire'.

Acknowledgement: This work was supported by National Science Foundations awards CNS-1763929 and CNS-1618030.

References

[1] "In-band Network Telemetry (INT)." [Online]. Available: https://p4.org/assets/INT-current-spec.pdf

[2] W. Stallings, *SNMP, SNMP v2, SNMP v3, and RMON 1 and 2 (Third Edition)*. Reading, Mass.: Addison-Wesley, 1999.

[3] "P4 language specification." [Online]. Available: https://p4.org/specs/

[4] Y. Tokusashi, H. T. Dang, F. Pedone, R. Soulé, and N. Zilberman, "The case for in-network computing on demand," in *Proceedings of the Fourteenth EuroSys Conference 2019*, ser. EuroSys '19. New York, NY, USA: Association for Computing Machinery, 2019. [Online]. Available: https://doi.org/10.1145/3302424.3303979

[5] J. Vestin, A. Kassler, D. Bh, K.-j. Grinnemo, J.-O. Andersson, and G. Pongracz, "Programmable event detection for in-band network telemetry," 09 2019.

[6] J. Hyun, N. Tu, J.-H. Yoo, and J. Hong, "Real-time and fine-grained network monitoring using in-band network telemetry," *International Journal of Network Management*, vol. 29, p. e2080, 10 2019.

[7] "In-band network telemetry (int) dataplane specification v2.0." [Online]. Available: hhttps://github.com/p4lang/p4-applications/blob/master/docs/INT_v2_0.pdf

[8] "Netronome NFP-4000 Flow Processor." [Online]. Available: https://www.netronome.com/m/documents/PB_NFP-4000.pdf

[9] N. Van Tu, J. Hyun, G. Y. Kim, J.-H. Yoo, and J. W.-K. Hong, "Intcollector: A high-performance collector for in-band network telemetry," in *2018 14th International Conference on Network and Service Management (CNSM)*. IEEE, 2018, pp. 10–18.

[10] 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. PP, 11 2019.

[11] Y. Kim, D. Suh, and S. Pack, "Selective in-band network telemetry for overhead reduction," 10 2018, pp. 1–3.

[12] J. Marques, M. Caggiani Luizelli, R. Iraja Tavares da Costa Filho, and L. Gaspary, "An optimization-based approach for efficient network monitoring using in-band network telemetry," *Journal of Internet Services and Applications*, vol. 10, 12 2019.

[13] D. Suh, S. Jang, S. Hanb, S. Pack, and X. Wang, "Flexible sampling-based in-band network telemetry in programmable data plane," *ICT Express*, vol. 6, 09 2019.

[14] J. Liang, J. Bi, Y. Zhou, and C. Zhang, "In-band network function telemetry," 08 2018, pp. 42–44.

[15] D. Bh, A. Kassler, J. Vestin, M. A. Khoshkholghi, and J. Taheri, "IntOpt: In-band network telemetry optimization for nfv service chain monitoring," 05 2019, pp. 1–7.

[16] N. Tu, J. Hyun, and J. Hong, "Towards onos-based sdn monitoring using in-band network telemetry," 09 2017, pp. 76–81.

[17] "P4 data plane programming for server-based networking applications." [Online]. Available: https://www.netronome.com/m/documents/WP_P4_Data_Plane_Programming.pdf

[18] "Redis." [Online]. Available: https://redis.io/

[19] "The CAIDA anonymized internet traces," http://www.caida.org/data/passive/passive_dataset.xml.

[20] P. Emmerich, S. Gallenmüller, D. Raumer, F. Wohlfart, and G. Carle, "MoonGen: A Scriptable High-Speed Packet Generator," in *Internet Measurement Conference 2015 (IMC'15)*, Tokyo, Japan, Oct. 2015.