Towards Efficient Selective In-Band Network Telemetry Report using SmartNICs

Ronaldo Canofre, Ariel G. Castro, Arthur F. Lorenzon, Fábio D. Rossi, Marcelo C. Luizelli

Abstract

In-band Network Telemetry (INT) is a promising network monitoring approach that allows broad and fine-grained network visibility. However, when a massive volume of telemetry data is reported to an INT collector, there might overload the whole network infrastructure, while still degrading the performance of packet processing at the INT sink node. As previously reported in the literature, programmable devices – in particular, SmartNICs – have strict constraints in terms of processing and memory. In this work, we propose to design and implement a lightweight Exponentially Weighted Moving Average based mechanism inside the SmartNIC data plane in order to assist the decision-making process of reporting INT data. By evaluating our solution in state-of-the-art SmartNICs, we show that our proposal can decrease the number of nonessential telemetry data sent to INT collectors by up to 16X compared to the de-facto INT approach while presenting minor overhead in terms of packet latency.

1 Introduction

In-band Network Telemetry is a promising near real-time network monitoring approach [12, 17, 22] that enables wide and fine-grained network visibility. In a nutshell, INT consists of instrumenting the collection of low-level network

Ronaldo Canofre, Ariel G. Castro, Arthur F. Lorenzon, Marcelo C. Luizelli
Federal University of Pampa (UNIPAMPA)
Alegrete, Brazil, e-mail: {canofre,ariel.aluno,arthurlorenzon,marceloluizelli}@unipampa.edu.br

Fabio D. Rossi
Federal Institute Farroupilha (IFFAR)
Alegrete, Brazil, e-mail: fabio.rossi@iffarroupilha.edu.br

monitoring statistics directly from the data plane – allowing network operators/monitoring applications to be fed with an unprecedented level of information. Examples of such in-network statistics include data plane metadata (e.g., per-packet processing time, or queue utilization), and/or custom-made ones (e.g., network flow inter-packet gap [24]). Over the last years, INT has been successfully applied to a series of use cases [18], including the identification of short-lived network behaviors [13] and network anomalies [11].

In the classic hop-by-hop INT specification (i.e., INT-MD (eMbed Data)[1]), an INT source node embeds instructions into production network packets typically using either unused header fields (e.g., IPv4 options) or by re-encapsulating the network traffic (e.g., using INT encapsulation). Then, INT transit nodes embed metadata to these packets according to the instructions given by the INT source. Last, an INT sink node strips the instruction out of the packet and sends the accumulated telemetry data to an INT collector. Figure 1 illustrates the whole INT procedure. In this example, a packet from network flow f_1 is used to collect INT data from forwarding devices A to F. Recently, investigations have made the first efforts to efficiently orchestrate how INT metadata are collected by network packets [22, 2, 11, 5] in order to increase network visibility and timely detect network events. That includes, for instance, selecting the appropriate network flows/packet to collect the right network telemetry metadata in the network infrastructure. This problem has been proved to be NP-hard since packets might have different spare capacities (e.g., limited by the MTU data link) [19].

Despite these efforts, little has yet been done to efficiently and wisely report the collected telemetry data to an INT collector [23]. In the case where all the telemetry data is reported to an INT collector, there might lead to *(i) an excessive usage of network links between the INT sink and the INT collector*. For instance, if we consider a 10 Gbit/s network link sending 64-Bytes packets (i.e., 14.88 Mpps) and collecting 1 Byte per INT node transit along the way, the volume of network traffic needed to be reported per second would be 118 Mbit $*$ hops (path length). In fact, this volume of reported data can increase substantially if we assume the canonical reference architecture for programmable devices[2] where each device has at least 30 Bytes of metadata; *(ii) performance degradation on packet processing capabilities at the INT sink* due to the usage of packet cloning/recirculation primitives inside the data plane. To send the network packet to the INT collector (or part of it), programmable devices have to rely on packet recirculation/cloning primitives to duplicate the packet – which dramatically reduces the performance in terms of throughput and latency [27]; and *(iii) the overwhelm of the INT collector application* with INT data packets.

To fill in this gap, in this work we propose a selective INT report mechanism entirely implemented using a SmartNIC. As previously reported by [27], programmable devices have stringent constraints in terms of processing capa-

[1] INT specification: `https://github.com/p4lang/p4-applications/blob/master/docs/INT_v2_1.pdf`

[2] `https://github.com/p4lang/p4c/blob/main/p4include/v1model.p4`

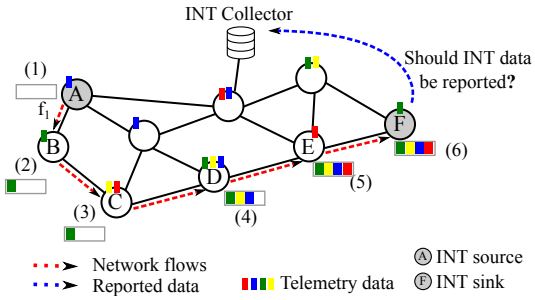


Fig. 1 Overview of the problem.

bilities (e.g., lack of floating-point operations) and memory usage limitations. We assume that the INT sink node is on top of a SmartNIC and, therefore, the decision whether report telemetry data or not is upon the NIC. Our proposed mechanism utilizes a lightweight Exponentially Weighted Moving Average inside the data plane. For that, we implemented it using P4 language and Micro-C routines to allow more complex operations inside the data plane. By performing an extensive performance evaluation using SmartNICs, we show that our proposed approach can reduce the amount of non-important telemetry data sent to INT collector by up to 16X (when compared to the Classical INT), while introducing negligible overhead in terms of packet latency.

The main contributions of this paper can be summarized as:

- an in-network mechanism implemented in state-of-the-art SmartNICs to wisely decide when to report INT metadata;
- a discussion of current limitations on implementing in-network computing in SmartNIC architectures; and
- an open-source code in order to foster reproducibility.

The remainder of this paper is organized as follows. In Section 2, we describe the SmartNIC architecture used in this work. In Section 3, we introduce our proposed approach. In Section 4, we discuss the obtained results. In Section 5, we overview the recent literature regarding in-band network telemetry, and. Last, in Section 6, we conclude this paper with final remarks.

2 Background

Cutting-edge programmable NICs (named SmartNICs) rely their architectures either on (i) multi-threaded, multi-core flow processor units or (ii) on FPGAs (Field Programmable Gate Arrays) to meet the increasing and strict demand. We concentrate our analysis on the general architectural elements of the Netronome SmartNIC architecture [21] – which is used afterward in our performance experiments – and rely on a multi-core architecture.

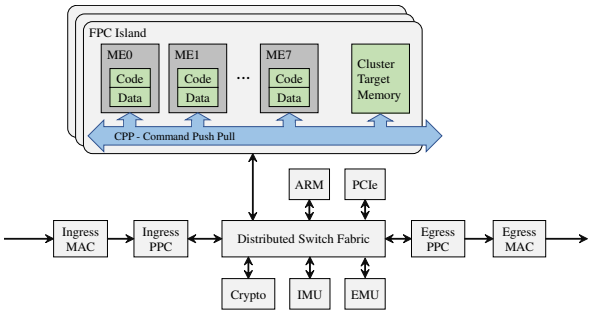


Fig. 2 An overview of the Netronome SmartNIC architecture [27].

The SmartNIC Netronome NFP4000 architecture manages its flow processing cores (FPC) in multiple islands (Figure 2). Each FPC includes eight Micro Engines (MEs) as a particular processor keeping its own instruction store (*code*) and local memory (*data*). Therefore, every ME in the architecture can run code with all other MEs in parallel. To support this feature, each ME holds 8 threads that can be used for cooperative multithreading in the manner that, at any given moment, at most, one thread is executing code from the same program. It means that each FPC handles at most eight parallel threads at 1.2Ghz (one thread per ME). In each FPC, local memory comprises 32-bit registers, shared between all eight threads. Such registers are separated into: (*i*) general-purpose registers (256 32-bits registers) – used by default to store any register of up to 32-bits size; (*ii*) transfer registers (512 32-bits registers) – used for copying register over the interconnection bus (e.g., from or to other FPCs or memories); (*iii*) next-neighbor registers (128 32-bits registers) – used mostly to intercommunicate with adjacent FPCs; and (*iv*) local memory (1024 32-bit registers) – which is a little bit slower than general register. When there is a demand for more memory than available space in local FPC registers, variables are automatically and statically assigned to other in-chip memory hierarchies. Further, there are other sorts of memory available to FPCs: (*i*) Cluster Local Scratch (CLS) (20-50 cycles); (*ii*) Cluster Target Memory (CTM) (50-100 cycles); (*iii*) Internal Memory (IMEM) (120-250 cycles); and (*iv*) External Memory (EMEM) (150-590 cycles). For further details, the interested reader is referred to [21].

As packets are acquired from the network, an FPC thread picks up the packets and processes them. Extra threads are assigned to new packets as they arrive. For example, the SmartNIC NFP-4000 supports up to 60 FPCs, which enables the process of up to 480 packets simultaneously. The SmartNIC allows to program it directly using Micro-C language (i.e., a subset of C language) or using high-level domain-specific languages such as P4 [3]. The code is then compiled and statically assigned to a particular subset of FPC.

3 ETA: Early Network Telemetry Flow Analyzer Approach

In this section, we first define the model used by our proposed approach to decide whether or not network telemetry data is sent to an INT collector. Then, we describe how it is implemented in a SmartNIC and discuss existing limitations and challenges.

3.1 Model and Problem Definition

We consider that a programmable forwarding device $d \in D$ has $N \in \mathbb{N}^+$ available metadata to be collected by an INT-enabled packet p and that such packet has a limited available capacity to carry up to $M \in \mathbb{N}^+$ of such N items ($M \geq N$). For simplicity, we assume that all devices D have the same telemetry information and that an INT-enabled packet p can only collect telemetry data atomically, that is, it either collects all N metadata from $d \in D$ or none of them. Packet p can only collect once the same subset of telemetry data from the device d. We assume that the INT source instructs the packet p correctly according to a given algorithm (e.g., [19]).

Consider that a packet p has collected $M' \subseteq M$ telemetry data along its routing path – which comprises a subset of D devices. When the packet p gets to the INT sink node, the question to be answered is: *should it be sent to the INT collector or not?* To answer this question, the INT sink node computes (i) a weighted average of metadata M' collected by packet p and (ii) an exponential weighted moving average. The former tends to weigh the collected telemetry data differently according to its importance. For instance, the processing time (or the queue utilization) metadata might be more important to be considered in the decision process than the packet size. In turn, the latter tends to keep in memory the observed behavior of the latest received telemetry information over time.

Upon a received packet p in the INT sink node, it extracts the M' collected telemetry metadata and computes a weighted average per packet $A_p = \sum_{i=1}^{M'} w_i \cdot M_i$ (Eq. 1), where $w_i \in [0, 1]$ is the i-*th* weight given to the telemetry data. We further assume that $\sum_{i=1}^{M} w_i = 1$ and that M_i corresponds to the i-*th* collected data. Such individual averages A_p are then summing up into a accumulated weighted average metric within a given time window W (we discuss this design choice next). The time window W is defined for simplicity as a predefined number of packets. However, it can be extended to other metrics such as a time interval.

Then, the accumulated weighted average of a given window W is given by $A_w = \frac{\sum_{p=1}^{W} A_p}{W}$ (Eq. 2). Similarly, the exponential weighted moving average is obtained by $A_e = \alpha \cdot A_w + (1 - \alpha) \cdot A_e$ (Eq. 3) where $\alpha \in [0, 1]$ comprises the importance given by the elements obtained in the last window W versus

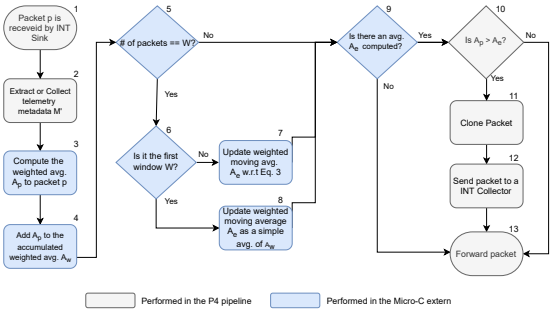


Fig. 3 Overview of the proposed P4+Micro-C pipeline approach.

the historical knowledge maintained by the moving average A_e. Observe that higher values assigned to α prioritize the behavior on the latest window, while lower values prioritize the observed behavior over time. The decision-making process is made in a per-packet manner; however, the decision-making metrics (e.g., A_e) are updated in a time-window manner.

3.2 Design and Implementation in a SmartNIC

Figure 3 illustrates an overview of the proposed approach pipeline implementation. Our approach utilizes the reference V1Model architecture to programmable forwarding devices as the basis to implement/add such functionalities into the Netronome SmartNIC. Also, our approach is based on P4-16 and in Micro-C languages.

Upon a packet is received by the SmartNIC, the packet is parsed accordingly. Our approach implementation resides just after the parsing step and the ingress pipeline (where the routing decision takes place), that is, in the egress pipeline. For this discussion, we assume that our forwarding device can process Ethernet frames and IP packets (or any known INT encapsulation protocol). Therefore, we omit the parsing steps since it is trivial and out of this work's scope. We then focused on the following up steps.

Our approach is implemented in the INT sink. Therefore, at this stage, all INT telemetry metadata has already been collected. The first steps of our approach consist of receiving the packet and extracting the telemetry data from it – or, in some cases, collecting them directly from the data plane (this might happen when the INT sink node acts as an INT transit node as well). This corresponds to steps 1 and 2 in Figure 3. The extracted data is then stored in a custom-made metadata header structure – named `ETA` metadata struct. This structure comprises M 32-bit structures and a 2-bit flag used to instruct the decision-making process. For instance, if a packet needs to be sent to the INT collector, this flag is used internally. In the Netronome architecture, the existing timestamps (`ingress_timestamp` and

`current_timestamp`) are 48-bit words. However, most of the applications use only the least significant bits (the last 32 bits) since they account for nanosecond differences. Further, as discussed next, the Netronome architecture also limits the results of arithmetic operations to 32 bits words (and therefore, we cannot operate on 48-bits timestamps). After extracting telemetry data into the `ETA` struct, our implementation calls a Micro-C extern code to handle more complex operations inside the data plane (depicted in Figure 4). For instance, floating-point operations are not allowed in the P4-16 language reference, nor does the SmartNIC natively implement it. To allow the SmartNIC code to be partially written in P4 and Micro-C – and more importantly, to exchange data between them – we rely on `ETA` structs and internal P4 metadata to enable such real-time communications.

The Micro-C code starts receiving data from the P4 pipeline and locking up memory regions using a mutex (steps 1-3 in Figure 4). Next, we calculate A_p and A_w according to Equation 1 and Equation 2, respectively. Figure 4 illustrates these procedures in steps 4-7. It is important to mention that neither the P4 reference architecture nor the Netronome support floating-point instructions. We implemented all these operations using fixed-point representation with 32 bits to surpass that limitation. In short, a fixed-point representation handles all real numbers as integers. The process scales up the numbers by multiplying by a constant factor C and then performs the multiplications/division as a regular integer. Finally, the scaled-up result is scaled down appropriately. In our implementation, we have used a constant C of 16 bits and performed the scale-up operation by applying a bit-shifting operation (i.e., $M_i' << 16$). After calculating A_p, we verify whether it is the case to send it out to the INT collector. We compare A_p with the observed A_e (which captures the historical behavior). In case the A_p packet value is higher than the dynamic A_e one, we mark the packet to be sent out to the collector (steps 8-12 in Figure 4).

As previously mentioned, our approach performs per-packet decisions. However, we update the exponential weighted moving average A_e at the end of a given window W. This is done because the average A_e depends on A_w – and, the Netronome architecture does not allow arbitrary integer divisions (only by the power of 2 by performing bit-shifting – for instance $M_i' >> 16$). Our approach verifies whether it reaches or not the end of a given window W (step 14 in Figure 4). If so, then it updates the moving average A_e (step 15-19 in Figure 4) according to Equation 3. Further, if it is the first time to reach the window W (i.e., $W' == W$), then A_e assumes a simple average of A_w (step 20 in Figure 4). Then, the Micro-C sets the mutex down and allows other threads to use the blocked shared memory. Last, our Micro-C code returns the calculated values to the original P4 pipeline (step 22-23 in Figure 4).

Back in the P4 pipeline (Figure 3), our approach clone and recirculate the packet p. The original packet is forwarded to its destination (step 13 in Figure 3), while the cloned one is sent to the INT collector (step 12 in Figure 3).

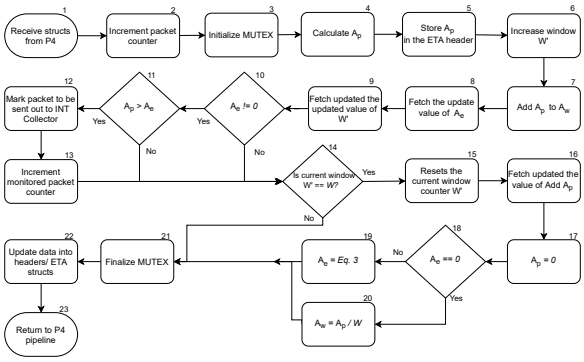


Fig. 4 Overview of the proposed Micro-C routine.

4 Performance Evaluation

In this section, we perform a performance evaluation of our proposal mechanism using a Netronome SmartNIC. We start by describing our environment setup, followed by discussing the results.

4.1 Environment Setup and Baseline Comparison

Setup. Our environment setup consists of three high-end servers. Each server has an Intel Xeon 4214R processor with 32 GB RAM. One server is our Device Under Test (DUT) – i.e., the server in which our solution (i.e., P4 program) is loaded – and the other two are used for traffic generation. All servers have a Netronome SmartNIC Agilio CX 10 Gbit/s network device with two network interfaces, which are physically connected (i.e., each traffic generator is connected to the DUT directly). We use MoonGen[10] as our DPDK[3] traffic generator. We instruct MoonGen using the Netronome Packet Generator[4]. In our experiments, we send 64B IPv4 packets at line rate (i.e., 10Gbit/s) with random source and destination prefixes. One of the traffic generator servers is used to send foreground network traffic, while the second is used to inject abrupt network traffic from time to time. This abrupt network traffic is generated with MoonGen to lead to a congested scenario. The experiment is run through 105 seconds, divided by time slots/epochs of 15 seconds each. We sent network traffic bursts in the following slots: $2nd$ (15s-30s), $4th$ (45s-60s), $6th$ (75s-90s). In our experiments, the SmartNIC Netronome acts simultaneously as an INT transit and an INT sink. First, it collects internal data plane metrics such as packet processing time and packet size (i.e., set M'). Then, we

[3] `https://www.dpdk.org/`

[4] `https://github.com/emmericp/MoonGen/tree/master/examples/netronome-packetgen`

assume that both metrics are weighted equally (i.e., $w_1 = w_2$) for calculating A_p, A_w, and A_e. We varied the window size W from 2^{20} to 2^{23} packets. Also, the parameters α in A_e varies from 0.1 to 0.9. Our solution is compiled using the Netronome's P4 compiler. The compiled code is statically assigned to a single micro engine (ME) inside the SmartNIC. This is done as there are some limitations on the Netronome's Mutex implementation when using different memories hierarchies (i.e., different from the ones inside the ME). All experiments were run at least 30 times to ensure a confidence level higher than 90%.

Baseline. We compare our `ETA` approach against the Full INT procedure and a fixed threshold one. In the former, all INT data are reported to the INT collector, while in the latter we use a constant value as the threshold. This constant value is obtained by calculating the average of all collected INT data in an offline manner. Our codes are publicly available[5] in order to foster reproducibility.

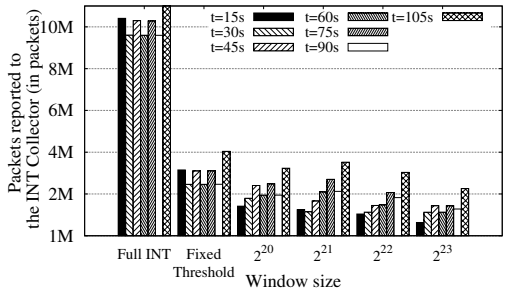


Fig. 5 Number of packets reported to an INT Collector.

4.2 Results

Number of packets sent to the INT Collector. We start by analyzing the number of packets that have been sent to an INT Collector in a given window W (Figure 5). For this experiment, we consider that $\alpha = 0.1$ (we later analyzed its impact). We observe that our approach outperforms the Full INT and Fixed-Threshold procedures in terms of packets reported by a factor of $16X$ and $5X$, respectively. The main reason consists of the dynamic adjustment made in the threshold value over time. Further, we also observe that our approach decreases the number of reported packets as the size of the windows W increases. For instance, we note 43% less network traffic being reported using a windows $W = 2^{23}$ than $W = 2^{20}$. With larger window size,

[5] urlhttps://github.com/canofre/mestrado/

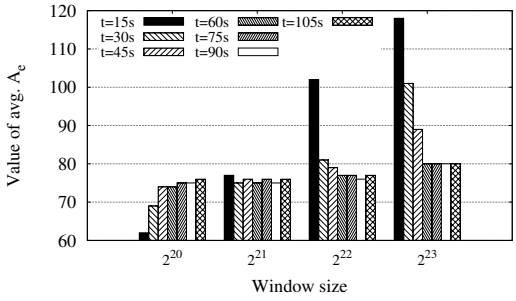


Fig. 6 Impact of window size on the average A_e with $\alpha = 0.2$

our metrics A_w tends to be smoother over time and less impacted by abrupt changes in the data plane metrics. Last, we also note a saw-tooth behavior between bars (e.g., $W = 2^{30}$), which represents the effect of sending packet bursts in a given time interval. The larger is the window size, the less is the saw-tooth behavior observed.

Impact of the window size on the computed average A_e. Next, we evaluate how the average A_e computed by the data plane evolves considering different window sizes (from 2^{20} to 2^{23}). Figure 6 illustrates the behavior when setting $\alpha = 0.2$. When the window size is small (e.g., 2^{20}), the average A_e increases over time until reaching a stable value. On the contrary, larger windows tend to decrease the average A_e value over time until reaching a similar stable value. With larger windows (i.e., 2^{22} and 2^{23}), the summation made before computing A_e within an epoch encompasses the periods where network bursts are sent. Therefore, it ends up increasing its initial value in comparison to small windows. Further, we also note that the value found in the stability tends to be more uniform in this case.

Impact of fine-tuning the parameters on the computed average A_e. We analyzed how the A_e evolves varying α from 0.1 to 0.9. For this experiment, we set the window size to $W = 2^{20}$. Note that higher values assigned to α tend to prioritize the behavior observed in the last time window (i.e., A_w), while lower values prioritize the historical behavior observed over time (i.e., A_e). As we can observe in Figure 7, the higher the α values the higher the average A_e gets over time. In this case, network traffic bursts change data plane metrics (e.g., in-network processing time) and then propagate such values' increases with more intensity to the following-up windows. In turn, lower values of α tend to prioritize the historical behavior and, therefore, the obtained values of A_e are less susceptible to short network traffic variations.

Impact on packet processing latency and throughput. Last, we evaluate the impact of our approach in terms of packet processing and latency when processing packets in the data plane. First, Figure 8(a) depicts the

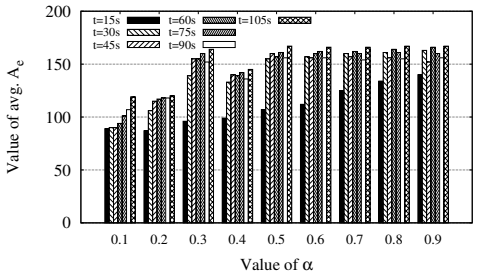
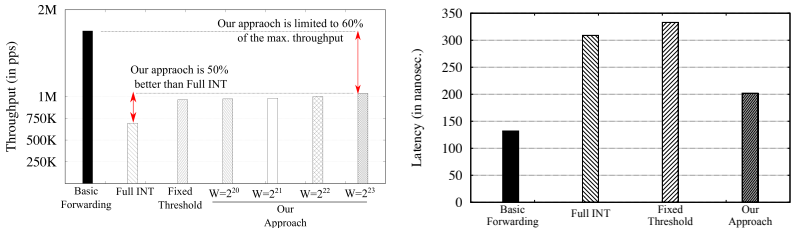


Fig. 7 Impact of the fine-tuning of α on the computed average A_e considering $W = 2^{20}$.



(a) Measured throughput. (b) Measured packet latency.

Fig. 8 Impact of Throughput and Latency of our proposed approach.

achieved throughput in packets per second. Our approach is able to outperform the Full INT strategy and the Fixed Threshold by 50% and by 10%, respectively. However, when compared to the Basic Forward – i.e., the same P4 code without any add-on – our approach is limited to run at most 60% of the maximum throughput. This occurs mostly because our approach demands more processing power and locks (due to mutex usage) and our approach is limited to running in a single ME (with 8 threads). In turn, Figure 8(b) illustrates the incurred data plane latency. We measured the latency as the difference between the ingress and egress timestamps inside the data plane. As we observe, our proposed approach adds around 60ns – that is, 1.53X higher than the Basic Forwarding approach. In turn, the Full INT and the Fixed Threshold approaches add 2.34X and 2.52X, respectively.

5 Related Work

The data plane programmability has opened up a wide range of research opportunities to solve existing network monitoring problems such as packet priorization [7], the usage of selective telemetry to reduce the network overload [14], and the classification and analysis of network flows [4]. As previously mentioned, INT allows the network flow packets to embed network teleme-

try data - such as in [26, 1, 6]. Sel-INT [26] leverages select group tables to selectively insert INT headers into software switches based on its bucket's weight and a certain probability. Similarly, PINT [1] leverages global hash functions and randomly decides to embed INT data in a given packet, while LINT [6] implements a mechanism with adaptive telemetry accuracy, where each node in the network analyzes its impact and decides whether to send the information to the collector.

P4Entropy [9] calculates the entropy of traffic by using the information contained in the packets. The entropy results are forwarded to the controller for storage and future analysis. Lin et al. [16] perform data collection to allow the SDN controller to evaluate the behavior of network traffic to improve data routing. Likewise, [8] proposes a DDoS schema entirely in the data plane. It leverages typical DDoS metrics such as incoming flows and packet symmetry ratio and periodically triggers alarms to external controllers.

With the emergency of hardware technologies such as SmartNICs and domain-specific programming languages such as P4, more applications are being implemented directly in the data plane. The implementation of selective and dynamic monitoring with the use of additional headers [25], the classification of packets into classes with the implementation of machine learning algorithms [29] and the detection of flow events [30] are some examples of programmatic packet header manipulation. NIDS [20] consists of a machine learning-based anomaly detection algorithm for detecting anomalous packets and future error mitigation. SwitchTree [15] performs the implementation of the Random Forest algorithm in the data plane for packet analysis and decision-making while updating the rules at runtime. [28] and [29, 15] present an implementation of machine learning models in the data plane using decision tree, and aim to generate network mapping for intrusion detection services.

Current research efforts in the INT domain (e.g., [1, 6, 26]) have mostly neglected the costs of reporting telemetry data to INT collectors. In fact, existing work have considered either a full report of information or a selective one based on static thresholds. In this work, we aim to fill this gap and offload the decision process of reporting INT data into the SmartNIC data plane in a dynamically manner. We propose a lightweight in-network mechanism based on a window-based moving average with collected INT data as input.

6 Final Remarks

In this paper, we propose a lightweight in-network mechanism to selective report in-band network telemetry to INT collectors. Our approach is based on a window-based moving average and it is tailored to SmartNIC architectures. By evaluating our approach in a state-of-the-art SmartNIC, we showed that our approach can report up to 16X less reporting statistics to INT collectors, while introducing a negligible overhead in terms of latency (1.5X higher than the baseline pipeline). As future work, we intend to explore machine learning

algorithms in the data plane to decide whether to report data or not. Also, we intend to explore other offloading alternatives so that the processing workload can be split up on different SmartNICs or in eBPF/DPDK approaches.

Acknowledgements

This work was partially funded by National Council for Scientific and Technological Development (CNPq) (grant 427814/2018-9), São Paulo Research Foundation (FAPESP) (grants 2018/23092-1, 2020/05115-4, 2020/05183-0), Rio Grande do Sul Research Foundation (FAPERGS) (grants 19/2551-0001266-7, 20/2551-000483-0, 19/2551-0001224-1, 21/2551-0000688-9).

References

1. Ben Basat, R., Ramanathan, S., Li, Y., Antichi, G., Yu, M., Mitzenmacher, M.: Pint: Probabilistic in-band network telemetry. In: Proceedings of ACM SIGCOMM. pp. 662–680 (2020)
2. Bhamare, D., Kassler, A., Vestin, J., Khoshkholghi, M.A., Taheri, J.: Intopt: In-band network telemetry optimization for nfv service chain monitoring. In: ICC 2019 - 2019 IEEE International Conference on Communications (ICC). pp. 1–7 (2019)
3. Bosshart, P., Daly, D., Gibb, G., Izzard, M., McKeown, N., Rexford, J., Schlesinger, C., Talayco, D., Vahdat, A., Varghese, G., Walker, D.: P4: Programming protocol-independent packet processors. ACM SIGCOMM 14 **44**(3), 87–95 (Jul 2014)
4. Castanheira, L., Parizotto, R., Filho, A.E.S.: Flowstalker: Comprehensive traffic flow monitoring on the data plane using P4. In: 2019 IEEE International Conference on Communications, ICC 2019. pp. 1–6. IEEE, Shanghai, China, May 20-24, 2019 (2019)
5. Castro, A.G., Lorenzon, A.F., Rossi, F.D., Da Costa Filho, R.I.T., Ramos, F.M.V., Rothenberg, C.E., Luizelli, M.C.: Near-optimal probing planning for in-band network telemetry. IEEE Communications Letters pp. 1–1 (2021)
6. Chowdhury, S.R., Boutaba, R., François, J.: Lint: Accuracy-adaptive and lightweight in-band network telemetry. In: IFIP/IEEE International Symposium on Integrated Network Management (IM). pp. 349–357 (2021)
7. Cugini, F., Gunning, P., Paolucci, F., Castoldi, P., Lord, A.: P4 in-band telemetry (int) for latency-aware vnf in metro networks. In: Optical Fiber Communication Conference (OFC) 2019. p. M3Z.6. Optical Society of America (2019)
8. Dimolianis, M., Pavlidis, A., Maglaris, V.: A multi-feature ddos detection schema on p4 network hardware. In: 2020 23rd Conference on Innovation in Clouds, Internet and Networks and Workshops (ICIN). pp. 1–6 (2020)
9. Ding, D., Savi, M., Siracusa, D.: Estimating logarithmic and exponential functions to track network traffic entropy in p4. In: NOMS 2020 - 2020 IEEE/IFIP Network Operations and Management Symposium. p. 1–9. IEEE Press (2020)
10. Emmerich, P., Gallenmüller, S., Raumer, D., Wohlfart, F., Carle, G.: Moongen: A scriptable high-speed packet generator. In: Proceedings of the ACM IMC. p. 275–287. IMC '15, ACM, New York, NY, USA (2015)
11. Hohemberger, R., Castro, A.G., Vogt, F.G., Mansilha, R.B., Lorenzon, A.F., Rossi, F.D., Luizelli, M.C.: Orchestrating in-band data plane telemetry with machine learning. IEEE Communications Letters **23**(12), 2247–2251 (2019)
12. Jeyakumar, V., Alizadeh, M., Geng, Y., Kim, C., Mazières, D.: Millions of little minions: Using packets for low latency network programming and visibility. ACM SIGCOMM CCR **44**(4), 3–14 (2014)

13. Joshi, R., Qu, T., Chan, M.C., Leong, B., Loo, B.T.: Burstradar: Practical real-time microburst monitoring for datacenter networks. In: Proceedings of the 9th Asia-Pacific Workshop on Systems. APSys '18, Association for Computing Machinery, New York, NY, USA (2018)
14. Kim, Y., Suh, D., Pack, S.: Selective in-band network telemetry for overhead reduction. In: IEEE International Conference on Cloud Networking (CloudNet). pp. 1–3 (2018)
15. Lee, J.H., Singh, K.: Switchtree: In-network computing and traffic analyses with random forests. Neural Computing and Applications (11 2020)
16. Lin, W.H., Liu, W.X., Chen, G.F., Wu, S., Fu, J.J., Liang, X., Ling, S., Chen, Z.T.: Network telemetry by observing and recording on programmable data plane. In: IFIP Networking Conference (IFIP Networking). pp. 1–6 (2021)
17. Liu, Z., Bi, J., Zhou, Y., Wang, Y., Lin, Y.: Netvision: Towards network telemetry as a service. In: 2018 IEEE 26th International Conference on Network Protocols (ICNP). pp. 247–248 (Sep 2018)
18. Luizelli, M.C., Canofre, R., Lorenzon, A.F., Rossi, F.D., Cordeiro, W., Caicedo, O.M.: In-network neural networks: Challenges and opportunities for innovation. IEEE Network **35**(6), 68–74 (2021). https://doi.org/10.1109/MNET.101.2100098
19. Marques, J.A., Luizelli, M.C., Da Costa, R.I.T., Gaspary, L.P.: An optimization-based approach for efficient network monitoring using in-band network telemetry. Journal of Internet Services and Applications **10**(1), 16 (Jun 2019)
20. Nam, S., Lim, J., Yoo, J.H., Hong, J.W.K.: Network anomaly detection based on in-band network telemetry with rnn. In: IEEE International Conference on Consumer Electronics - Asia (ICCE-Asia). pp. 1–4 (2020)
21. Netronome: Internet (2020), `https://www.netronome.com/static/app/img/products/silicon-solutions/WP_NFP4000_TOO.pdf`
22. Pan, T., Song, E., Bian, Z., Lin, X., Peng, X., Zhang, J., Huang, T., Liu, B., Liu, Y.: Int-path: Towards optimal path planning for in-band network-wide telemetry. In: IEEE INFOCOM. pp. 1–9 (Apr 2019)
23. Saquetti, M., Canofre, R., Lorenzon, A.F., Rossi, F.D., Azambuja, J.R., Cordeiro, W., Luizelli, M.C.: Toward in-network intelligence: Running distributed artificial neural networks in the data plane. IEEE Communications Letters **25**(11), 3551–3555 (2021)
24. Singh, S.K., Rothenberg, C., Luizelli, M.C., Antichi, G., Pongracz, G.: Revisiting heavy-hitters: Don't count packets, compute flow inter-packet metrics in the data plane. In: ACM SIGCOMM Poster. pp. 1–4. ACM, New York, NY, USA (2020)
25. Suh, D., Jang, S., Han, S., Pack, S., Wang, X.: Flexible sampling-based in-band network telemetry in programmable data plane. ICT Express **6**(1), 62–65 (2020)
26. Tang, S., Li, D., Niu, B., Peng, J., Zhu, Z.: Sel-int: A runtime-programmable selective in-band network telemetry system. IEEE transactions on network and service management **17**(2), 708–721 (2019)
27. Viegas, P., Goes de Castro, A., Lorenzon, A.F., Rossi, F.D., Luizelli, M.C.: The actual cost of programmable smartnics: diving into the existing limits. In: Barolli, L., Amato, F., Moscato, F., Enokido, T., Takizawa, M. (eds.) Advanced Information Networking and Applications. pp. 381–392. Springer International Publishing (2021)
28. Xavier, B.M., Guimarães, R.S., Comarela, G., Martinello, M.: Programmable switches for in-networking classification. In: IEEE INFOCOM 2021 - IEEE Conference on Computer Communications. pp. 1–10 (2021)
29. Xiong, Z., Zilberman, N.: Do switches dream of machine learning? toward in-network classification. p. 25–33. HotNets '19, Association for Computing Machinery, New York, NY, USA (2019)
30. Zhou, Y., Sun, C., Liu, H.H., Miao, R., Bai, S., Li, B., Zheng, Z., Zhu, L., Shen, Z., Xi, Y., Zhang, P., Cai, D., Zhang, M., Xu, M.: Flow event telemetry on programmable data plane. In: Proceedings of ACM SIGCOMM. p. 76–89. SIGCOMM '20, Association for Computing Machinery, New York, NY, USA (2020)