Lightweight INT on the Tofino programmable switch

Angelos Dimoglis
a.dimoglis@uva.nl
University of Amsterdam
The Netherlands

Leandro C. de Almeida
leandro.almeida@ifpb.edu.br
Federal Institute of Paraíba
Brazil

Konstantinos Papadopoulos
konpapad@uom.edu.gr
University of Macedonia
Greece

Chrysa Papagianni
c.papagianni@uva.nl
University of Amsterdam
The Netherlands

Panagiotis Papadimitriou
papadimitriou@uom.edu.gr
University of Macedonia
Greece

Paola Grosso
p.grosso@uva.nl
University of Amsterdam
The Netherlands

ABSTRACT

In-band network telemetry (INT), enabled by programmable data planes and the appearance of programming protocol-independent languages such as P4, emerged as a viable approach for network monitoring. INT allows the collection of fine-grained network information in real-time, increasing network visibility, at the cost of network overhead. Several lightweight INT approaches have been recently proposed that attempt to alleviate the transmission overhead of INT, while maintaining a high degree of monitoring accuracy. However, their impact on the resources of the respective hardware network devices has been hardly investigated as most of the approaches are evaluated via simulation. In this study, we provide proof of concept implementations of two lightweight INT approaches that have been proposed for path tracing on the Intel Tofino ASIC, identifying the challenges of porting the solution to the selected target. We examine their performance, providing an in-depth analysis of resource consumption.

CCS CONCEPTS

• **Networks** → **Programmable networks**; **Network monitoring**.

ACM Reference Format:
Angelos Dimoglis, Leandro C. de Almeida, Konstantinos Papadopoulos, Chrysa Papagianni, Panagiotis Papadimitriou, and Paola Grosso. 2024. Lightweight INT on the Tofino programmable switch. In *The 30th Annual International Conference on Mobile Computing and Networking (ACM MobiCom '24), November 18–22, 2024, Washington D.C., DC, USA.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3636534.3696729

ACM MobiCom '24, November 18–22, 2024, Washington D.C., DC, USA

ACM ISBN 979-8-4007-0489-5/24/11.
https://doi.org/10.1145/3636534.3696729

1 INTRODUCTION

In the last few years, networks have been rapidly increasing in scale and becoming more complex than ever before. Current data centers, ISPs, and enterprise networks are operating at traffic rates of hundreds of millions of packets per second per switch. For this reason, today's networks come with high requirements regarding the performance, utilization, availability and security. In order to meet these requirements, network monitoring is more crucial than ever before. Many passive and active monitoring solutions have been introduced to detect security issues, misconfigurations, equipment failure and to perform traffic engineering. The emergence of programmable data planes has led to the introduction of in-band Network Telemetry (INT), a new form of network telemetry providing significant advantages over previous methods, such as flexible programming, in-depth network visibility. This is done by inserting network state information directly into the data packets' headers. In fact, INT can be performed effectively on programmable switches which provide flexible packet forwarding at Terabit speeds. The P4 programming language is the most prominent example in this field, as it allows the use of a common language interface on different kinds of devices (*e.g.,* FPGAs, NPUs, ASICs, etc.) [5, 11, 16].

The P4 INT data plane specification [2] describes different INT modes of operation; at the two extremes telemetry reports can be either directly exported by each INT node, from their data plane to a telemetry collector, or they can be embedded into the packets along the data path along with the INT instructions (INT-MD mode). Nevertheless, the INT framework poses some downsides. The most critical one is the transmission overhead, which in the INT-MD mode increases linearly with the number of hops and telemetry values being carried by packets. In [7] a lightweight form of INT

is proposed to reduce transmission overhead. Specifically, two versions are presented, *i.e.,* a deterministic (DLINT) and a probabilistic (PLINT), which are applied forpath tracing.

DLINT is based on the per-flow aggregation scheme, as telemetry values are spread across the packets of a flow. This approach requires coordination among the switches, which can be accomplished by maintaining per-flow telemetry state on each switch with the use of Bloom Filters. On the other hand, in PLINT the telemetry values are encapsulated into the packets with a certain probability, obviating the need for coordination among the switches. The evaluation results in [7] show that both DLINT and PLINT reduce transmission overhead compared to P4-based INT[2], while achieving high efficiency and accuracy in terms of path tracing. However, the respective experiments were conducted in Mininet using BMv2 software switches. In this study we are porting the DLINT and PLINT approaches to the P4Programmable Tofino Switch by Intel and we examine the performance of both solutions in an environment consisting of both hardware and virtual switches. Specifically, we are (i) looking into the challenges of porting the solution to the selected hardware target, (ii) investigate the impact of the INT approaches on the utilization of the hardware resources.

In Section 2, we provide background information on the two methods. In Section 3 we present the implementation for the Tofino Switch. In Section 4, we examine the utilization of the hardware resources for DLINT and PLINT in a hybrid network consisting of virtual and hardware Tofino Switches. In Section 5, there is a short description of similar state-of-the-art solutions. Finally, in Section 6 we conclude reflecting on the viability of lightweight INT.

2 BACKGROUND

We hereby elaborate on the functionality of the two INT techniques (*i.e.,* DLINT and PLINT) [7]. The main objective of both INT mechanisms is to eliminate telemetry data redundancy within a flow. To this end, DLINT and PLINT exercise per-flow aggregation, *i.e.,* they spread telemetry values across the packets of a flow. For instance, in the context of path tracing, instead of encapsulating all hop IDs within each packet (which would lead to a substantial transmission overhead), the hop IDs are spread among multiple packets, thereby, maintaining a small fixed-size telemetry header. In this respect, DLINT and PLINT follow different approaches, which are briefly explained below (see [7] for further details).

DLINT. DLINT relies on INT node coordination, such that each node is aware which action(s) to perform (*e.g.,* insert a telemetry value) upon each incoming packet. This is achieved through per-flow telemetry states maintained within the INT nodes. More precisely, DLINT utilizes the following three telemetry states (Fig. 1): (i) *Awaiting Init*, where the INT node waits for an INIT signal in order to insert telemetry data, (ii) *Ready to Insert ID*, where the INT node is ready to insert its data into the following incoming packet of the respective flow, and (iii) *Inserted ID*, where the INT node has already inserted its own telemetry data and waits for a signal to revert to the initial state (*Awaiting Init*) in order to re-insert subsequent telemetry value(s). Practically, the third state facilitates the reset of an INT node (telemetry-wise), enabling an uninterrupted insertion of telemetry data over the entire duration of each flow. To reduce the amount of telemetry state that needs to be maintained within each INT node, DLINT utilizes Bloom Filters.

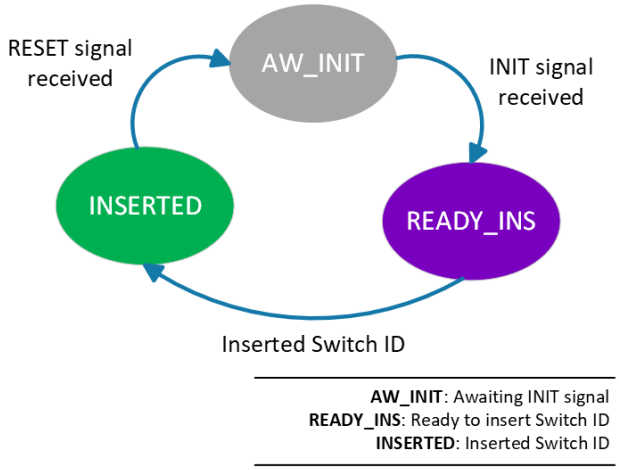

Figure 1: DLINT: Finite State Machine on the switches

PLINT. Unlike the deterministic nature of DLINT, PLINT employs a probabilistic approach, obviating the need for any coordination among the INT nodes. More specifically, the main idea behind PLINT is that each INT node can independently insert telemetry data to each packet with a certain probability. An entirely random insertion of telemetry values would benefit INT nodes closer to the destination, since their data would be less likely to be replaced by other INT nodes. PLINT addresses this issue by leveraging on reservoir sampling, which guarantees an equal probability for the encapsulation of telemetry indicators among all INT nodes. Since reservoir sampling requires the knowledge of hop-count by each INT node, PLINT stores in its telemetry header the TTL value of each packet at its entrance into the INT domain (*i.e.,* the TTL seen by the INT source).

3 LIGHTWEIGHT INT ON TOFINO

In this section we describe the implementation of DLINT and PLINT on the Tofino Switch. We are offering a detailed insight into the control flow of both algorithms and the structure of the used telemetry headers in each case. Finally,

we list the challenges that we came across while trying to port the code to the Tofino Switch. Furthermore, we describe how we dealt with them and what modifications were needed to make the code fully compatible.

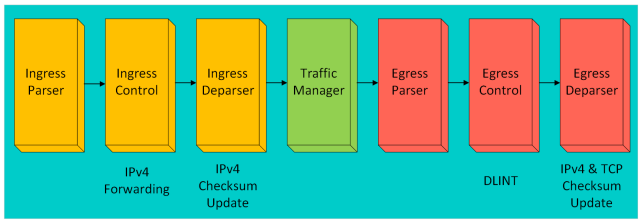

Figure 2: Pipeline in TNA (Tofino Native Architecture)

Porting DLINT: To port the code we had to modify the original implementation to be compliant with the TNA architecture depicted in Figure 2. The Ingress Control implements the standard IPv4 forwarding functionality. The Egress Control contains the INT logic. The DLINT algorithm is executed only for TCP flows. A schematic of the DLINT implementation at the egress pipeline and its interaction with the switch's control plane is presented in Figure 3. Assuming the packet's type is TCP, the present switch is being identified. This means that the switch's ID is being retrieved from the control plane using the respective Match-Action Table (MAT). In the following, the destination of the packet is being checked, based on the egress port. A different block of code will be executed depending on the destination indicating a *RESET signal* or moving between the states *READY_INS* and *INSERTED* depicted in Figure 1.

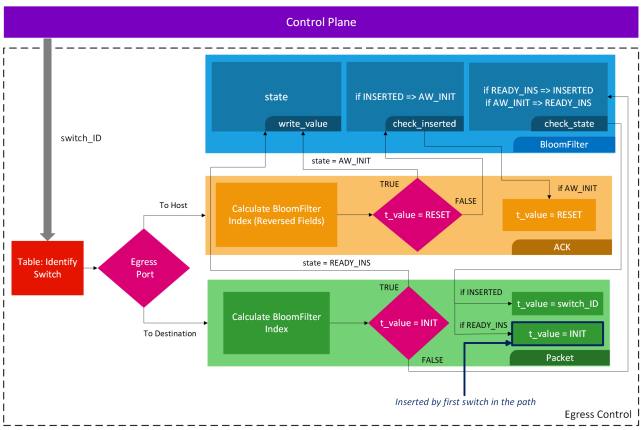

Figure 3: Overview of the DLINT implementation on TNA

Before going forward with the control flow, let us review how the coordination states are stored inside the switches. Similarly to the previous implementation, we are using a probabilistic data structure called Bloom Filter. BF enables the compression of an arbitrary data set into a bit vector and provide membership lookup using hash functions. To implement the data structure we employ a multi-cell register. Each cell represents a specific TCP flow. Thus each cell stores a 2-bit number, which can be 0, 1 or 2. These numbers represent the states AW_INIT, READY_INS and INSERTED respectively. This register is being indexed by a hash function based on the CRC32 algorithm. In order to have a distinct cell for each flow, the function's arguments include the IP Addresses and the TCP ports of both ends (host and server). We are also using the number of the protocol as an argument (IPv4 in our case), so that there are no collisions. So, once the destination of the incoming packet has been determined, the index of the register is being calculated.

TCP traffic forwarded from the host (source) to the destination, is considered valid to carry telemetry data, such as signals for the switches or switch IDs. In that case, if the packet is already carrying an INIT signal, the state of the switch is being updated from AW_INIT to READY_INS using a register action. If the packet is not carrying any telemetry signal, then there are two possible scenarios depending on the state of the switch.

Case 1: If the switch is at AW_INIT, the state is being changed to READY_ INS. This means that the specific switch is the first in the path and it should insert the INIT signal into the packet. Now that the packet is carrying the INIT signal, it will be able to change the state of the remaining switches as described in the previous case.

Case 2: If the switch's state is READY_INS, it is time to insert its telemetry values into the packet (the switch's ID). Thus, the state is being changed to INSERTED and the packet is now carrying the switchID of that switch. This packet is being merely forwarded by the rest of the switches, as every packet is supposed to carry only one SwitchID (per-flow aggregation). The next packet will carry the switchID of the next switch, etc.

Once all of the switches have transitioned to the INSERTED state, it means that all the IDs of the switches have been collected by the telemetry server and the path has been fully reconstructed. When the TCP connection is severed, a [RST,ACK] packet is being sent by the server/receiver. We are using [RST,ACK] packets to carry the RESET signal instead of any [ACK] packet. This way we make sure that enough packets have been sent to collect the IDs of all the switches in the path. This packet is being recognised by the first switch in the opposite direction (last switch in the recorded path) and the corresponding block of code is executed (see block labeled as "ACK" in Figure 3). the switch's state is being changed to its initial state (AW_INIT) and the packet is now carrying the RESET signal. This signal will be recognised by

the remaining switches in the path and their state will be modified to AW_INIT.

This last step concludes the control flow of our implementation on Tofino 2. In the below picture (Figure 4) there is the structure of the telemetry header that is being used. The telemetry data are embedded to the packet as TCP Options, so the telemetry header should include some additional fields. The first field is the kind of the TCP option, which can be any number that is not being used by any other standard protocol (e.g. 72). The next field is the length of the TCP option, that is 4 bytes. The last field is the payload carrying the telemetry data. It was necessary to expand this field to 16 bits (instead of 8 from the previous implementation), because the checksum calculation requires 16-bit alignment. Besides, a 16-bit field seems more realistic for storing the ID of a switch.

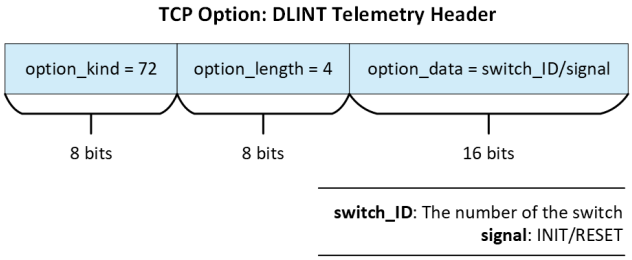

Figure 4: Structure of the telemetry header for DLINT

Porting PLINT: Similarly to DLINT, the IPv4 forwarding is being performed in the Ingress Pipeline and the main program of PLINT is in Egress. In this approach, there is no need for coordination among the switches, thus we are not using a register to store states. The switches are inserting their IDs based on the reservoir sampling. This means that the first switch insert its ID with a probability of 1 (always), the second one with a probability of 1/2, the third one with a probability of 1/3, etc.

But because there is the possibility of having non-P4 switches in the path, the used probability is 1/hopNumber, where hopNum = init_TTL - ipv4_ttl. The init_TTL refers to the TTL since the packet entered the telemetry session (encountered the first P4 Switch). So, there is a probability of 1/hopNum for every switch to replace the switch ID inside the packet with its own. Once the packet has entered the Egress Control, the switch is being identified. Like in the previous approach, this means that the ID of the switch is being retrieved through a MAT (see Figure 5). Now if the packet is not carrying any telemetry data, this indicates that the packet is at the first switch along the data path of the flow. So, telemetry data is stored into the packet, including the ID of the first switch. This happens every time a packet enters the first switch in the path.

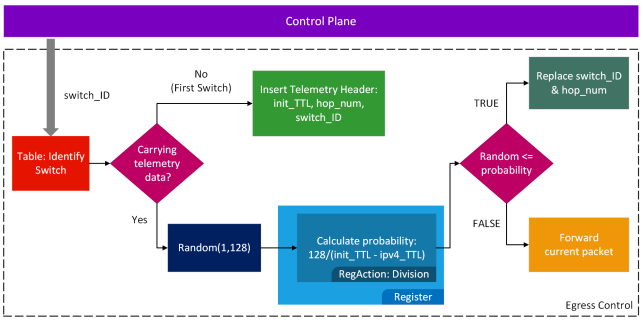

Figure 5: Overview of the PLINT implementation on TNA

On the other hand, if the packet is already carrying a telemetry header, another sequence of actions is being followed. First, a random number from 1 to 128 is being calculated. This is going to be compared to our probability. The probability is the maximum possible random number (128) divided by the number of hops (init_TTL - ipv4_TTL). The division is taking place inside a Register Action. If the calculated random number is less than or equal to the result of the division, the switch ID inside the telemetry header is being replaced by the ID of the present switch. Otherwise, the packet is being forwarded without any modification of the telemetry values. The structure of the used telemetry header is depicted in Figure 6. We are using TCP Option (84) to distinguish it from the telemetry header of DLINT. Here the length is 8 bytes (not 6 as expected), because of the padding field.

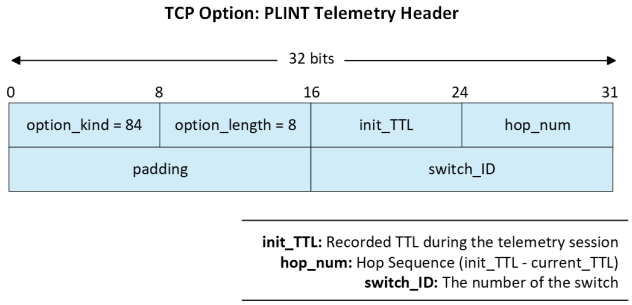

Figure 6: Structure of the telemetry header for PLINT

Challenges: In our effort to port the approaches on the switch we encountered several challenges. This stems from the different data types and limited hardware resources. In both cases we had to use different workarounds to make some functions possible, such as multiple operations on a single register and comparisons.

The primary challenge was in porting DLINT, where many conditions require read/write operations on the Bloom Filter

for a single packet. Considering the restriction of being able to execute only one Register Action per packet, one possible solution is the recirculation of a packet before calling the next Register Action. However, this approach is impractical; it introduces significant delay, as the packet would need to pass through the pipeline multiple times. Taking that into account, the best course of action was to fully take advantage of the SALU capabilities inside the Register Action. The reconstruction of the algorithm was inevitable. Specifically, considering that only two comparisons are allowed inside an action, many of them had to be computed in advance. In this way, we were able to determine the value that should be written in the register. In addition, more register actions were needed, depending on the case. Specifically, we are using 3 different Register Actions in our implementation (see Figure 3). In PLINT there were not any major challenges. The division operation is supported by the ALU on Tofino 2, as mentioned in subsection 3, making the calculation of the probability possible.

4 PERFORMANCE EVALUATION

Evaluation Environment: The evaluation of the DLINT and PLINT approaches is conducted in a testbed that consists of software and hardware switches and two communicating hosts. The software switches are Tofino 2 Models running on the remote machines, where the hosts are located, while the HW target is a P4-programmable EdgeCore AS9516-32D switch (TNA 2). The traffic is generated by the *iperf* tool. The overview of this topology is shown in Figure 7.

The Host initiates a TCP connection to the Destination/Server and sends a flow of packets. Each switch is inserting its ID based on the running program (DLINT/PLINT). At the last hop (switch 5), the telemetry header is being extracted and sent to the telemetry server. This is where the recorded path is being reconstructed.

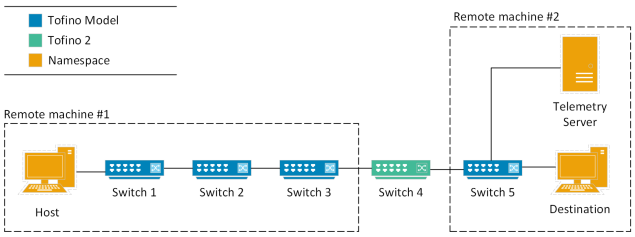

Figure 7: Topology of the evaluation environment

Evaluation Metrics & Measurements: To measure the latency introduced by the INT logic, we move the DLINT and PLINT implementation to the Ingress Control, as we can use the recorded timestamps which can be retrieved from the packet's standard metadata.Specifically, we employ the Ingress global_tstamp that refers to the time that the packet arrived at the Ingress Control. The latency computation is taking place at the Egress Control, where the Egress global_tstamp is available; the Egress global_tstamp refers to the time when the packet arrived at the Egress Control. To estimate the actual time for the Ingress Processing, we subtract from the Egress global_tstamp the Ingress global_tstamp, as well as the time that the packet has spent in the queue of the traffic manager that lies between the Ingress Deparser and the Egress Parser (see Fig. 2) So, we use the time between the packet's enqueue and dequeue, which can be retrieved by the deq_timedelta timestamp. Thus, the delay is calculated as follows and the value is added to the packet's telemetry header $Delay = global\_tstamp_{Eg} - deq\_delta - global\_tstamp_{In}$.

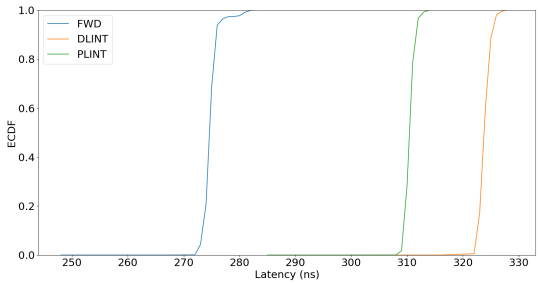

Figure 8: Delay (single)

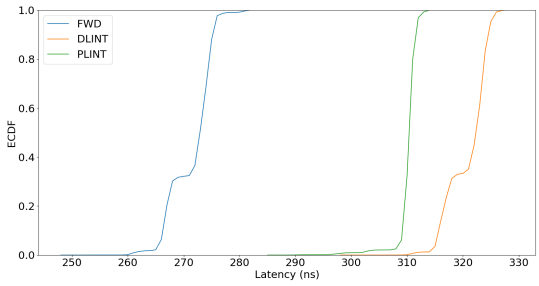

Figure 9: Delay (mix)

To evaluate the impact of the DLINT/PLINT approaches, we compare the respective processing delay against the delay introduced by the IPV4 forwarding functionality (FWD). The respective delay CDFs for the duration of the experiment are presented in Figures 4 and 9, for different basic cases of TCP flows (starting with a single TCP flow at 5 Mbps and a set of 5/10/20 Mbps flows). The selection of the TCP flow rates is constrained by the use of the Tofino models in the hybrid experimentation environment.

L3 forwarding is the baseline subtracted from the PLINT and DLINT measurements to retrieve the impact of the INT approaches on processing delay. PLINT's latency is lower than DLINT's, as it requires less processing and use of stateful resources compared to DLINT. For example, in PLINT no network state is maintained on the switches, so no registers are required. The only process that is taking place is the calculation of the probability and the insertion of the telemetry header. PLINT and DLINT increase processing delay on average by 12% and 17%, respectively, compared to the baseline.

Resource	DLINT	PLINT	FWD
Stages	7	7	2
SRAM	0.4%	0.3%	0.1%
TCAM	0.4%	0.4%	0.4%

Table 1: Resource Utilization

Table 1 presents the memory utilization (TCAM and SRAM) and number of stages. The results are obtained using the P4 Insight (*p4i*) tool[1] provided by Intel to inspect P4 code. The use of TCAM is similar between the two methods and FWD as it is being used to store only the keys for match-action tables. On the other hand, there is a significant difference on the utilized SRAM employed by registers, so DLINT's requirements are higher due to the use of the Bloom Filter for keeping telemetry state.

Power type	Ingress Pipeline		
	DLINT	PLINT	FWD
Weight	171.5	150.8	36.2
Worst-case Power (W)	1.25	1.12	0.35

Table 2: Power Consumption

With regards to power consumption, in Table 2, we present for the PLINT, DLINT and FWD implemented in the ingress pipeline the following metrics:
Weight: a unit-less metric provided by P4 Insight, that represents relative resource usage in each block of the Ingress pipeline. Higher weights correlate with greater resource demand, which indirectly reflects higher power consumption.
Worst-case Power: providing the power consumption (in Watts) in the worst case scenario. These results can be collected from the `mau.power.log` file, which is generated by the P4 Compiler of Intel's SDE.

As shown in Table 2, based on the reported weights as indicators of power consumption, compared to the baseline is approximately 4 and 5 times more for DLINT and PLINT. Based on the same metric DLINT is approximately 12% more power-consuming than PLINT, as there is a difference of 20,7 in the weight values between DLINT and PLINT. However, our findings on the worst case power consumption, indicate that both our implementations of PLINT and DLINT may be up to 3.5 times more power hungry than the basic forwarding. We assume that the power-efficient design of the switch can mitigate the impact of additional resource usage, leading to lower than expected increase in power consumption.

[1]https://www.intel.com.br/content/www/br/pt/products/details/network-io/intelligent-fabric-processors/p4-insight.html.

5 RELATED WORK

The trade-off between monitoring accuracy and INT overhead has been investigated for INT solutions. Most of the proposed methods can be categorized as either deterministic or probabilistic. Regarding the former, several solutions adopt a sampling rate for inserting INT fields in packets of a flow, as means to reduce overhead [4] [12] [13] [10] where the rate can be adjusted at runtime either by the control plane (e.g., [13]) or the data plane (e.g., [10]). FINT [14], based on a triple bitmap, enables setting telemetry tasks and parameters (*i.e.,* the combination of telemetry metadata types, the INT period, etc.) dynamically at runtime, reducing INT impact on network performance. Alternative propositions employ network topology partitioning [8] or clustering [6] to ensure full coverage for the network, scalable telemetry and freshness/timeliness of telemetry information. The DLINT approach employs telemetry states, maintained in the switches, to support inter-switch coordination for the spreading of telemetry data across the packets of a flow in order to reduce the network overhead. Similar to PLINT, there are a number of INT approaches that employ probabilistic logic for inserting telemetry into the data packets[1] [9] [3] avoiding the hassle of coordination among switches or via the controller. Solutions that combine in band and sketch-based approaches, either deterministic (e.g., [15]) or probabilistic (e.g.,[17]) are complementary approaches that use sketch-based compact data structures to achieve low network overhead sacrificing accuracy. All approaches excluding PRoML-INT [12], evaluate the performance using simulations without investigating the impact of the INT approach on the resources and processing of the hardware network devices.

6 CONCLUSIONS

Following the advances in programmable data planes, INT emerged as a monitoring framework that allows the collection of fine-grained telemetry information from the data plane at line rate. In this paper, we have investigated how two lightweight INT approaches, DLINT and PLINT, could be implemented on the programmable switch and convey (i) the

corresponding challenges for programming such solutions on the Tofino 2 ASIC and (ii) their impact on the utilization of its constrained resources. Our evaluations show that on average, PLINT and DLINT increase processing delay by 12% and 17%, respectively, compared to the baseline L3 forwarding. Their impact in overall memory utilization is minor, while the increase in the number of stages used and power consumption is approximately by a factor of 3.5 for PLINT and DLINT. This implies that the implementation could be potentially further fined-tuned in the HW target.

ACKNOWLEDGMENTS

This work has been partially funded by the Dutch Research Council NWO project CATRIN (NWA.1215.18.003) and the European Commission Horizon Europe SNS JU project DESIRE6G (101096466).

REFERENCES

[1] Ran Ben Basat, Sivaramakrishnan Ramanathan, Yuliang Li, Gianni Antichi, Minian Yu, and Michael Mitzenmacher. 2020. PINT: Probabilistic in-band network telemetry. In *Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication*. 662–680.

[2] The P4.org Applications Working Group. [n. d.]. *In-band network telemetry dataplane specification v2.1*. https://p4.org/p4-spec/docs/INT_v2_1.pdf

[3] Abdulkadir Karaagac, Eli De Poorter, and Jeroen Hoebeke. 2020. In-Band Network Telemetry in Industrial Wireless Sensor Networks. *IEEE Transactions on Network and Service Management* 17, 1 (2020), 517–531. https://doi.org/10.1109/TNSM.2019.2949509

[4] Youngho Kim, Dongeun Suh, and Sangheon Pack. 2018. Selective in-band network telemetry for overhead reduction. In *2018 IEEE 7th International Conference on Cloud Networking (CloudNet)*. IEEE, 1–3.

[5] Oliver Michel. 2019. *Packet-Level Network Telemetry and Analytics*. Ph. D. Dissertation. University of Colorado at Boulder.

[6] Dandan Mo, Zhiruo Liu, Du Chen, and Deyun Gao. 2021. C-INT: An Efficient Cluster Based In-Band Network Telemetry. In *2021 4th International Conference on Hot Information-Centric Networking (HotICN)*. IEEE, 129–134.

[7] Konstantinos Papadopoulos, Panagiotis Papadimitriou, and Chrysa Papagianni. 2023. Deterministic and Probabilistic P4-Enabled Lightweight In-Band Network Telemetry. *IEEE Transactions on Network and Service Management* (2023).

[8] Goksel Simsek, Doğanalp Ergenç, and Ertan Onur. 2021. Efficient network monitoring via in-band telemetry. In *2021 17th International Conference on the Design of Reliable Communication Networks (DRCN)*. IEEE, 1–6.

[9] Enge Song, Tian Pan, Chenhao Jia, Wendi Cao, Jiao Zhang, Tao Huang, and Yunjie Liu. 2021. INT-label: Lightweight In-band Network-Wide Telemetry via Interval-based Distributed Labelling. In *IEEE INFOCOM 2021 - IEEE Conference on Computer Communications*. 1–10.

[10] Dongeun Suh, Seokwon Jang, Sol Han, Sangheon Pack, and Xiaofei Wang. 2020. Flexible sampling-based in-band network telemetry in programmable data plane. *ICT Express* 6, 1 (2020), 62–65.

[11] Lizhuang Tan, Wei Su, Wei Zhang, Jianhui Lv, Zhenyi Zhang, Jingying Miao, Xiaoxi Liu, and Na Li. 2021. In-band network telemetry: A survey. *Computer Networks* 186 (2021), 107763.

[12] Shaofei Tang, Jiawei Kong, Bin Niu, and Zuqing Zhu. 2020. Programmable Multilayer INT: An Enabler for AI-Assisted Network Automation. *IEEE Communications Magazine* 58, 1 (2020), 26–32.

[13] Shaofei Tang, Deyun Li, Bin Niu, Jianquan Peng, and Zuqing Zhu. 2019. Sel-INT: A runtime-programmable selective in-band network telemetry system. *IEEE transactions on network and service management* 17, 2 (2019), 708–721.

[14] Shengxu Xie, Guyu Hu, Changyou Xing, Jiachen Zu, and Yaqun Liu. 2022. FINT: Flexible In-band Network Telemetry method for data center network. *Computer Networks* 216 (2022), 109232. https://doi.org/10.1016/j.comnet.2022.109232

[15] Kaicheng Yang, Yuanpeng Li, Zirui Liu, Tong Yang, Yu Zhou, Jintao He, Tong Zhao, Zhengyi Jia, Yongqiang Yang, et al. 2021. SketchINT: Empowering INT with TowerSketch for per-flow per-switch measurement. In *2021 IEEE 29th International Conference on Network Protocols (ICNP)*. IEEE, 1–12.

[16] Minlan Yu. 2019. Network telemetry: towards a top-down approach. *ACM SIGCOMM Computer Communication Review* 49, 1 (2019), 11–17.

[17] Yikai Zhao, Kaicheng Yang, Zirui Liu, Tong Yang, et al. 2021. LightGuardian: A Full-Visibility, Lightweight, In-band Telemetry System Using Sketchlets.. In *NSDI*. 991–1010.