A Comprehensive P4-based Monitoring Framework for L4S leveraging In-band Network Telemetry

Huu Nghia Nguyen, Bertrand Mathieu, Marius Letourneau, Guillaume Doyen, Stéphane Tuffin, Edgardo Montes de Oca

A Comprehensive P4-based Monitoring Framework for L4S leveraging In-band Network Telemetry

Huu Nghia Nguyen[†], Bertrand Mathieu[‡], Marius Letourneau[*], Guillaume Doyen[§],
Stéphane Tuffin[‡], Edgardo Montes de Oca[†]
[†]Montimage, Paris, France, {huunghia.nguyen,edgardo.montesdeoca}@montimage.com,
[‡]Orange Innovation, Lannion, France, {bertrand2.mathieu,stephane.tuffin}@orange.com
[*]LIST3N, University of Technology of Troyes, Troyes, France, marius.letourneau@utt.fr,
[§]OCIF - IRISA (UMR CNRS 6074), IMT Atlantique, Rennes, France, guillaume.doyen@imt-atlantique.fr

***Abstract*—The Low-Latency Low-Loss Scalable throughput (L4S) architecture has recently been proposed to reduce the network latency of low-latency services and to allow their flows to coexist with classic ones in the same domain. This coexistence implies monitoring and security challenges. However current monitoring methods, primarily based-on sampling and polling, exhibit performance and granularity limitations. This paper describes the challenges for monitoring LL services and details our solution when introducing a fine-grained and real-time monitoring capability in our P4-based L4S implementation using In-band Network Telemetry. The initial experimental evaluation shows that our solution is able to monitor the metrics of an L4S switch with very few networking and processing overhead and without disturbing the L4S behaviour.**

***Index Terms*—P4, INT, L4S, High-precision Monitoring**

I. Introduction

Today, many new applications require low or near-zero latency, such as virtual reality, cloud gaming, tele-robotics or tactile internet. End-to-end latency is affected by several factors with packet sojourn time in routers or switches being among well-known factors [1] influenced by the traffic load. In this context, the L4S architecture [2] has recently been proposed to reduce the queuing delay of low-latency (LL) traffic in a way that does not harm classic (CL) traffic. While some LL and CL traffic coexistence issues such as throughput sharing, are addressed in L4S specifications, deploying L4S still implies assurance and security challenges that are calling for a monitoring framework. For instance, three categories of threats to make LL applications unusable have been identified in [3] .

Monitoring L4S implies being capable of observing the internal states of routers and switches to further evaluate, detect, and eventually protect forwarding planes' behaviour. The current methods to obtain reports about internal states are polling or sampling based. Polling causes performance limitation due to the induced polling delay and processing overhead for the L4S node. Sampling may miss small flows and network events such as spikes or anomalies when they occur between two samples.

Meanwhile, the P4 programming language [4] has emerged. It aims at programming packet forwarding in network devices. It is a disruptive instrument [5] as it helps to reduce the need of dedicated hardware devices by introducing programmable devices and APIs to process and control network traffic. We implemented an L4S switch using P4 language in [6]. In this implementation, we modified the code of the BMv2 switch, which is a virtual switch supporting P4, to use its measurement API and wrote the results into its execution log stream. This offers some preliminary elements for debugging and troubleshooting L4S but without the attributes of a fully operational monitoring solution.

This paper describes the challenges in monitoring LL applications and our solution to implement a fine-grained and real-time monitoring system[1] for L4S using P4 language. The monitoring system is based on the In-band Network Telemetry (INT) technology [7] which is a remote network monitoring framework. We overcome the trade-off between precision of monitoring and monitoring overhead thanks to on-demand monitoring which gives the ability to select the metrics to be captured and to set the condition for reporting these metrics. To the best of our knowledge, this is the first dedicated INT monitoring framework for L4S networking devices using P4.

The rest of the paper is organised as follows. Section II presents the background about L4S and INT. We detail the challenges and our approach to tackle them in Section III. We evaluate our implementation in Section IV. Section V presents related work. Section VI concludes the paper.

II. Background

A. L4S Architecture

A key concept of L4S is the Dual queue coupled Active Queue Management (AQM) with one queue for LL and another queue for CL traffic. The coupling between the two queues allows these two traffic types to fairly share bandwidth when they coexist in the same bottleneck which is a prerequisite to LL deployment over the Internet. Consequently, LL traffic leveraging TCP-Prague congestion control can coexist with throughput oriented traffic using more common congestion controls such as Reno/Cubic without the former starving the latter [8].

The Dual queue AQM consists of three main components. The first one classifies the ingress LL and CL packets to

[1]The code and test results are open-source at https://github.com/mosaico-anr/p4-int-l4s

determine which queue the packet should be forwarded to. The classification is usually based on a 2-bit Explicit Congestion Notification (ECN) field in the IP header however other classification methods may be applied to steer non-L4S low-latency flows in the LL queue. Once classified a packet is processed by the AQM of the CL or LL queue with the marking or dropping probability of the CL queue being proportional to the square of the marking probability of the LL queue. This coupling between the two queues forms the second component of the Dual queue AQM. It ensures the fair sharing of bandwidth between the two types of traffic. It represents a core feature of the L4S architecture as the coexistence and fairness of the two types of traffic are strong prerequisites in the design of L4S. The last component is a scheduler that gives conditional priority to the LL traffic over the CL traffic. The conditional priority being meant to avoid short-term starvation of the CL traffic by the LL traffic.

B. In-band Network Telemetry

Network telemetry has emerged as a mainstream technical term [9] to refer to an automated process for remotely collecting and processing network information. In-band network telemetry uses INT packets in the data plane to carry INT metadata that are telemetry instructions and collected information. The P4 Working Group [2] recently defined the INT data plane specification, including the INT system, the INT metadata and the INT report [7]. An INT system basically consists of (i) INT-capable devices, which can be eventually configured by an SDN controller, and (ii) a collector, which receives and extracts information from INT reports that are sent by the devices.

An INT-capable device can play one or more roles: source, transit or sink. Based on the received configuration from the control plane, a source device selects INT packets, then embeds a telemetry instruction bit map into the packets to indicate the network information to be measured. While matching and forwarding an INT packet, a transit device interprets the instructions to collect the required information. The sink device removes INT metadata from the packet. A node will not attach its information into a packet if the resulting packet size is greater than its Maximum Transmission Unit (MTU).

The P4 Working Group defines three INT modes of operation. In the INT-XD (eXport Data) mode, an INT-capable device plays all the three roles, i.e., it selects INT packets, collects then sends telemetry data to the collector. In the INT-MX (eMbed instructXions) and INT-MD (eMbed Data) modes, the source and the sink roles are played by two different devices. The device also plays the transit role to collect data. The difference is that the collected information is directly exported by the node to the collector in INT-MX mode while the information is embedded into packets and forwarded to the next node in INT-MD mode.

[2]https://p4.org

III. Challenges and Design Choices

A. Lack of INT Monitored Metrics dedicated to L4S

The P4 Working Group specifies 8 sets of metrics which are for monitoring general networking devices. Additional metrics are needed to monitor L4S devices: the numbers of dropped packets and marked packets for each LL and CL queues. Although in the initial 8 sets some metrics do not concern L4S, such as the 7^{th} set meant to collect level 2 ingress and egress port IDs, we implemented all of them, to be compatible with the P4 Working Group specification, which is implemented and used on a variety of devices.

It is worth noting that we identified the smallest set of metrics that represent the internal information of an L4S device to save space, hence reducing the bandwidth consumed by INT to transfer the metric values. We do not take into account the metrics that can be collected outside the device by analysing the packets before or after it, e.g., bandwidth, throughput or packet inter-arrival time. Our framework monitors the following metrics:

1) *Device ID* is the unique identification of the device.
2) *Level 1 Ingress and Egress port IDs* are the IDs of the ports on which the packet was received and sent.
3) *Hop latency* is the time, in microsecond, that it takes for the packet to be processed within the device.
4) *Queue ID and queue occupancy* are the ID of a queue and its build-up of traffic in the queue, expressed as the number of packets, at the moment the packet was sent out. We have two queues in a L4S devices: queue 0 for LL and queue 1 for CL traffic.
5) *Ingress timestamp* is the device's local time, in nanosecond, when the packet was received.
6) *Egress timestamp* is the device's local time, in nanosecond, when the packet was sent out.
7) *Level 2 ingress and egress port IDs* are the IDs of the logic ports, applying for layer 3 switched virtual interface, on which the packet was received or sent.
8) *Egress port TX link utilisation* is the current usage of the egress port the packet was sent through.
9) *Numbers of marked packets and dropped packets* are the amount of packets that have been marked and dropped by L4S, respectively. Only the packets being dropped due to the behaviour of L4S are counted. By analysing the ingress and egress traffic, we can obtain the total dropped packets including the ones dropped by the device when it does not find a route to send the packets.

As such, the above metrics can be used to identify, for example, the dominant contributing IP sender whose packets occupy most of a queue during a given interval. Indeed, by using the ingress and egress timestamps, one can get the set of packets that were present in the queue during the given interval. Furthermore, since the INT collector can also extract the IP source field of those packets, one can then easily identify their dominant IP source.

The next two subsections focus on the implementation of two main components in the framework, INT-capable devices using P4 and an INT collector.

B. P4-based INT Networking Devices

1) P4-based Monolithic Application: A P4 program is monolithic which causes difficulty to write programs in a reusable and modular way. Furthermore, the existing P4-based INT frameworks are usually implemented as a main program performing network monitoring. Instead, our main P4 program implements L4S (P4-L4S).

Our INT-based monitoring framework (P4-INT) is improved from the existing P4-based INT [10] to conform to INT version 2.0 by supporting INT-MX mode which does not exist in the version 1.0 of the P4 Working Group specification. Although, our P4-INT do not support the type-length-value (TLV) data type as it is not required for monitoring L4S and P4 does not natively support multiple fields with varying lengths. It is organised as a library that is called by the main program, P4-L4S. Our P4-INT code consists of three main building blocks, so-called `control` in P4 language, to realize the three roles of an INT node. The separation of P4-INT and P4-L4S allows to easily upgrade each component.

In addition to the P4 Working Group metrics which are supported by the devices, we need to implement counters to compute the statistic of the numbers of dropped or marked packets. These counters should not be at the packet level since they accumulate the total numbers of dropped or marked packets. They are globally available across the packets that are in the same traffic type, either LL or CL. We thus use a P4 `register` with four elements to store these counters. A register element is reset to zero to avoid repeatedly reporting once its value has been embedded inside an INT packet.

2) Overhead of Fine-grained Monitoring: The framework needs to capture the L4S device states per packet which would heavily consume device resources if all metrics of all packets were to be analysed, especially with high throughput traffic. To mitigate this high resource consumption, we design P4-INT to perform conditional measurements: on an L4S device it only measures the metrics requested via the INT instruction bit map; the measurement of given metrics and the complete INT-capabilities of the device can be activated or deactivated at runtime by the device control plane.

The overhead in device resource consumption can also be reduced by triggering measurements only for flows matching filters configured by the control plane.

C. INT Collector

1) Fast Reaction in a Short Timescale: A monitoring framework is often deployed together with a detection and reaction framework. It should provide precise networking information enabling the later to react in an appropriate way to ensure the network actually sustains LL services. Our framework provides high precision network state information by relying on a push mechanism that uses network sockets, Kafka or Redis message buses, rather than a temporal database [10], to deliver in near real-time the collected metrics values.

Precisely, we extended our existing network traffic analyser, MMT [11], so that it can act as an INT collector by implementing two new plugins in its deep packet inspection library to parse the two protocols, INT metadata and INT report, that are defined by the P4 Working Group. Originally, MMT is a software solution with a plugin architecture, to passively analyse network packets. By using the new plugin to decode the INT metadata protocol, MMT can be eventually deployed on-the-fly behind an INT-capable device to capture and extract INT metadata directly from the egress packets in the data plane.

2) Huge Reports generated by Collectors: Whereas MMT can analyse network traffic with very high throughput, it may saturate the third-party application with a huge amount of reports. For instance, a collector, without any further processing, will generate one report per INT packet [10]. We overcome this bottleneck by implementing in MMT two new filters. MMT reports only the metric values according to the condition predefined by the users via these filters. As such, the framework gives users fine-grained control on the whole monitoring chain, from collecting metrics to forwarding reports.

The event-based filter allows MMT to generate a report only when some metrics' values change. It reduces unnecessary reports while preserving the fine-grained information. Listing 1 shows an example of an event-based filter to tell MMT to send only the reports, named `vary-latency`, when matching the two following conditions: (i) the *Hop latency* metric, designated in MMT by the term `int.hop_latencies`, is being collected; and (ii) latency values of each queue change with respect to the last report. If these conditions are satisfied, then MMT will send a report containing the values of the prescribed metrics in the `attributes` expression to a Redis message bus that has been configured beforehand.

```
event-report vary-latency {
  event = "int.hop_latencies"
  delta-cond = {"int.hop_latencies","int.hop_queue_ids"}
  attributes = {"ip.src","int.hop_switch_ids","int.
    hop_ingress_times","int.hop_egress_times"}
  output-channel = {redis}}
```

Listing 1. An event-based report to retrieve only the change of queue latency

The query-based filter allows MMT to generate periodically statistic by performing some query operations on a window of INT metadata. The current supported operations are: `sum` that returns the sum of values; `count` that returns the number of values; `avg` that returns the average of values; `var` that returns the variance of values in the group; `diff` that returns the difference between two consecutive values; `last`, `first` that returns the last or first value in the group respectively.

IV. Experimental Evaluation

We set up a testbed to evaluate our framework. It includes: (1) two client-server pairs that allow generating LL and CL traffic, (2) an INT-capable P4-based L4S switch and an INT sink switch between the clients and servers, and (3) MMT that

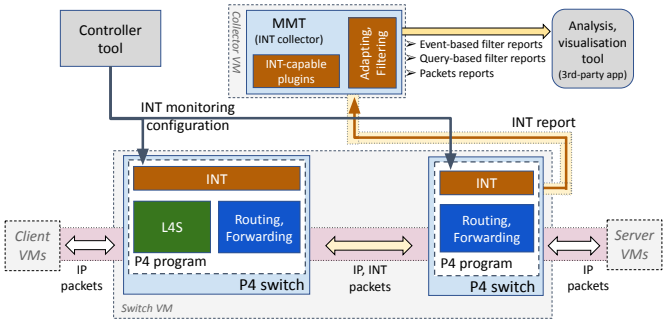

Fig. 1. P4-based monitoring system for L4S leveraging INT

acts as an INT collector. In our test, the client sends data to the server. We used BMv2 virtual switches and their MTU setup is large enough to be able to carry the INT metadata in the packets. These entities are installed inside VirtualBox Virtual Machines (VMs) hosted on a Dell laptop. We use internal networking mode to create software-based networks which are only visible to the concerned VMs and isolated from the other VMs.

Figure 1 details the components inside the switches and the collector of the testbed. We configure the switches via their control planes by using Thrift protocol. The INT monitoring configuration allows to enable or disable INT capability, to set the INT roles, to select the metrics to be monitored, and to select the IP packets to carry INT metadata. The L4S switch on the left side plays both the source and transit roles to extract and embed INT metadata in INT packets which are then forwarded to the next switch which plays the sink role to send INT reports to the collector. Thus the testbed follows the INT-MD mode [7].

A. Overhead on Packet Latency

The INT computation of the device may introduce an extra latency when parsing INT instructions and embedding required metrics into INT packets. Indeed, if all the metrics are collected, the device will add to an INT packet 48 bytes to carry the collected values and 12 bytes of the INT protocol header. Consequently, each packet at the egress port of the switch in our testbed contains 60 additional bytes. To evaluate the latency overhead, we implemented two simple client and server programs to actively measure the end-to-end packet latency. The client inserts the current time in a packet field and sends it to the server. The latter simply sends the packet back. The client then compares the current time and the one embedded in the packet to get the Round-Trip Time (RTT) of the packet without requiring time synchronization between the client and the server. In the case of the measurements without INT, we removed the related P4 code of INT in the L4S switch.

We conducted several measurements with different TCP packet sizes, 100, 200, 500 and 1000 bytes, on two different client-server VM pairs to measure LL and CL traffic. Each measurement sends 10000 packets. We intend to avoid queuing delay that may disturb our measurements as a side effect by introducing a 100 ms delay between two consecutive packets in the client program, and disabling TCP slow start on both the client and server VMs. We present the results in the Cumulative Distribution Function (CDF) diagram in Figure 2.a. The horizontal axis stands for the measured RTT values and the vertical axis for their distribution. We can see that almost all RTT values vary from 2500 to 4000 μs. The average latency without INT for LL and CL packets are respectively 3053 μs and 3101 μs, while these values with INT are 3130 μs and 3234 μs. Therefore, the additional latency increases 2.52% and 4.29% for CL and LL traffic. We can notice that the RTT of LL traffic is slightly higher than that of CL traffic. This difference does not contradict the effort of L4S to achieve low latency. Indeed, L4S tries to reduce the queuing latency which is not taken into account in our measurements. The additional latency is mainly influenced by the computation time of the switch that might require more time to calculate the mark or drop probability of LL traffic. The measured values would be smaller within a testbed using a P4 hardware switch.

B. Overhead on Resource Usage

We then evaluate the overhead of resources usage, such as CPU and memory, of the L4S switch with two configurations: with and without INT processing. We note that the L4S VM has two virtual CPU and 4 GB of RAM. To evaluate the overhead, we used iPerf3 to generate LL and CL traffic from client to server VMs. The traffic was limited to avoid dropping packets. All the 9 sets of metrics are required to be collected when evaluating with INT processing.

Figure 2.b represents the average values of the CPU and memory usage, computed based on 5 measurements of each configuration. We can see that the resource usage of the L4S switch when processing INT is close to the one without INT. Indeed, without INT processing the average CPU and memory usages are 55.57% and 28.97 MB, while the values are 57.84% and 29.16 MB with INT processing. Consequently, the resource consumption of INT processing increases 4.08% in term of CPU usage and 0.66% in term of memory usage. Thus, this stands for a very acceptable value for a monitoring framework.

C. Overhead of Reports

Finally, we examine the overhead which can be reduced by the event-based filter. We ignore the query-based filter as it periodically generates reports without depending on the traffic. We used iPerf3 to generate traffic that was captured at the egress of the switch and saved this capture into a file by using tcpdump. The INT metrics from the captured traffic are then extracted offline by our INT collector. This allows performing several tests on the same traffic with two different configurations of the collector: (1) without using the filter, thus one report is generated for one INT packet; (2) using event-based filter for queue latency, as depicted in Listing 1, and (3) using events-based filter for queue occupancy.

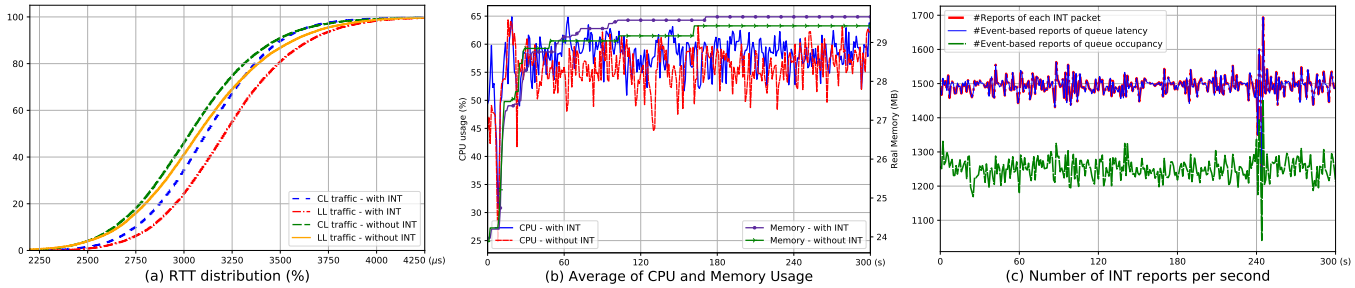

Fig. 2. Overhead of INT processing

Figure 2.c represents the number of reports per second corresponding to theses configurations. Since the queue latency is measured at the microsecond scale, this leads to very few consecutive packets having the same queue latency. Consequently, very few reports are filtered, and the number of queue delay reports is close to that of INT packets. However, the number of queue occupancy reports reduces significantly. This can be caused by the bulk packet processing of the switch, that is, the packets in a same bulk report holds the same queue occupancy number.

During the 300 seconds of the test, the total number of reports without filtering is 448216, i.e., 1494 reports per second. The numbers of reports for queue latency and occupancy are 448120 and 375266 respectively. These results show that one can reduce the report overhead by 0.02% and 16.27% respectively without losing fine-grained monitoring of the queue latency and occupancy.

D. Assessment of the L4S Behaviour

As a second set of tests, we verified the expected behaviour of L4S that can be retrieved through our INT monitoring framework. We used TCP-Prague and TCP Cubic congestion control algorithms on LL and CL VMs, respectively. We generated two TCP flows for each kind of traffic. Bandwidth of each flow is limited to 4 Mbps due to the capacity of our virtual switches. ECN support is enabled on LL VMs and disabled on CL VMs. We evaluated the collected metric values via several tests using iPerf3 to generate LL and CL traffic. We present below the most noticeable results.

In Figure 3.a, we can see that the bandwidth is fairly shared between these two kinds of traffic. The sharing ratio is detailed in Figure 3.d, where it is close to 1. The LL traffic is very slightly dominant but this is not considerable and it conforms to the behaviour of L4S P4-based implementation [6].

We notice some peaks in the bandwidth measurements although we limited each flow to 4 Mbps. We analysed that this phenomenon is not related to our solution but is caused by the virtual environment which impacts the VMs by actually reducing the sending rate of iPerf3 which afterwards tries to increase the sending rate to reach again the limit of 4 Mbps.

The averages per second of queue delay and queue occupancy metrics are presented in Figure 3.b and c respectively. Their corresponding CDF in Figure 3.d and e show the distribution of their values. We can see that 99% of queuing delay values are less than 5 ms for LL traffic and 25 ms for the CL one, which are the target values we configured for the L4S switch for our tests. The CL queue is also more filled than the LL one. This confirms the behaviour of L4S to achieve low-latency by reducing the queue delay.

Despite the fact that the INT technology can introduce overheads in the monitoring framework, we can conclude that our framework is able to provide the metrics values that are similar to those presented in [3], [6] and conform to the expected behaviour of L4S switches.

V. Related Work

The metrics of an L4S device are extracted in our previous work [3] by periodically calling system commands such as ss and tc to poll the statistics from an L4S AQM. Although the polling frequency is very short, e.g., four times per base RTT, it may miss microbursts in traffic, which mostly occur during at most tens of microseconds [12] and induce usually periods of high queue utilization eventually leading to packet delay or loss. To overcome this issue, the current framework does not rely on packet sampling, compressing, aggregating, nor coarse-grained counters. It performs the monitoring at the packet level. Consequently, it will not miss any relevant phenomenon. Hence, it ensures the full coverage of the monitored device states.

The closest work to ours is the monitoring system implemented by the L4S team to evaluate the Linux-based L4S implementation [2], [8]. The authors override 2 bytes of the identification field of the IP packet header to store the L4S information: 11 bits for queue delay and 5 bits for the number of dropped packets. The authors installed a traffic analyser behind the AQM to capture the outgoing packets and extract the information. However, limitation of space in 2 bytes leads to the limitation of precision. By contrast, our framework also stores the collected information inside the packets respecting the INT protocol format, thus more space can be used although one is still limited by the MTU.

A survey of the INT technology is presented in [9]. The authors in [13] proposed another survey of dedicated P4-based INT. The authors in [14] go beyond INT, which only covers the collection of elementary device metrics at the packet level, leaving more complex analysis to external systems, by allow-

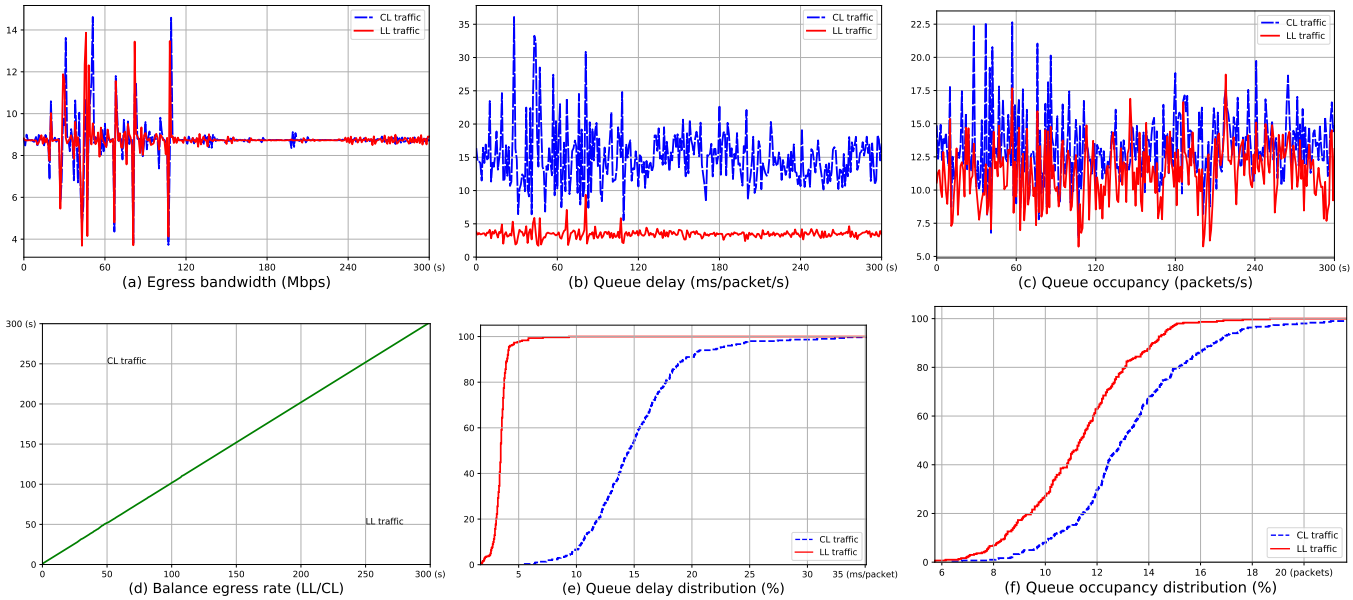

Fig. 3. Behaviour of the L4S switch with two LL and two CL flows of 4 Mbps each flow at the ingress ports.

ing to program more sophisticated analysis and conditional actions directly inside the data plane using the Big Packet Protocol, for instance, to dynamically select which metrics to collect depending on statistics computed from previous data.

We can cite two P4-based implementations of INT in [10], [15]. The authors in [10] propose a high-performance monitoring framework. However it is not suitable for real-time detection or reaction because it can be delayed when using a database to store the collected metrics and then a third-party application needs to query the information from the database. In [15], the authors aim at finding a balance between reducing the payload overhead of INT packets and the accuracy of the network view constructed from the telemetry. The overhead is mitigated by aggregating the INT data, but it is traded for a coarser view of the network state. Our framework overcomes these limitations by using sockets to be able to send reports in near real-time and introducing on-demand monitoring feature which allows the users to select the metrics to be monitored and the conditions to fulfill to report them.

VI. Conclusion and Future Work

We have presented in this paper the initial design and open-source implementation of a framework using INT to monitor P4-based L4S networking devices. The framework provides fine-grained values of the monitored metrics in real-time. The experimental evaluation shows that our solution has the capability to capture the metrics of an L4S switch with a low overhead that does not alter the behaviour of the L4S system.

In our future work, we plan to evaluate its high-precision features by conducting an experiment on a high performance testbed which will be deployed onto physical servers and a hardware switch. Our final objective is to have a network equipment embedding the framework that fully monitors the L4S system for both performance assurance and security purposes.

Acknowledgements

This work is partially funded by the French National Research Agency (ANR) MOSAICO project, under grant No ANR-19-CE25-0012.

References

[1] S. Nádas et al. “A Congestion Control Independent L4S Scheduler,” in *Proc. of ANRW*, 2020, pp. 45–51.
[2] K. De Schepper et al. “PI2: A Linearized AQM for both Classic and Scalable TCP,” in *Proc. of CoNEXT*, 2016, pp. 105–119.
[3] M. Letourneau et al. “Assessing the Threats Targeting Low Latency Traffic: The Case of L4S,” in *Proc. of CNSM*, 2021, pp. 544–550.
[4] P. Bosshart et al. “P4: Programming Protocol-Independent Packet Processors,” *Computer Communication Review*, pp. 87–95, 2014.
[5] F. Paolucci et al. “P4 Edge Node Enabling Stateful Traffic Engineering And Cyber Security,” *Journal of Optical Communications and Networking*, pp. A84–A95, 2019.
[6] B. Mathieu et al. “Evaluating the L4S Architecture in Cellular Networks with a Programmable Switch,” in *Proc. of ISCC*, 2021, pp. 2–7.
[7] The P4.org Working Group “In-band Network Telemetry (INT) Dataplane Specification V2.1,” Tech. Rep., 2020.
[8] O. Albisser et al. “DUALPI2 - Low Latency, Low Loss and Scalable Throughput (L4S) AQM,” in *Proc. of Netdev*, 2019.
[9] L. Tan et al. “In-band Network Telemetry: A Survey,” *Computer Networks*, vol. 186, no. December, 2021.
[10] J. Hyun et al. “Real-time and fine-grained network monitoring using in-band network telemetry,” *International Journal of Network Management*, vol. 29, no. 6, 2019.
[11] B. Wehbi et al. “Events-Based Security Monitoring Using MMT Tool,” in *Proc. of ICTS*, 2012, pp. 860–863.
[12] Q. Zhang et al. “High-Resolution Measurement of Data Center Microbursts,” in *Proc. of SIGCOMM*, 2017, pp. 78–85.
[13] P. M.-Lopez et al. “Passive In-Band Network Telemetry Systems: The Potential of Programmable Data Plane on Network-Wide Telemetry,” *IEEE Access*, vol. 9, pp. 20 391–20 409, 2021.
[14] A. Clemm et al. “Network-Programmable Operational Flow Profiling,” *IEEE Communications Magazine*, vol. 57, no. 7, pp. 72–77, 2019.
[15] G. Simsek et al. “Efficient Network Monitoring via In-band Telemetry,” in *Proc. of DRCN*, 2021, pp. 1–6.