Evaluating the L4S Architecture in Cellular Networks with a Programmable Switch

Bertrand Mathieu, Stéphane Tuffin

Evaluating the L4S Architecture in Cellular Networks with a Programmable Switch

Bertrand Mathieu, Stéphane Tuffin
Orange Labs
2 Av. Pierre Marzin – 2300 Lannion – France
{bertrand2.mathieu, stephane.tuffin}@orange.com

***Abstract*—Low-latency applications, such as cloud gaming or cloud robotics are very demanding in terms of network latency. The IETF defines the L4S (Low Latency Low Loss Scalable throughput) architecture, to enable the delivery of high-bitrate low-latency applications without degrading the quality of other services. To deploy and upgrade L4S in network equipment to follow the transport protocols entering an age of quick evolutions, P4 (Programming Protocol-Independent Packet Processor), a programmable data plane concept, can help since it facilitates the deployment of networking software. In this paper, we propose a P4-based L4S solution and the performed evaluation proves our system behaves as expected. Furthermore, to be largely deployed, L4S should also be efficient under real cellular network conditions, which can vary over time and where the main network bottlenecks are. However, our evaluation shows limitations of L4S in providing high throughput while ensuring low latency delivery with time varying network conditions.**

Keywords— L4S, P4, Low-Latency, AQM, Programmable

I. INTRODUCTION

Network latency is known to be a pain point for a broad family of applications including cloud gaming [1], cloud robotics [2] and tele-robotics [3]. While the amount of information exchanged is highly variable among applications, the delay at which this information is transmitted by networks is a shared issue. End-to-end latency can be improved at different points of the delivery chain: endpoints, network links or network equipment [4]. Existing network solution like prioritization such as with DiffServ [5], or resource reservation can help for managed networks, however these approaches reveal intractable with high bitrate traffic as they starve the other traffic sources. Latency-aware congestion controls are endpoint based solutions, but they are unable to control the latency and can be starved in presence of latency unaware traffic. Another option consists in assisting the traffic source to react to network congestion, like with ECN (Explicit Congestion Control) [6][7], but it needs distinct queues for low-latency applications and other traffic types, leading to possible unfair throughput sharing. To improve the coexistence of different traffic types, the IETF proposes the promising L4S (Low Latency Low Loss Scalable throughput) architecture [8], together with the DualQ Coupled AQM [9], which enables the fair sharing of bandwidth of the low-latency (LL) traffic and other best effort (BE) traffics, via a coupling between the two queueing algorithms. For ensuring low latency, L4S relies on Accurate ECN [10], an evolution of ECN, which allows the sender to react more finely according to the congestion level, but which requires an adapted transport stack, such as DCTCP, designed for data centers, or TCP-Prague, adapted from DCTCP to be deployed over the Internet [11]. Moreover we anticipate that other transport protocols such as QUIC (Quick UDP Internet Connections) and RTP (Real-Time Transport Protocol) and other congestion controls such as RMCAT (RTP Media Congestion Avoidance) will also support L4S in the future.

The L4S system requires upgrades on endpoints but also in network equipment, where evolutions can take a very long time to be deployed. To speed it, the programmable network paradigm was adopted with technologies such as SDN (Software Defined Networking) and NFV (Network Function Virtualisation) and more recently, the P4 (Programming Protocol-Independent Packet Processor) concept [12][13]. This solution makes the network more agile by enabling the quick deployment of data plane networking software that is executed on network equipment hardware, and shortens the update lifecycles.

In this paper, we advocate the use of such concepts, which will be the corner stone of the future networks. We then designed and implemented a L4S solution for a P4-based programmable network equipment, which would allow fast deployment of L4S evolutions, proving the feasibility to make it. L4S is still recent and, to the best of our knowledge, has been evaluated only for constant network conditions. However, in current networks, the main bottleneck is the access network where congestion can often happen because of the limited and varying capacity of the link. Cellular networks, where latency can greatly change, are then a key issue for L4S and it is therefore crucial to evaluate it in such conditions. The evaluations we performed showed that, under time varying network conditions, L4S does not perform as well as in constant bitrate conditions, since it can hardly achieve a good throughput while ensuring low latency, because of the transmission time intervals.

In Section II, we present some works related to ours. In Section III, we describe the L4S solution for a P4-based network switch. The evaluation of the solution and the evaluation of L4S under time varying network conditions is detailed in section IV, before we conclude this paper in section V.

II. RELATED WORK

The relevance of P4 for different use-cases and amongst them for assisting the delivery of low-latency services is

explored in some papers [14]. As the reduction of queuing delays is reckoned to have one of the most performance-gain-to-deployment-cost ratio for end-to-end latency [15], we focus in this paper on this type of solutions. AQM (Active Queue Management) algorithms [16] are used to prevent network congestion, by deciding to drop or to mark packets depending on the filling level of the queue.

Few published papers deal with P4 and AQM, and we can mainly cite three. Noticing the lack of modern AQM, such as CoDel and PIE, in deployed network equipment, the authors of [17] demonstrate and evaluate the implementation of CoDel using P4. Their objective is to fight bufferbloat for any type of traffic rather than to distinguish the LL traffic from the classic traffic. Assuming that programmable traffic management at the data plane can lead to great benefits for QoS, the authors of [18] implement in P4 the PI2 AQM [19] for reducing queuing delay. They proved that implementing a modern AQM in P4 is not tricky and requires only basic bit manipulations at the data plane. A P4-based framework, using DPDK (Data Plane Development Kit), for evaluating AQM algorithms is proposed in [20]. Their goal is focused on the evaluation framework and they demonstrated it using the PIE and RED AQMs. This is then complementary with our approach which is focused on L4S but ours integrates the possibility to emulate time varying network conditions.

An interesting work is the self-tuning queuing delay regulation system proposed in [21]. The objective is to design an AQM system that adapts itself to the network and traffic conditions. This AQM seems promising but the bandwidth sharing is not fully ensured and it is not yet available for programmable switch. Porting it onto a P4 framework could be beneficial to our goal, as we did for L4S.

Regarding the work related to L4S, few papers worth being mentioned. The L4S concept has been first submitted and evaluated with a Linux-based implementation in [22] and refined in [23], where the main concepts of a dual coupled AQM for bandwidth sharing are introduced. It has initiated the creation of an IETF working group [8] aiming at standardizing a L4S architecture. In [24], additional tests are performed to evaluate the Linux-based L4S solution with DCTCP for Datacenters networks and with TCP-Prague for use in operators' networks. In this paper, we reproduce tests similar to [24] in order to prove the P4-based L4S solution behaves as expected and similarly to the Linux-based implementation, but we also performed tests under time varying network conditions, which has not yet been done.

In [25], the authors argue that existing L4S schedulers are not able to handle the fairness degraded by heterogeneous RTTs, and propose the Virtual Dual Queue Core Stateless AQM. Our P4S-based implementation does not have this limitation thanks to our WRR (Weighted Round Robin) optimized scheduler. Furthermore, they did not evaluate the proposed AQM under variables networks conditions.

III. P4-BASED DESIGN & IMPLEMENTATION OF L4S

This section describes the P4-based network equipment which implements the IETF L4S architecture (Fig. 1).

The use of the P4 environment is motivated by the possibility to have an open network equipment allowing operators to quickly deploy programs in response to the coming fast evolution of applications, transport protocols or algorithms. Indeed, our system works with IP/TCP and ECN flags, but we might imagine evolutions or other transport protocols with different headers (e.g., QUIC could soon support ECN, as it has already been implemented in the picoquic stack, RTP could later support it as well), which can easily be taken into account with P4 (i.e. by changing the parser of the P4 program). Similarly, we might think that a better solution than L4S can emerge and implementing it in P4 will allow its fast deployment in network equipment. The possible modifications, changes or upgrades of the parser, classifier or AQMs, and the genericity of the solution are the reasons why we opt for the P4 framework.

The following describes how the P4 program processes packets and explains the role of the L4S modules, shown in Fig. 1, on top of the P4 pipeline (as described by [13]).

- The first component is the P4 parser which extracts the IP and TCP headers of the packet, to evaluate the ECN bit.

Then the packet classifier, implemented in the ingress part of the P4 program assigns the packet either to the LL or BE queue, depending on the ECN value. In the current version, this module is simple and just parses the ECN bit of the Diffserv field of the IP header to detect if the packet belongs to an "Accurate ECN" flow or not. When ECN-capable applications agreed with network operators, such a classification can provide sufficient security. Furthermore, since the program is aimed at being deployed in a network equipment processing packets at line-rate, we should implement a fast and efficient solution. However, we might imagine a classifier adapted to less controlled environment, based on flow patterns analysis (with Machine Learning techniques for instance), which could more securely classify traffic, avoiding malicious applications to play with the ECN flag. If this processing is too complex to be programmed in P4, an option we envision is to deport this network function, which might be deployed as a Virtual Network Function and connect it to a lightweight classifier module developed in P4, running in the network element, remotely configurable by the main module.

We base our implementation on the P4 Linux BMv2 software. However, it is not fully adapted to our needs, since it currently implements a strict priority queueing mechanism (i.e., a lower priority queue is only served if all higher priority queues are empty) and with such a policy, the LL traffic will starve the BE traffic. It is then necessary to rethink the queue management policy of the BMv2 software to have two distinct queues, and we implement a weighted round robin (WRR) scheduler, configured with a 1/16 weight as recommended by the IETF [9] to dequeue packets and achieve the desired bandwidth sharing policy between the LL and BE queues. We also have to add throttling at the queue level since BMv2 only throttled the packet rate at the output port level thus creating an unmanaged bottleneck defeating the purpose of the two AQMs. It should be noted that these modifications have currently to be done in the BMv2 switch because the P4 traffic manager is not yet programmable [17] and we expect next P4 releases to change that.

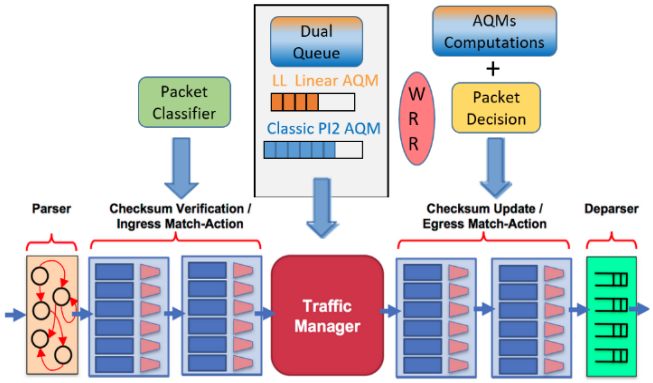

Fig. 1: P4-based implementation of L4S

- We implement the AQMs of L4S in the egress part of the P4 program. The P4 framework offers an API (Application Programming Interface) to retrieve the time spent by one packet in a queue, which is used as input value to the AQMs. P4 also offers registers to store/retrieve data common to the switch i.e. not specific to the packet being processed. They are used for the last update time, the last mark/drop probability and the last delay in the queue, required by PI^2. Finally, the PI^2 parameters (Update time, α, β) are stored in a table of the P4 program and configurable by a controller. The latter can then dynamically modify those values to change the behavior of the PI^2 AQM. We implement a linear AQM and a PI^2 AQM as suggested by the IETF, but if other AQMs were known to be better suited, it would be possible to update network devices with another P4 program. After AQM computations, based on their respective probability, a decision for the LL and BE packets is taken: forward as is, mark or drop. The LL probability is the maximum between the probability output of the native AQM (LL traffic) and the probability of the base AQM (BE traffic) multiplied by a factor K (2 by default). This is where the "coupling" feature is, leading not to penalize the BE traffic.
- Finally, the deparser of the P4 program has to reconstruct the final packet before forwarding it, if the packet has not to be dropped.

IV. Evaluation

A. *Testbed*

For the evaluation, we set up a tested, consisting of a P4-based L4S network switch (or Linux-based L4S program), 1 client and 1 server for a LL connection (using TCP-Prague and Accurate ECN) and 1 client and 1 server for a BE connection (using classic TCP with the Cubic congestion control algorithm). The 5 entities are hosted on the same laptop (a HP Elitebook 840, i5 bi-core and 8 GB RAM), each one in a KVM virtual machine.

We instrument the switch to measure the time spent by packets in each queue, the number of packets being in the queues after the one just sent, and the number of unused transmission opportunities. For bandwidth sharing, we measure the throughput received by each client and the senders' congestion window.

We perform the tests with Iperf3 (during 300sec) and with a HTTP download (using lighttpd/Wget and content files of 10Mb and 100Mb). The sender tries to send as much packets as it can and based on the received ECN marks (indicating congestion) for the LL traffic or on the detection of lost packets for the BE one, it reduces its bitrate.

We configure delay (5ms, 10ms, 20ms, 50ms and 100ms as done in [24]) with NetEm to emulate RTT between the servers and the switch. On the client side, we perform tests with constant bitrate configurations at 4 Mbps, 12 Mbps, 24 Mbps, 40 Mbps and with Mahimahi [26] for emulating time varying access network conditions. The 2 clients share the bandwidth. Finally we configure the P4-based switch to start notifying possible congestion for LL traffic with a low threshold at 3ms. For the BE traffic (PI^2 AQM), this value is configured at 15ms.

For space issues and because some behaviors are similar irrespective of the bandwidth or of Iperf3 or HTTP downloads, we do not present the results of all tests performed, but simply focus on the most noticeable results.

B. *P4-based L4S vs Linux-based L4S*

In this section, our goal is to validate the P4-based implementation of L4S by comparing its results with the ones obtained by the Linux-based reference implementation [23], also validated by [24].

As we can see on Fig. 2 and Fig. 3, bandwidth sharing between the LL traffic and the BE traffic is very fair (almost the same bitrate, ratio close to 1) for our P4 implementation. Our solution is very stable and even better than the Linux-based one, where the ratio decreases as the delay increases. One reason can be that the scheduler is different. For Linux, the authors implemented a Time-Shifted FIFO scheduler, whereas we implement a Weighted Round-Robin one, assumed more stable, as recommended by the IETF [9].

For the time spent by the packets in the two queues (Fig. 4), we can see that our solution ensures that LL packets are delivered in a very short time as L4S requires. Our implementation is a bit less stable than the Linux-based software, but this is not crucial. This is due to the linux L4S software running in the kernel space, while our P4 program runs in the user space, having thus longer and more variable processing times. Indeed, we roughly measured a processing time of a few micro-seconds for the linux L4S module and a time of hundreds of micro-seconds for our P4 implementation. Since the final goal is to deploy such a P4 program onto hardware P4 switch, this issue would be avoided as processing times would be much shorter.

For the BE traffic, the delivery is a bit longer but in a reasonable time, in line with the PI2 configuration of L4S. In our solution, the BE delay in the queue is a bit shorter than the Linux one. This is due to our WRR scheduling implementation, modified to optimize the transmission opportunities. Indeed, if there is no packet in the LL queue while it is scheduled to send one, we send one (if any) of the BE queue so as not to lose this transmission time.

To conclude this comparison, we can say that our P4-based implementation of L4S works very well and offers the expected behavior for both types of traffic.

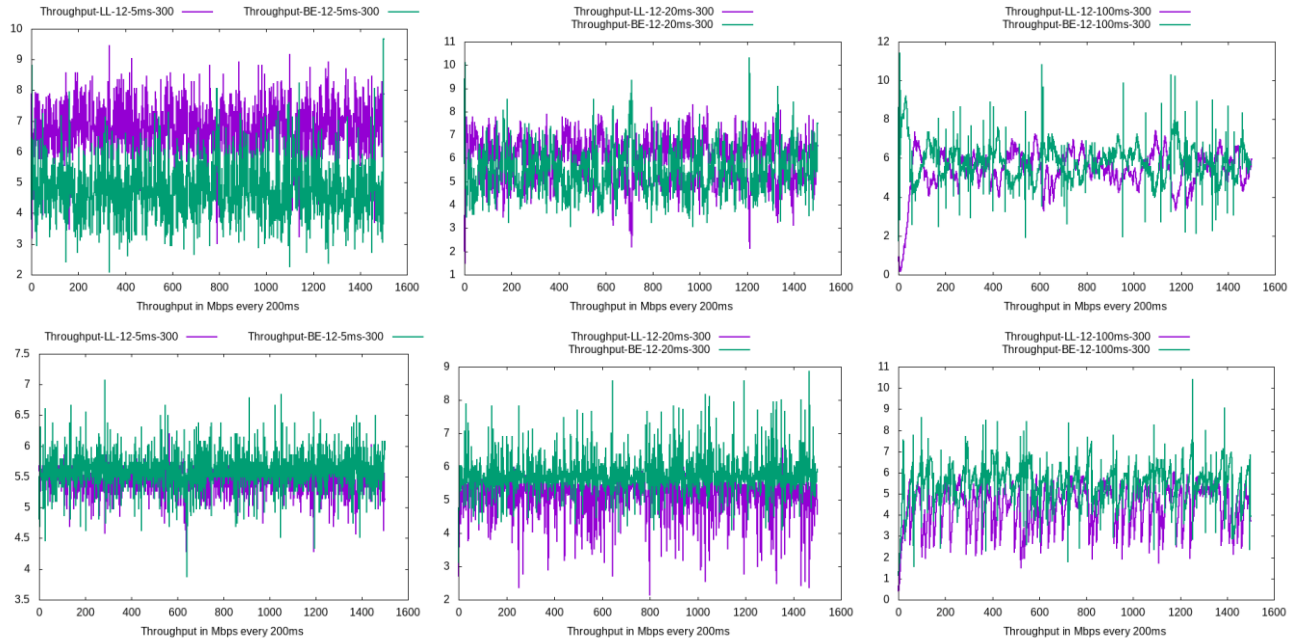

Fig. 2: Bandwidth sharing (in Mbps) with Throughput=12Mbps, with delay=5, 20 and 100ms for top) Linux L4S implementation; bottom) P4-based L4S implementation

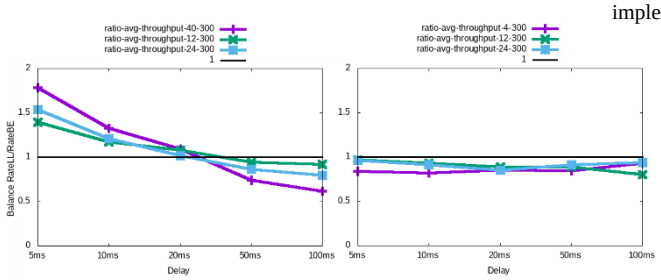

Fig. 3: Bandwidth sharing with delay=5, 10, 20, 50 and 100ms for left) Linux implementation; right) P4 implementation

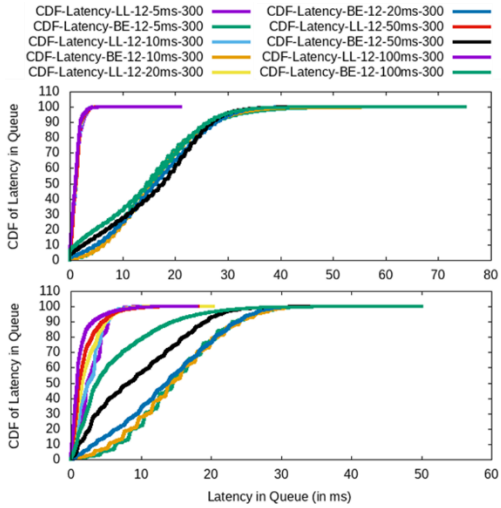

Fig. 4: CDF of latency in queue (LL & BE) with Bandwidth=12Mbps and delay=5, 10, 20, 50 and 100ms for top) Linux; bottom) P4

C. L4S under time varying network conditions

To our knowledge, L4S has been evaluated only under constant network conditions. However, in real cellular networks, the bitrate is not constant and can greatly vary. L4S behavior then needs to be evaluated under such time varying conditions.

Mahimahi [26] offers a tool named Link Shell which allows to schedule the transmission of packets at a given interface, based on the transmission opportunities (txops) specified in a file. Several such files, generated from measurements on real cellular networks, are available as open data in Mahimahi's github [27].

To use the same time varying behaviour as in Mahimahi with our P4-based L4S solution, we modified the scheduling process of the P4 BMv2 switch to reproduce the Link Shell behavior and send packets according to the transmission time provided by the files capturing real txops measurements.

We performed tests with two Mahimahi txops files (Verizon and TMobile dataset) but the L4S system reduced the low-latency traffic to a very low bitrate, sometimes leading to connections resets. Analyzing the datasets showed the mean bitrate is low since the duration between two txops (inter-txops time) is sometimes long, much more than the configured L4S threshold (3ms) and that very few packets could be sent during txops. We think these txops files, extracted from network captures dated from 2016, are no more representative of current cellular bitrates and decided to make our own captures, using the saturator tool [28][29]. This new dataset shows an average bitrate of almost 19 Mbps, shorter inter-txops times and more packets sent during txops. Fig. 6 shows the CDF of inter-txops time and the CDF of the number of packets transmitted per txop for the Verizon and Orange datasets. We can see that with the Orange traces, about 80% of the inter-txops times are less than 3ms, whereas it is below 70% for Verizon, and that 3 or more packets can be sent per txop about 50% of the times, whereas Verizon offers it for only 5% of the time.

With the Orange file, we were able to perform tests but even if the file corresponds to a higher capacity cellular network, the results of the L4S system is far from being satisfactory, compared to the tests performed with a constant bitrate.

Indeed, although the Orange file leads to a mean bitrate of about 19 Mbps, the L4S system behaves less efficiently (Fig. 5-top) than the tests performed with a constant bitrate at 12 Mbps (Fig 2-bottom), the LL traffic throughput being far

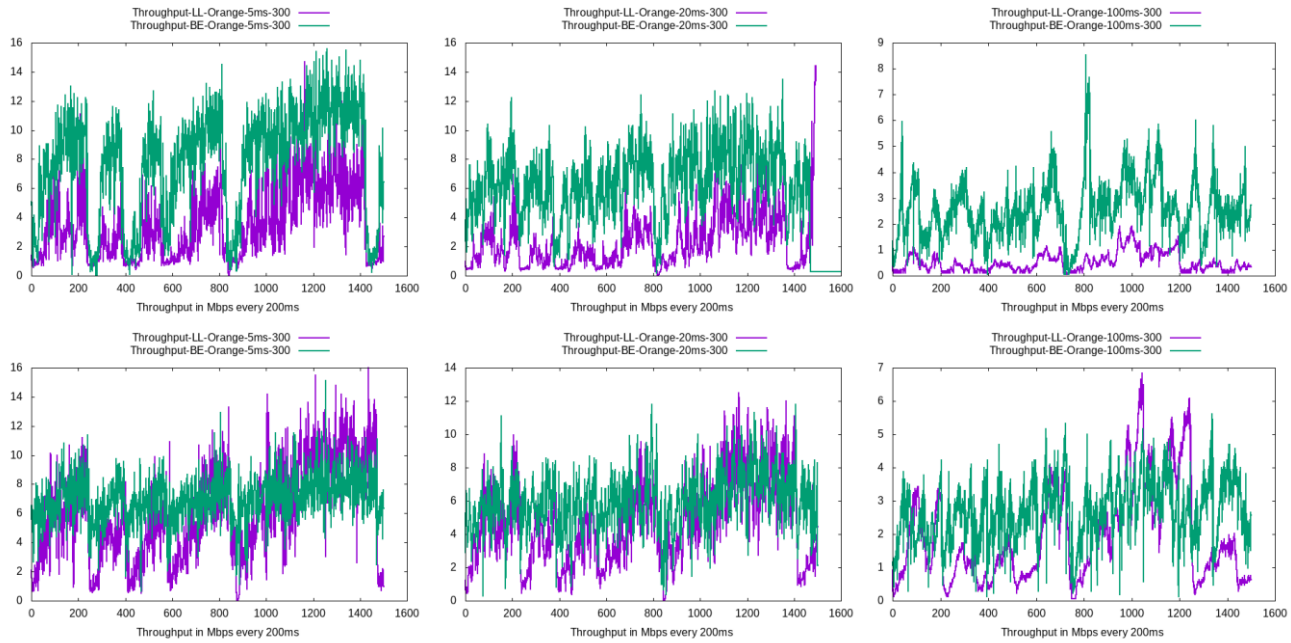

Fig. 5: Bandwidth sharing (in Mbps) with Orange traces, delay=5, 20 and 100ms, for top) L4S with Threshold=3ms; bottom) L4S with Threshold=9ms

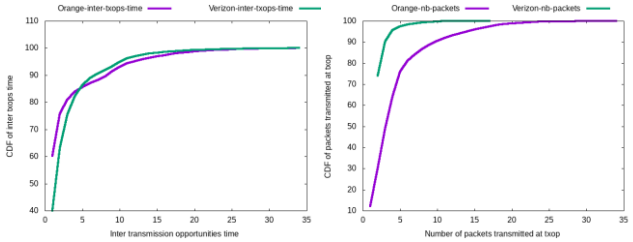

Fig. 6: Analysis of Verizon & Orange Mahimahi traces file, left) inter txops time; right) packets transmitted per txop

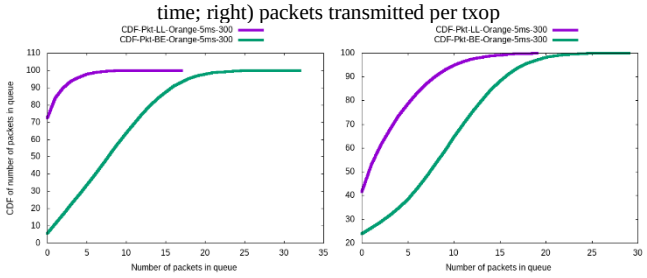

Fig. 7: CDF of the number of packets in queues for P4-based L4S with Orange traces and delay=5ms, for left) L4S Threshold=3ms; right) L4S Threshold=9ms

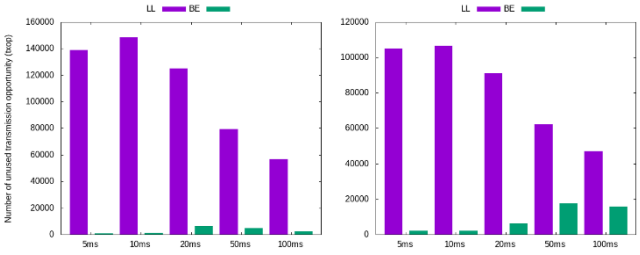

Fig. 8: Number of unused txop for P4-based L4S with Orange traces, for left) L4S Threshold=3ms; right) L4S Threshold=9ms

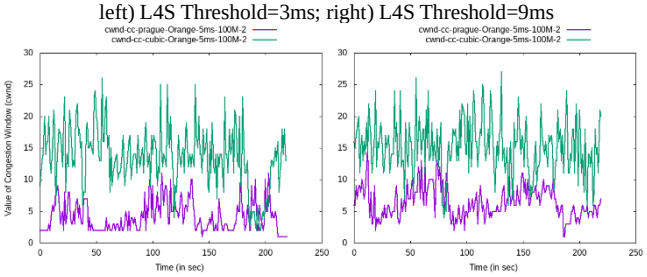

Fig. 9: Congestion Window for P4-based L4S with Orange traces, delay of 5ms for left) L4S Threshold=3ms; right) L4S Threshold=9ms

from the BE traffic throughput. The reason is that L4S very often marks LL packets because the time spent by the packet in the LL queue is frequently more than the configured 3ms threshold. This marking leads to sender's reduction of the bitrate. This is a normal behavior looking at the CDF of the Orange dataset (Fig. 6-left) where we can see than about 20 % of the dataset have an inter-txops time above 3ms. Furthermore the LL queue can only hold a few packets so as to ensure low latency. Indeed, more than 5 packets in the queue leads to marked packets (with a rough estimation of a constant 19 Mbps bitrate, one packet is sent every 0.65 ms, thus 5 packets in the queue are enough to reach the 3ms threshold before marking LL packets). This is confirmed by Fig. 7-left, where the LL queue size contains more than 5 packets less than 5% of the time, while the Orange network can simultaneously send more than 5 packets 20% of the time (Fig. 6-right). Thus, the txops are wasted because there is no packet in the LL queue.

To confirm this, we measure on the P4 node the number of unused transmission opportunities. We can see that this value is huge for the LL traffic, about 140000 for a 5ms delay, which is 28 % of all the txops (Fig. 8-left), while it is about 10000 (3% of txops) for a constant bitrate of 12 Mbps.

We also measure on the server's side the congestion window (with the linux ss tool), and Fig. 9-left highlights that the congestion window is always quite low for the LL traffic, and when it tries to increase, the sender quickly reduces it, because of the marked packets. Contrarily, the congestion window for the BE traffic is much higher.

To double-check that the current L4S architecture, with a low threshold value for LL traffic, is not adapted to current time varying network conditions, we perform the same tests but with a LL threshold configured at 9ms (instead of 3ms). This value is far from being satisfactory for a low-latency application, but it just aims to confirm the effect of the variability of the inter-txops time on the L4S system. With such a threshold and with the same Orange dataset, the results

are much better. We can see that the LL traffic can be transmitted at a higher bitrate, much closer to the BE traffic (Fig 5-bottom), as with a constant bitrate. The number of unused txops (Fig. 8-right), while still high, is lower than the one with a threshold of 3ms (about 30% lower) and that the number of packets in the LL queue size is more than 5 packets about 20% of the time (Fig. 7-right). Looking at the servers' congestion window (Fig. 9-right), we can see than the LL server can increase much more its window, confirming higher delivery rate because of less frequently marked packets.

This evaluation allows us to conclude that the L4S architecture is sensitive to the inter-txops time of the network and requires frequent txops. The number of packets that can be transmitted during a transmission opportunity is less critical since the small number of packets in the LL queue limits its effect. In current cellular networks, L4S is then not really effective and would require shorter time transmission intervals, such as allowed by 5G New Radio numerology.

V. Conclusion & Future Work

In this paper, we have presented a programmable P4-based network switch implementing a L4S solution. The evaluation proved the feasibility of such a solution, which performs as expected, i.e., ensure the forwarding of low-latency packets (latency kept below the target value) while not penalizing other best-effort traffic. We also showed that L4S under time varying network conditions, such as in real cellular networks, does not perform as well as in constant bitrate conditions, since it can hardly achieve a good throughput while ensuring low latency. We presented the recommendation for short transmission time intervals in order to have L4S behaving efficiently. Our future work will be 1) to modify L4S in order to act separately on delays caused by the scheduling of txops by the base station (e.g., influence the target queuing delay objective) from delays imputable to the sending rate of the source (e.g., to mark packets); 2) to implement our solution into a P4 hardware switch (such as those with a Tofino Chipset), prior steps towards possible deployment into real cellular networks.

Acknowledgement

This work is partially funded by the French ANR MOSAICO project, No ANR-19-CE25-0012.

References

[1] M. Carrascosa and B. Bellalta, "Cloud-gaming: Analysis of Google Stadia traffic," in Computer Science ArXiv, 2020

[2] B. Kehoe, S. Patil, P. Abbeel and K. Goldberg, "A Survey of Research on Cloud Robotics and Automation," in IEEE Transactions on Automation Science and Engineering, vol. 12, no. 2, pp. 398-409, April 2015

[3] Avgousti, S., Christoforou, E.G., Panayides, A.S. et al. Medical telerobotic systems: current status and future trends. BioMed Eng OnLine 15, 96, 2016

[4] X. Jiang et al., "Low-Latency Networking: Where Latency Lurks and How to Tame It," in Proceedings of the IEEE, vol. 107, no. 2, pp. 280-306, Feb. 2019

[5] Nichols, K., Blake, S., Baker, F., and D. Black, "Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers", RFC 2474, December 1998

[6] Ramakrishnan, K., Floyd, S., and D. Black, "The Addition of Explicit Congestion Notification (ECN) to IP", RFC 3168, September 2001

[7] N. Khademi, G. Armitage, M. Welzl, S. Zander, G. Fairhurst and D. Ros, "Alternative backoff: Achieving low latency and high throughput with ECN and AQM," 2017 IFIP Networking Conference (IFIP Networking) and Workshops, Stockholm, pp. 1-9, 2017.

[8] B. Briscoe, K. De Schepper, M. Bagnulo, and G. White. "Low Latency, Low Loss, Scalable Throughput (L4S) Internet Service: Architecture". Internet-Draft draft-ietf-tsvwg-l4s-arch-08, Work in Progress, 2020.

[9] K. De Schepper, B. Briscoe, and G. White. "DualQ Coupled AQMs for Low Latency, Low Loss and Scalable Throughput (L4S)". Internet-Draft draft-ietf-tsvwg-aqm-dualq-coupled-14, Work in Progress, 2021.

[10] B. Briscoe, M. Kuehlewind, and R. Scheffenegger, "More Accurate ECN Feedback in TCP", draft-ietf-tcpm-accurate-ecn-11, Work in progress, March 2020

[11] B. Briscoe, K. De Schepper, O. Albisser, J. Misund, O. Tilmans, M. Kuehlewind, R. Scheffenegger, M. Bagnulo, and A. Ahmed. "Implementing the 'TCP Prague' Requirements for L4S". In Proc. Netdev 0x13, 2019

[12] M. Budiu, C. Dodd. "The P4-16 Programming Language", In ACM SIGOPS Operating Systems Review (OSR), Vol. 51, no 1, August 2017.

[13] P4 Open Source Programming Language; p4.org

[14] Kfoury, Elie F. et al. "An Exhaustive Survey on P4 Programmable Data Plane Switches: Taxonomy, Applications, Challenges, and Future Trends." ArXiv abs/2102.00643, 2021.

[15] B. Briscoe et al., "Reducing Internet Latency: A Survey of Techniques and Their Merits," in IEEE Communications Surveys & Tutorials, vol. 18, no. 3, pp. 2149-2196, thirdquarter 2016

[16] R. Adams, "Active Queue Management: A Survey," in IEEE Communications Surveys & Tutorials, vol. 15, no. 3, pp. 1425-1476, Third Quarter 2013

[17] R. Kundel, J. Blendin, T. Viernickel, B. Koldehofe and R. Steinmetz, "P4-CoDel: Active Queue Management in Programmable Data Planes," 2018 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN), Verona, Italy, 2018

[18] C. Papagianni and K. De Schepper, "PI2 for P4: an active queue management scheme for programmable data planes," inProceedings of the15th International Conference on emerging Networking EXperimentsand Technologies, pp. 84–86, 2019.

[19] Rohit P. Tahiliani, Hitesh Tewari. 2017. Implementation of PI2 Queuing Discipline for Classic TCP Traffic in ns-3. In Proceedings of Workshop on ns-3, Porto, Portugal, June 2017

[20] S. Laki, P. Vörös, and F. Fejes. "Towards an AQM Evaluation Testbed with P4 and DPDK". In Proceedings of the ACM SIGCOMM, New York, NY, USA, pp 148–150, 2019

[21] Kahe, G., Jahangir, A.H. "A self-tuning controller for queuing delay regulation in TCP/AQM networks". Telecom Syst 71, pp 215–229, 2019

[22] K. De Schepper, O. Bondarenko, I. Tsang, and B. Briscoe. "PI^2: A Linearized AQM for both Classic and Scalable TCP". In Proceedings of the 12th International on Conference on emerging Networking EXperiments and Technologies (CoNEXT), pp 105—119, Irvine, California, USA, December 12-15, 2016

[23] O. Albisser, K. De Schepper, B. Briscoe, O. Tilmans, and H. Steen, "DUALPI2 - Low Latency, Low Loss and Scalable (L4S) AQM", . In Proc. Netdev 0x13, 2019.

[24] D. BoruOljira, K-J. Grinnemo, A. Brunstrom, and J. Taheri. "Validating the Sharing Behavior and Latency Characteristics of the L4S Architecture". SIGCOMM Comput. Comm. Rev. 50, 2, 37–44, April 2020.

[25] S. Nádas, G. Gombos, F. Fejes, and S. Laki. "A Congestion Control Independent L4S Scheduler", In Proceedings of the Applied Networking Research Workshop (ANRW '20). Association for Computing Machinery, New York, NY, USA, 45–51, 2020.

[26] R. Netravali, A. Sivaraman, S. Das, A. Goyal, K. Winstein, J. Mickens, and H. Balakrishnan. 2015, "Mahimahi: accurate record-and-replay for HTTP", In Proceedings of the USENIX Conference on Usenix Annual Technical Conference (USENIX ATC '15), USA, 417–429, 2015

[27] Mahimahi github; https://github.com/ravinet/mahimahi/tree/master/traces

[28] K. Winstein, A. Sivaraman, and H. Balakrishnan.."Stochastic forecasts achieve high throughput and low delay over cellular networks". In Proceedings of the 10th USENIX conference on Networked Systems Design and Implementation (nsdi'13), pp 459—471, Lombard, IL, USA, April 2-5, 2013

[29] Saturator github; https://github.com/keithw/multisend/blob/master/sender/saturatr.cc