Active Queue Management on the Tofino programmable switch: The (Dual)PI2 case

Gergő Gombos*, Maurice Mouw†, Sándor Laki*, Chrysa Papagianni† and Koen De Schepper‡
*Faculty of Informatics, ELTE Eötvös Loránd University, Budapest, Hungary
†Informatics Institute, University of Amsterdam, Amsterdam, The Netherlands
‡Network Systems & Security Research Lab, Nokia Bell Labs, Antwerp, Belgium
ggombos@inf.elte.hu, mmouw@os3.nl, lakis@inf.elte.hu, c.papagianni@uva.nl, koen.de_schepper@nokia-bell-labs.com

ICC 2022 - IEEE International Conference on Communications | 978-1-5386-8347-7/22/$31.00 ©2022 IEEE | DOI: 10.1109/ICC45855.2022.9838674

***Abstract*—The excess buffering of packets in network elements, also referred to as *bufferbloat*, results in high latency. Considering the requirements of traffic generated by video conferencing systems like Zoom, cloud rendered gaming platforms like Google Stadia, or even video streaming services such as Netflix, Amazon Prime and YouTube, timeliness of such traffic is important. Ensuring low latency to IP flows with a high throughput calls for the application of Active Queue Management (AQM) schemes. This introduces yet another problem as the co-existence of scalable and classic congestion controls leads to the starvation of classic TCP flows. Technologies such as Low Latency Low Loss Scalable Throughput (L4S) and the corresponding dual queue coupled AQM, DualPI2, provide a robust solution to these problems. However, their deployment on hardware targets such as programmable switches is quite challenging due to the complexity of algorithms and architectural constraints of switching ASICs. In this study, we provide proof of concept implementations of two AQMs that enable the co-existence of scalable and traditional TCP traffic, namely DualPI2 and the preceding single-queue PI2 AQM, on an Intel Tofino switching ASIC. Given the fixed operation of the switch's traffic manager, we investigate to what extent it is possible to implement a fully RFC-compliant version of the two AQMs on the Tofino ASIC. The study shows that an appropriate split between control and data plane operations is required while we also exploit fixed functionality of the traffic manager to support such solutions.**

***Index Terms*—P4, AQM, L4S, TCP**

I. Introduction

Cluttered video, audio drops or delay when playing an online game or during a Zoom session. Latency is frustrating; a small increase can result in reduced performance for many of today's applications. One of the causes for latency and jitter is the excess buffering of packets in switched networks, also referred to as *bufferbloat*. To address the strict low-latency requirements of emerging applications scalable TCP variants like Data Center TCP (DCTCP) and TCP Prague have been developed. Their use is however limited to datacenters, as their co-existence with classic congestion controls, leads to starvation of classic TCP flows. Active Queue Management (AQM), while not new, has regained interest to help address the problems regarding bufferbloat in combination with these newer technologies. To support capacity-seeking applications that want both full link utilization and low queuing delay, a new Internet service called Low-Latency Low-Loss Scalable-Throughput (L4S) [1] has been proposed that combines scalable and classic congestion signals from the network. To support the incremental deployment of L4S, the DualQ Coupled AQM concept is introduced that suggests the use of separate queues for L4S and Classic traffic. Fairness among different flows is achieved by a coupled packet dropping/marking mechanism. DualPI2 AQM [2] is a realization of this concept, that uses PI2 as the coupled AQM [2].

Transitioning from existing AQM and Congestion Control technologies to newer ones creates additional challenges. While such AQM algorithms can be implemented and deployed rather easily on software-based switches, backbone and data center links are typically equipped with fixed-function high-speed switching ASICs that usually provide a restricted set of standard AQM schemes. Advances in data-plane programmability (e.g., reconfigurable match tables [3], the P4 programming language [4], etc.) and the advent of programmable network switch ASICs, enable the rapid introduction of new features and easy customization of the network. Unfortunately, while programmable data plane solutions such as P4 and supported architectures, provide an excellent way to define the packet forwarding behavior of network devices, programmable devices still typically have fixed function traffic management engines.

In this study, we provide proof of concept implementations of two AQMs that enable the co-existence of scalable and traditional TCP traffic, on an Intel Tofino switching ASIC, namely DualPI2 and the preceding single-queue PI2 AQM. Regarding the latter, the P4 program for the BMV2 reference switch is provided in [5]. However, porting the function to a hardware target is not a straightforward task. Given the complexity of PI2 and DualPI2 algorithms, we investigate to what extent it is possible to implement a fully RFC-compliant version of the two AQMs. Towards that end, we (i) adopt an appropriate split between control and data plane operations that allow us to overcome the constraints of the Tofino match-action pipelines, and (ii) exploit fixed functionality of the traffic management engine i.e., scheduling approaches, to support such solutions.

In the remainder of the paper, Section II provides an overview of the P4 programming language and describes the functionality of the PI2 and DualPI2 AQM algorithms. Section III describes in detail their corresponding implementation in P4. In Section IV, we compare and validate our implementation via experimentation. Section V provides an

overview of related work. Finally, in Section VI, we highlight our conclusions and discuss directions for future work.

II. Background

The technologies relevant for this paper are discussed briefly in this section.

A. Active Queue Management

The PI2 algorithm proposed by De Schepper et al. [2] uses the Proporional-Integral (PI) controller to determine the output probability p'. The controller periodically (T) samples the queuing delay to update p'. Any queue change (growth or shrink) is quickly corrected by a proportional gain factor β and any residual error (deviation from the target) is relatively slowly pulled back to the target by an integral gain factor α based on $p' = p' + \beta(\tau_{t-1} - \tau_t) + \alpha(\tau_0 - \tau_t)$. Fig. 1 depicts how this simple PI controller algorithm is integrated into the PI2 AQM. Packets are first classified based on the 2-bit ECN field in packet IP header. This determines if the packets will go further to be dropped or marked. The output of the basic PI controller (p') is squared when dropping Cubic packets (think twice to drop) or doubled when marking L4S traffic, to support fairness between them. Note that the notation $p' > R^2$ means that we choose two independent random variables uniformly and both need to be less than p', resulting in the squared drop probability p'^2.

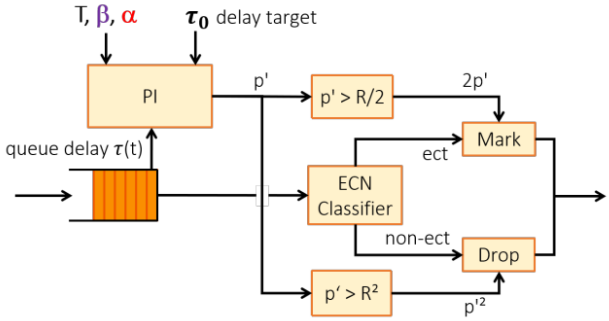


Fig. 1: Overview of the PI2 AQM algorithm. [2]

DualPI2 [6] maintains separate queues for L4S and Classic traffic and applies time-shifted FIFO scheduling between the two queues, while fairness among different flows is achieved by a coupled packet dropping/marking mechanism. The L4S queue runs a low latency AQM with a very shallow ECN marking threshold. DualPI2 uses PI2 as the coupled AQM.

B. The P4 Programming Language

P4 is a domain-specific language for programming packet processors. [4]. P4 allows engineers to write the processing behaviour of packets on a programmable device called a *target*, which could be a device such as a FPGA, network processor, ASIC etc.While the P4 language is designed to be protocol-independent and target-independent, a target does require a compiler that maps the P4 source code into the target's instruction set. The current specification of the language (P4_16), introduced the concept of the P4 architecture that defines the P4-programmable blocks of a target and their data plane interfaces. Along with the corresponding P4 compiler, it enables programming the P4 target. Depending on the target, there are different architectures/models that can be used such as the v1model, Portable Switch Architecture (PSA) or the Tofino Native Architecture (TNA). Each of these architectures can provide their own custom externs as well as their limitations.

The TNA is a proprietary architecture originally created by Barefoot. TNA is very similar to the standard PSA architecture with some additional features [7].

III. (Dual)PI2 on Tofino

A. Porting PI2

***Limitations*:** Congestion detection requires timely information on the state of the queue, while the decision to drop a packet takes place at the ingress. However the PI2 AQM in [5] is integrated into the P4 egress pipeline, where the P4 program retrieves queuing information as part of the standard packet metadata, and potentially drops any packets thereafter. For the P4 architecture TNA the situation is similar, but this may change with the new generation of Tofino ASICs, enabling easier deployment of AQM schemes.

We encountered several noteworthy constraints in the Tofino ASIC while porting the PI2 v1model code to the TNA. The first important limitation is that multiplications needed for implementing the PI controller cannot be used as flexible as in software targets. Hardware targets like the Tofino ASIC were designed to achieve high-speed packet processing performance and thus the use of complex arithmetic operations is constrained. Divisions and multiplications needed for implementing the PI control could however be approximated by bit shift operations. We first applied these approximations to replace the original computations in the v1model-based PI2 code. However, our attempt proved to be too complex for a single MAU, as it requires multiple sequential bit shift operations to determine the output probability p'. Note that such an approximation can be implemented so that it occupies multiple stages of the pipeline.

Moreover, the PI2 program for BMV2 requires several read and write operations using several registers while processing a packet through a pipeline. However, a read operation followed by a write operation on the same register for the same packet while it traverses a pipeline is not allowed for the TNA.

Implementation: To overcome the aforementioned limitations, the choice was made to implement a part of the PI2 algorithm in a custom control plane application, similar to [8]. Fig. 2 provides an a overview of the PI2 AQM algorithm implementation in P4 using the TNA for the Tofino target, highlighting the split between control and data plane operations.

The ingress part of the P4 code (not depicted in Fig. 2) forwards the ingress global timestamp of each packet via a bridged header as metadata to the egress pipeline. The queuing delay τ_t is determined using the ingress and egress timestamps

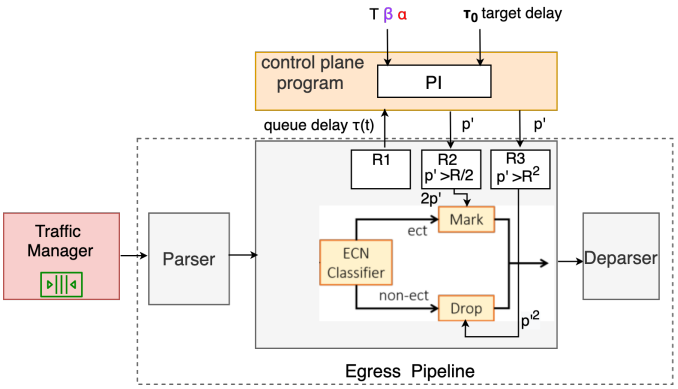


Fig. 2: Overview of the PI2 AQM implementation for TNA.

for the packet and is stored in register *R1*. The control-plane periodically retrieves the queuing delay from *R1* on the data-plane and uses it in the PI formula described in Section II to determine the probability factor p' every update time T. The control-plane then stores the updated probability p' in two registers (*R2* and *R3*) on the data-plane. Using appropriate register actions and arithmetic operations as described in [5], in the first register we check if $p' > R/2$ while for the second if $p' > R^2$ (meaning that two values chosen uniformly at random are less than p'.). The corresponding Boolean values are used for deciding whether to mark scalable TCP traffic or drop Classic TCP traffic, respectively.

B. DualPI2 Implementation

The implementation of DualPI2 requires the use of two distinct queues. The original algorithm applies a time-shifted FIFO scheduling between L4S and Classic queues that is too complex for P4 programmable ASICs. By using weighted round robin as the conditional priority scheduler, the L4S traffic can sacrifice some throughput during overload to guarantee a minimum throughput service for classic traffic, allowing the output probability p' from the corresponding classic queue to be sampled. The congestion signals for the two physical queues are coupled, similar to the single-queue coupled AQM PI2 using a stronger signal in L4S queue, and weaker in the classic one. We employ Deficit Round Robin that is supported by the traffic manager of the Tofino asic, setting the scheduling weight of the Classic queue to 1/10.

The ingress part of the P4 DUALPI2 program (not depicted in Fig. 3) forwards the ingress global timestamp of each packet (L4S or classic TCP) via a bridged header as metadata to the egress pipeline, similar to the aforementioned PI2 implementation. Moreover it implements an IP-ECN classifier, forwarding L4S and classic TCP traffic to the assigned L4S and classic queue respectively. The control plane (i) estimates the probability factor p', using the delay experienced by non-scalable TCP traffic at the classic queue and (ii) sets the weights and priorities for the queues scheduled using DRR. The L4S queue runs at egress a simple ECN step AQM with a marking threshold set at T=1ms. Thus, as described in [9] an L4S packet can be marked either by its native AQM if $\tau_{L4S}(t) > T$ or by the coupled AQM if $p' > R/2$.

IV. Proof of Concept

Evaluation Environment and Reporting. The evaluation of the proposed P4-based PI2 and DualPI2 implementations has been carried out in a testbed that consists of a P4 programmable Stordis BF2556X-1T switch and two communicating hosts: 1) *Traffic Source* and 2) *Traffic Sink*. The traffic is generated by the *iperf* tool. The base RTT is emulated on *Traffic Sink* using the *tc netem* tool, delaying the ACKs of TCP flows. The links between the switch and machines had a capacity of 40 Gbps. The bottleneck is emulated by the P4-switch (the outgoing port towards *Traffic Sink*). Flows apply either TCP Cubic as classic congestion control or TCP Prague as L4S congestion control.

Experiments, Measurements & Evaluation Metrics. In this section, we examine the performance of the proposed PI2 and DualPI2 implementations. We consider various three different bottleneck capacities (120 Mbps, 200 Mbps and 1 Gbps) and five base RTTs (5, 10, 20, 50 and 100 ms). The target delay in both PI2 and DualPI2 is 20 ms, similarly to prior work. The ECN marking threshold in the step AQM of DualPI2 used for handling L4S traffic is set to 1 ms. In each scenario, we consider TCP flows with both Cubic and Prague congestion controls while the number of flows are varied. Our investigation focuses on the per-flow goodput reported by *Iperf*, the aggregated goodput of classic Cubic and L4S Prague flows, the total traffic (i.e., utilization), flow-fairness, the observed queueing delay and the drop probability calculated by the two applied AQMs. Due to page limitations, we only show figures for selected experiments.

1) Increasing traffic intensity: Our first measurement scenario is composed of four 120 s long phases. In each phase, new flows enter the system, resulting in increasing traffic intensities. The numbers of Cubic and Prague flows in the four phases are 1-1, 2-2, 10-10 and 25-25. Due to the limited page numbers, we only include selected measurements with

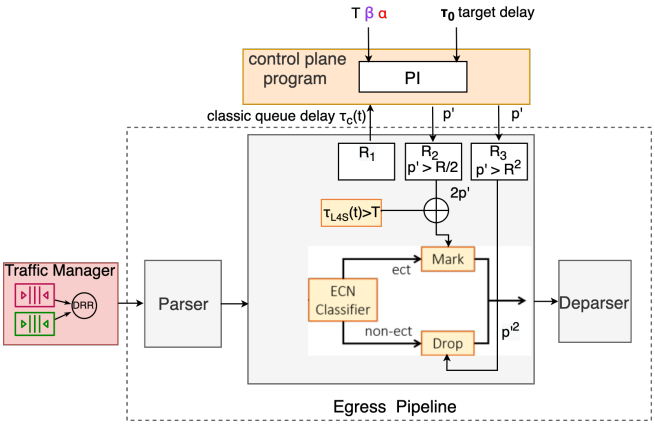
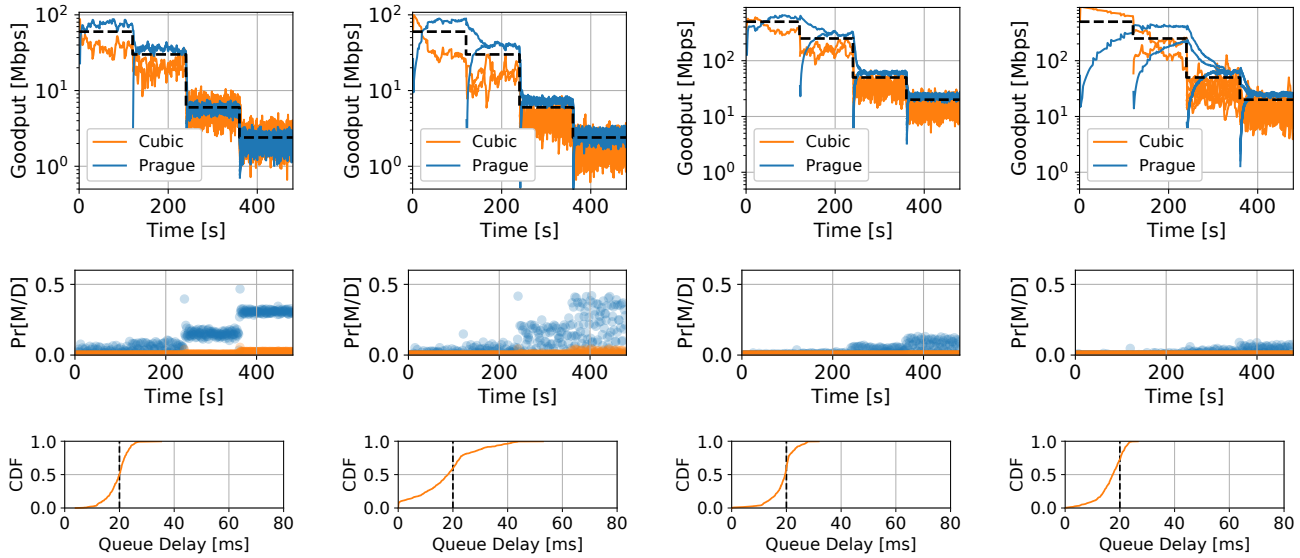
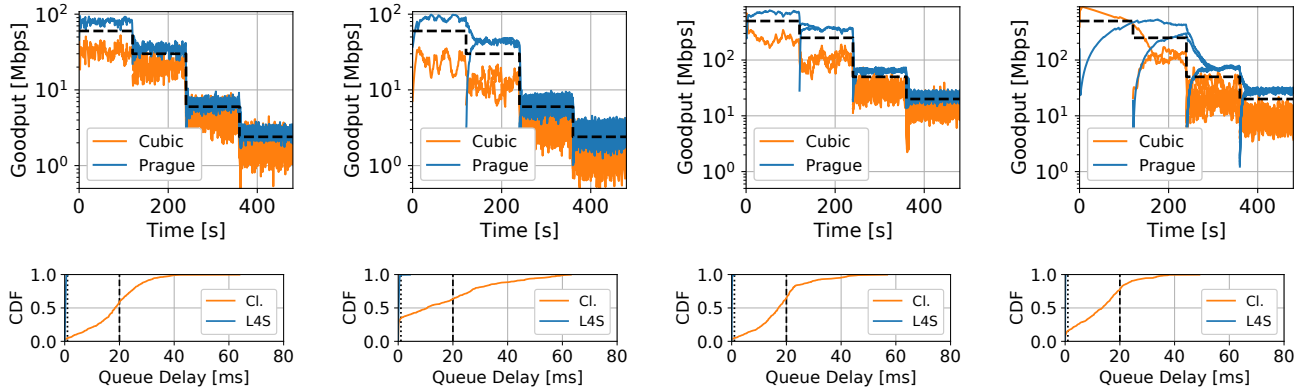


Fig. 3: Overview of the DualPI2 implementation for TNA.



(a) 120Mbps, 10ms **(b) 120Mbps, 50ms** **(c) 1Gbps, 10ms** **(d) 1Gbps, 50ms**

Fig. 4: PI2 AQM. The columns represent scenarios with different bottleneck capacities and base RTTs. Each column shows the per-flow goodput values (top), the applied drop probabilities (middle), the CDF of observed queue delays (bottom).



(a) 120Mbps, 10ms **(b) 120Mbps, 50ms** **(c) 1Gbps, 10ms** **(d) 1Gbps, 50ms**

Fig. 5: DualPI2 AQM. The columns represent scenarios with different bottleneck capacities and base RTTs. Each column shows the per-flow goodput values (top) and the CDFs of observed Classic and L4S queue delays (bottom).

two bottleneck capacities: 120 Mbps used as baseline whose results can be compared to the ones of [2], [6] and 1 Gbps.

Our first experiments depicted in Fig. 4 focus on the performance of the proposed PI2 AQM implementation. The left two columns depict reference experiments at a low speed bottleneck of 120 Mbps with similar settings to the ones used in [2], [5]. Not surprisingly, Prague flows get higher throughput share when the number of flows are limited (in the first two phases), as Cubic flows significantly reduce their sending rate even for a single packet drop. As the number of flows increases, the flow fairness between Prague and Cubic ones is getting better, regardless of the RTT settings. One can also observe that the scalable congestion control of Prague results in more stable per-flow throughput than the classic Cubic. Larger deviations from the target delay are only visible when the RTT is 50 ms (2.5x the target delay) and the number of flows are large (last two phases). In this case, Cubic flows require larger buffers to achieve good utilization, reflected by the observed queueing delays. The base drop probabilities are in accordance to the observations reported in [2]. For the 50 ms RTT the larger queue delays caused by the slower feedback loop of TCP are handled by increased drop probabilities.

The right two columns of Fig. 4 depict the same measurements with a 1 Gbps bottleneck. When the number of flows are limited (first two phases), both Cubic and Prague flows need more time to ramp up their congestion window and get close to the desired fair share. It is visible even for the case of 10 ms RTT, but more significant for 50 ms RTT, indicating

that Prague flows start slower. The queue delays are controlled by the algorithm in all cases.

Our DualPI2 AQM design has more differences to the reference implementation [6] since the time-shifted FIFO scheduling is too complex for hardware switches. To get comparable results to PI2 and experiments described in [6], we also show measurements carried out at a slow-speed bottleneck (120 Mbps). The left two columns of Fig. 5 depicts that DualPI2 results in similar per-flow goodput and classic queueing delay to PI2 at 120 Mbps. Prague and Cubic flows show a fair behavior. The slightly higher queue delay can be seen in the last two phases, potentially caused by the non-equally weighted deficit round robin scheduling between L4S and Classic queues and the smaller L4S queue delays. Note that the drop probabilities are similar to the PI2 case.

At a 1 Gbps bottleneck, Prague flows get somewhat higher throughput share than with PI2. Prague/L4S packets experience sub-millisecond queue delays and thus they can ramp up their sending rates faster than Cubic ones where the queue delay is kept around the target. As the traffic load is increasing (last two phases), the queue delay shows larger deviance from the target value, but it is still controlled.

2) Large-scale measurements: In addition to the selected cases presented in the previous section, we summarize the measurements in all the investigated cases. Due to page limitations, the figures only illustrate aggregated values (25th and 99th percentiles, and mean) of two metrics: 1) the relative goodput error calculated for each flow as $\frac{flow\ goodput - fair\ share}{fair\ share}$ and 2) the observed queue delays.

First, we consider a symmetric case when the numbers of Cubic and Prague flows are equal. Fig. 6 depicts 48+48 aggregated measurements for both PI2 and DualPI2 methods. The labels on the x-axis show the settings of each experiment, including the bottleneck capacity, base RTT and the number of Classic-L4S flows. Each experiment consists of a 120s long measurement with static traffic. One can observe that the absolute relative error in average lays below 0.5 for both methods. Larger deviations can only be seen in the large RTT cases (50ms and 100ms). High-speed bottleneck, small flow numbers and large RTT lead to the slow start of Prague flows (similarly to Fig. 5-d). Queue delays are controlled in all the cases. The coupled AQM of DualPI2 leads to somewhat larger queue delays for packets of classic flows, while L4S flows experience almost zero queue delay.

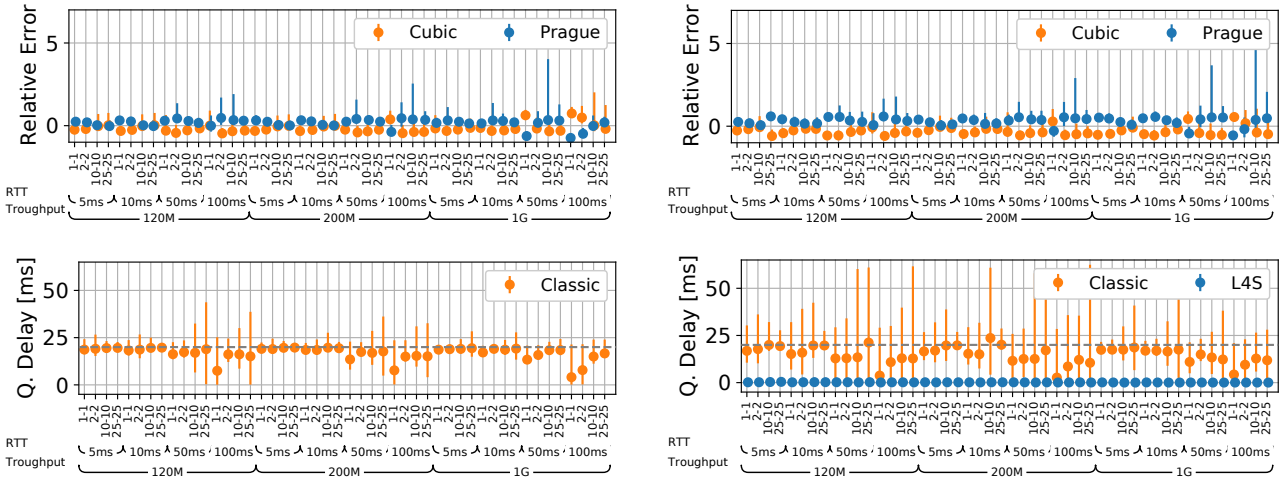
The second scenario depicted in Fig. 7 assumes a more realistic scenario where the traffic mix is dominated by Classic flows and the share of L4S flows are only 10%; the number of Cubic and Prague flows for each bottleneck-RTT settings are 9-1, 45-5 and 90-10. Though in most cases Prague flows get higher share of the joint bottleneck (esp. with DualPI2) but the relative error of Cubic flows remains around zero, indicating that L4S flows do not violate the less aggressive Classic flows significantly. The queue delays are controlled and the classic delay only shows deviations in case of large RTTs (50ms or 100ms). The L4S delay and its deviation is negligible.

Fig. 8 depicts other asymmetric scenarios where the share of L4S flows in the mix is 90% and only 10% of flows use Cubic congestion control. For both PI2 and DualPI2 the average relative goodput error is small, indicating fair behaviour between the two traffic classes. The largest deviance can be seen at a 1Gbps bottleneck with 100ms RTT when a single Cubic flow competes against nine Prague flows. Cubic gets higher throughput share since it ramps up its sending rate much faster than Prague similarly to the cases shown in Sec. IV-1. Though the queue delays are controlled by both methods, DualPI2 results in large L4S delays when the RTT is small and the per-flow throughput share is limited. This deviation disappears as we increase the number of flows or the base RTT, or both.

The low-speed evaluation results are in align with related literature. In addition, we also performed experiments at 1 Gbps where DualPI2 and PI2 have not been validated before. At this bottleneck speed, both methods show good flow fairness, except the case of 100ms RTT. The combination of a high capacity bottleneck and large RTT slows down the convergence of sending rates to the fair share and a single packet drop can result in huge per-flow throughput decrease.

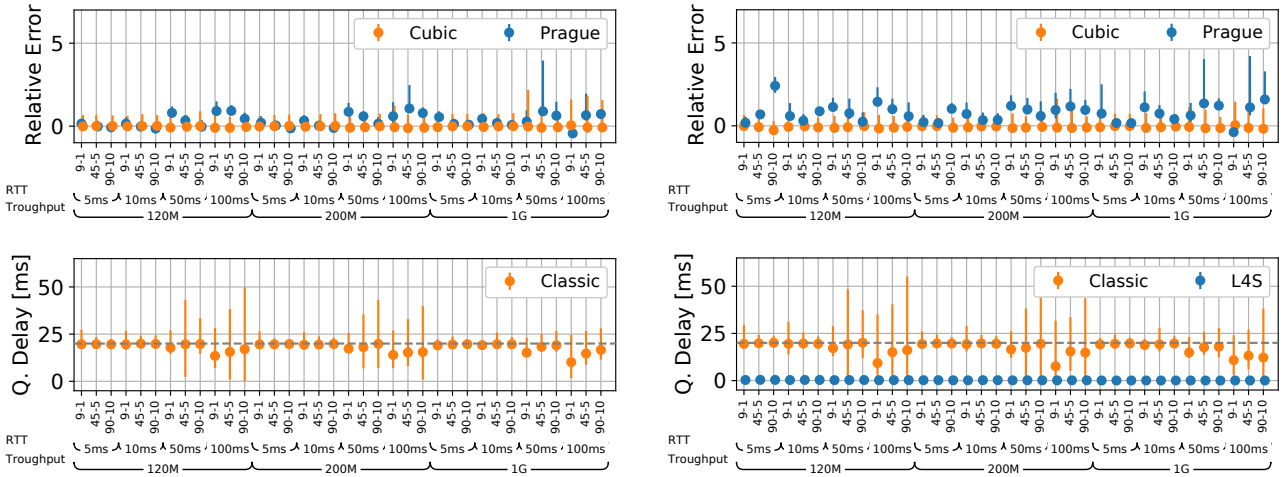
V. Related Work

Research into AQM algorithm implementations on programmable data-planes is a current and active research topic. Sivaraman et al. [10] enable programmability at the data plane by adding an FPGA to the fast path of a hardware switch using an interface to packet queues, implementing CODEL and RED as a proof of concept. Kundel et al. [11] demonstrated that is possible to implement such algorithms for P4 programmable data planes, by providing a proof-of-concept implementation of CODEL for bmv2. The main difficulty was the lack of support for complex arithmetic functions (mainly the square root). The authors employed the P4 longest prefix match table feature to map approximations for the square root. In [5], a PI2 implementation in P4 is provided for reference switch BMV.In [12] authors present an implementation of activity-based congestion management using P4, introducing additional target-specific externs for floating point operations. Finally, in [13] PIE and RED AQM schemes are implemented using P4 for DPDK. Kunze et al. [8] created three different implementations of the PIE algorithm in P4, each using different mechanisms available in TNA. The first method relies on the control plane, the second on data plane registers and the third on table look-ups. The control plane variant performed the best, for which the speed was close to line-rate although it failed to match the target delay. The authors state that design choices based on the identified constraints can have significant impact on performance. Since none of the implementations successfully controlled the queuing delay we chose to employ a similar approach using the control plane for the PI2 and DualPI2 algorithms on TNA as this seemed to be the most promising one.



(a) PI2 **(b) DualPI2**

Fig. 6: Symmetric case with the same number of Cubic and Prague flows. Each column in the figures illustrates a 120s long experiment with static flows and settings. The labels on the x-axis include the Cubic-Prague flow numbers, the emulated base-RTT and the bottleneck capacity. For each cases, the dots represent the observed mean values while bars illustrate the 25th and 99th percentiles of both relative errors and queue delays.

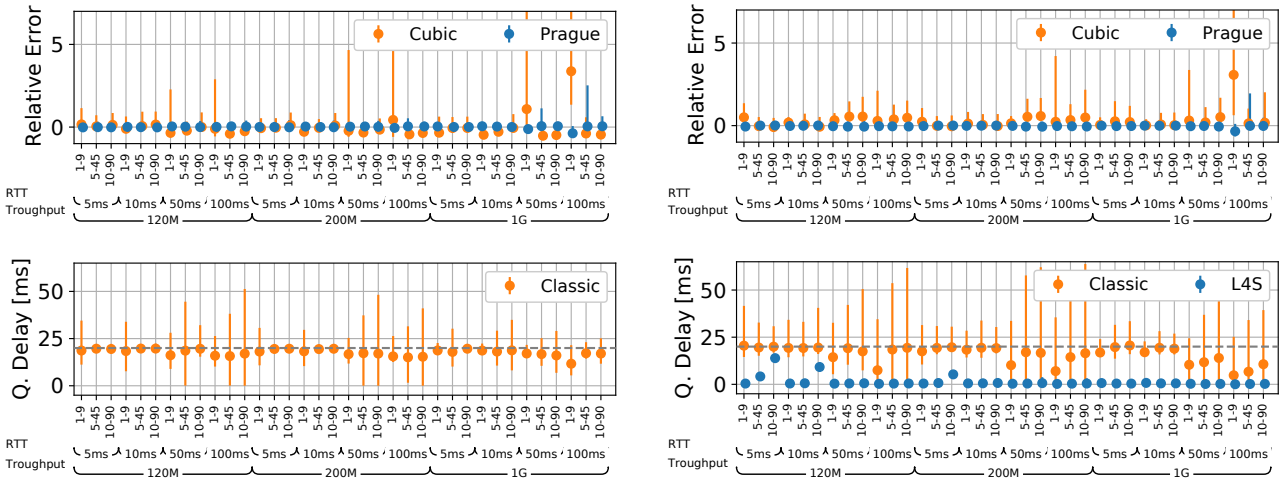


(a) PI2 **(b) DualPI2**

Fig. 7: Asymmetric case with 9:1 Classic-L4S ratio. Notations are same as in Fig. 6.

VI. Conclusion

Active Queue Management algorithms such as DualPI2 have been proven efficient in dealing with the bufferbloat problem in the presence of both scalable and non-scalable congestion controls. These methods could easily be implemented in Linux kernel, but with the advent of high capacity access links a high-speed router in the network core could also become a bottleneck. In this paper, we have investigated how PI2 and DualPI2 could be implemented on high-speed programmable switches. We have shown that it is often hard to program current RFC-standardized AQM algorithms on P4 programmable hardware targets. The proposed implementation is based on the appropriate split between control and data plane operations, employing fixed scheduling operation from the traffic manager. Our evaluations shows the correctness and performance of the proposed implementations. However, the proposed hybrid design requires continuous feedback from the control plane, and in case of high-speed networks (40-100 Gbps) it may be too slow for smooth control of the drop/marking probabilities. Our future work will include the investigation of this problem and the redesign of data plane algorithms to enable the operation of PI2 and DualPI2 at 100 Gbps bottlenecks.



(a) PI2 **(b) DualPI2**

Fig. 8: Asymmetric case with 1:9 Classic-L4S ratio. Notations are same as in Fig. 6.

ACKNOWLEDGMENT

G. Gombos and S. Laki thank the support of the "Application Domain Specific Highly Reliable IT Solutions" project that has been implemented with the support provided by the National Research, Development and Innovation Fund of Hungary and financed under the Thematic Excellence Programme TKP2020-NKA-06 (National Challenges Subprogramme) funding scheme.

REFERENCES

[1] B. Briscoe *et al.*, "Low Latency, Low Loss, Scalable Throughput (L4S) Internet Service: Architecture," Internet Engineering Task Force, Internet-Draft draft-ietf-tsvwg-l4s-arch, Oct. 2018, work in Progress. [Online]. Available: https://datatracker.ietf.org/doc/draft-ietf-tsvwg-l4s-arch

[2] K. De Schepper *et al.*, "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'16*. ACM, 2016, pp. 105–119.

[3] P. Bosshart *et al.*, "Forwarding metamorphosis: Fast programmable match-action processing in hardware for sdn," in *ACM SIGCOMM Computer Communication Review*, vol. 43, no. 4. ACM, 2013, pp. 99–110.

[4] "P4 open source programming language." [Online]. Available: https://p4.org/

[5] C. Papagianni *et al.*, "Pi2 for p4: An active queue management scheme for programmable data planes," in *Proceedings of the 15th International Conference on Emerging Networking EXperiments and Technologies*, ser. CoNEXT '19 Companion. New York, NY, USA: Association for Computing Machinery, 2019, p. 84?86.

[6] O. Albisser *et al.*, "Dualpi2 -low latency, low loss and scalable throughput (l4s) aqm."

[7] F. Hauser *et al.*, "A survey on data plane programming with p4: Fundamentals, advances, and applied research," *arXiv preprint arXiv:2101.10632*, Jan 26, 2021. [Online]. Available: https://arxiv.org/abs/2101.10632

[8] I. Kunze *et al.*, "Tofino + p4: A strong compound for aqm on high-speed networks?" in *Proceedings of the International Symposium on Integrated Network Management (IM '21)*, May 2021.

[9] K. D. Schepper *et al.*, "DualQ Coupled AQMs for Low Latency, Low Loss and Scalable Throughput (L4S)," Internet Engineering Task Force, Internet-Draft draft-ietf-tsvwg-aqm-dualq-coupled, Nov. 2018, work in Progress. [Online]. Available: https://datatracker.ietf.org/doc/draft-ietf-tsvwg-aqm-dualq-coupled

[10] A. Sivaraman *et al.*, "No silver bullet: extending sdn to the data plane," in *Proceedings of the Twelfth ACM Workshop on Hot Topics in networks*. ACM, 2013, p. 19.

[11] R. Kundel *et al.*, "P4-codel: Active queue management in programmable data planes," in *Proceedings of the IEEE 2018 Conference on Network Functions Virtualization and Sofwtare Defined Networks*. IEEE, 2018, pp. 27–29.

[12] M. Menth *et al.*, "Implementation and Evaluation of Activity-Based Congestion Management Using P4 (P4-ABC)," *Future Internet*, vol. 11, no. 7, p. 159, 2019.

[13] S. Laki *et al.*, "Towards an AQM Evaluation Testbed with P4 and DPDK," in *Proceedings of the ACM SIGCOMM 2019 Conference Posters and Demos*. New York, NY, USA: Association for Computing Machinery, 2019, p. 148–150.