Evaluating L4S Framework Performance with Programmable Data Plane Hardware

Leandro C. de Almeida[1], **Paulo Ditarso Maciel Jr.**[1], **Fábio L. Verdi**[2]

[1]Academic Unit of Informatics – Federal Institute of Paraíba (IFPB)
João Pessoa – PB – Brazil

[2]Department of Computer Science – University of São Carlos (UFSCar)
Sorocaba – SP – Brazil

{leandro.almeida,paulo.maciel}@ifpb.edu.br, verdi@ufscar.br

Abstract. *L4S, which stands for Low Latency, Low Loss, Scalable Throughput, is an initiative within the Internet Engineering Task Force (IETF) dedicated to enhancing the performance of real-time interactive applications across the Internet, like online gaming, video conferencing, and virtual reality. Its core principles revolve around implementing a novel variant of congestion control (CC) algorithm on end hosts and deploying an Active Queue Management (AQM) scheme on network nodes. Evaluating AQM algorithm adherence within the L4S framework on conventional network devices is challenging due to limited access to implementation details. However, the game changed with the introduction of data plane programmability, which facilitates the incorporation of intelligence during packet processing at the hardware's most proximate level, without the necessity for control plane intervention. In this context, this paper evaluates the implementation of two key L4S-capable algorithms, iRED and PI2, using a real hardware P4-capable switch (Tofino2). The evaluation aims to verify the adherence of these algorithms to the L4S framework and observe their coexistence with non-L4S flows. Through controlled variations in bandwidth and delay, we assess the "goodput" metric to understand under what conditions iRED and PI2 demonstrate enhanced fairness.*

1. Introduction

Over the past thirty years, the topic of Internet congestion has received widespread attention from the scientific community [Peterson et al. 2022]. Since the first paper [Jacobson 1988] published, Internet congestion has been consistently characterized as a "resource allocation" problem. In the context of this work, these "resources" can be understood such as both link bandwidth and buffer space. Based on this understanding, congestion control (CC) mechanisms were created to allocate resources fairly within the shared network infrastructure.

To achieve greater fairness in the use of shared network resources, CC mechanisms reduce the transmission of additional data packets within the network. However, depending on the intensity and attributes of the network traffic, the CC mechanisms implemented at the host level may prove insufficient in ensuring optimal network utilization. Such limitations arise when network buffers are full, forcing the router to employ a tail-drop mechanism to discard the most recently received packets. This approach

is considered suboptimal due to its propensity to induce "TCP global synchronization" [Malangadan et al. 2023]. In this case, the CC process necessitates the integration of auxiliary mechanisms to mitigate network congestion, such as Active Queue Management (AQM), a traditional mechanism employed in network device queues.

In summary, AQM is an active approach to mitigating congestion in computer networks. Instead of simply storing packets in a queue until the router is ready to transmit them, AQM actively monitors the queue and takes steps to prevent it from reaching harmful congestion levels. This is done by discarding packets before the queue reaches its maximum capacity, using algorithms such as RED (Random Early Detection) [Floyd and Jacobson 1993], to proactively identify packets and control the rate of data entering the network. By doing so, the active management of queues helps ensure a smooth traffic flow and improves overall network performance.

Recently, the IETF proposed a new framework that enables Internet applications to achieve Low queuing Latency, Low congestion Loss, and Scalable throughput control (L4S) [Briscoe et al. 2023]. The L4S architecture introduces incremental changes to both hosts and network nodes. On the host side, L4S incorporates a novel variant of a "Scalable" CC algorithm, known as TCP Prague [Briscoe et al. 2018]. It adjusts its window reduction in proportion to the extent of recently observed congestion. This stands in contrast to "classic" CC algorithms, which typically implement a worst-case scenario, reducing by half upon detecting any congestion signal. At network nodes, L4S introduces a dual-queue coupled mechanism [De Schepper et al. 2023], within AQM, wherein one queue is designated for *Classic Traffic* and another for *Scalable Traffic*. This coupled mechanism facilitates fair bandwidth utilization, ensuring harmonious coexistence between congestion control flavors. Assessing the adherence of AQM algorithms to the L4S framework using standard network equipment presents a modern hurdle though, mainly due to the limited access to implementation details typically confined to network hardware manufacturers. Nevertheless, harnessing the programmable data plane, as highlighted in [Barefoot/Intel 2021], offers a pathway to deploy these algorithms and delve into the nuances of their operation more feasibly.

In this work, we evaluate the main L4S-capable algorithms currently implemented in programmable data plane hardware, namely iRED [de Almeida et al. 2022] and PI2 [Gombos et al. 2022]. It is noteworthy that a hardware-programmable data plane offers flexibility and support for rapid prototyping and iteration. This allows customization to meet the specific needs of different applications and keep up with technological developments. Both algorithms were implemented in a local environment testbed, using a P4-capable switch Tofino2. The proposed evaluation aims to confirm the adherence of the two algorithms by the L4S framework, as well as the coexistence behavior between L4S-compliant and non-L4S flows. To do that, we collect the "goodput" metric under different network conditions, by varying bandwidth and delay in a controlled manner. The obtained results indicate in which circumstances iRED and PI2 exhibit greater fairness.

The rest of this paper is organised as follows. Section 2 presents related work. In Section 3, we review the fundamental concepts about AQM and L4S. Section 4 describes the AQM implementation in programmable data plane hardware. The proposed assessment is presented in Section 5, by detailing the setup environment and obtained results. Lastly, we present our concluding remarks and perspectives on future work in Section 6.

2. Related Work

It was challenging to find close related works, considering the novelty of evaluating AQM solutions implemented in the data plane through an L4S framework. Therefore, we discuss the literature from three different perspectives, somewhat related to the proposed work. Firstly, we highlight works that emphasize the importance of the addressed topic [BoruOljira et al. 2020]. Then, we present solutions implemented in different areas, such as cloud gaming [Ky et al. 2023] and INT [Nguyen et al. 2023]. We also include works that propose an evaluation of L4S based on Linux or simulation [Srivastava et al. 2022, Alli-Oke 2022]. Lastly, we describe the two proposals that will serve as the basis for the evaluation intended in this work [De Schepper et al. 2016, de Almeida et al. 2022]. To the best of our knowledge, this is the first work to evaluate the performance of the L4S framework with programmable data plane hardware.

In [BoruOljira et al. 2020], the authors assess the behavior of dual queue coupled AQM within the L4S architecture, stressing the importance of research reproducibility and encouraging further investigation into L4S sharing behavior. They validate experimental results on DCTCP and TCP Cubic flows using various AQM setups and explore TCP Prague performance and per-flow pacing impact on bottleneck sharing. Experimental outcomes on the coexistence of DCTCP and TCP Cubic flows using PI2 SingleQ AQM and DualPI2 DualQ AQM are presented. The network setup includes a bottleneck router with DualPI2 AQM, 1 Gbps capacity for all links, and emulated RTTs using Netem. Throughput observations reveal DCTCP flows achieving below fair share in dual queue mode, while TCP Prague consistently reaches its fair share. TCP Prague with DualPI2 AQM meets L4S Internet service design goals without causing starvation for either flow. However, L4S with DualPI2 AQM fails to provide window fairness between DCTCP and TCP Cubic, showing sensitivity to packet bursts in L4S.

The research in [Ky et al. 2023] introduces a novel system for detecting cloud gaming traffic in computer networks by combining P4 programmable hardware with unsupervised machine learning (ML). Unlike traditional software-based ML methods, which can be slow and less adaptable, this system leverages P4 hardware to accelerate parts of the ML process while still maintaining accuracy with unsupervised ML techniques. A new unsupervised ML model was developed to effectively identify cloud gaming traffic, even when encountering new services. The model is also optimized for implementation on P4 hardware. Results demonstrated that the system accurately and swiftly identifies cloud gaming traffic, suggesting its suitability for use by internet service providers. The experimental evaluation highlighted the efficacy of the P4 hardware switch for feature extraction and the robustness of the ML model in classifying cloud gaming traffic accurately.

In [Nguyen et al. 2023], the authors present a new monitoring approach for a P4-based L4S implementation, utilizing In-band Network Telemetry (INT). This framework offers real-time and fine-grained monitoring, enabling quick reaction times by delivering precise network state information through various message buses like network sockets, Kafka, or Redis. It identifies significant contributors by analyzing metrics such as timestamps and IP sources. The setup involves INT-capable P4-based L4S switches, client-server pairs generating different traffic types, and a monitoring framework. Results indicate low-overhead metric capture (e.g., queue delay, dropped packets) in real-time, align-

ing with previous studies and L4S switch expectations. Tests using TCP-Prague and TCP Cubic congestion control algorithms demonstrate bandwidth sharing and the impact of virtual environments on sending rates for both traffic patterns.

The work in [Srivastava et al. 2022] evaluates different strategies for managing congestion in networks, focusing on the challenge of balancing effectiveness and low delay in protocols that rely on delay-based CC alongside those that use loss-based CC. The authors implemented these strategies in Linux and tested them in a controlled environment using Cloudlab. They assessed performance using metrics like throughput and RTT across scenarios with varying numbers of delay-based and loss-based flows. Parameters were adjusted based on queueing delay, smoothed RTT, and backoff probability. Results underscore the difficulty of achieving effective coexistence while maintaining low delay in delay-based CC protocols when sharing a link with loss-based protocols.

The authors of [Alli-Oke 2022] explore AQM in computer networks, focusing on its performance through simulations and numerical tests. They clarify the mathematical foundations of AQM, drawing from feedback control theory, and dispel misconceptions. Using software such as MATLAB-Simulink and NS-3, they simulate various AQM schemes and compare results for accuracy. Additionally, they outline the implementation of proportional-integral-derivative (PID) controllers, including the transformation of continuous controllers into discrete ones using bilinear transformation. The article highlights considerations such as traffic balancing, variable selection, and proper packet discarding in controller-based AQM simulations. Overall, it serves as a valuable resource for understanding and analyzing AQM algorithms based on control systems, providing simulation files for further research.

Queue occupancy is crucial for AQM algorithms as it informs the probability of packet drops. The PI2 algorithm [De Schepper et al. 2016] uses queue delay per packet to determine dropping decisions in the egress pipeline. It's a linearized AQM based on the Proportional Integral (PI) algorithm, employing queue delay and PI gain factors to trigger drop policies. For classic TCP, the output probability of the PI controller is squared for drops, while for scalable TCP, it's doubled. In contrast, the iRED algorithm [de Almeida et al. 2022] operates in both ingress and egress pipelines, deciding packet drops based on average queue size and dropping probability. If iRED decides to drop, it clones the packet and sends it back to indicate congestion. The original packet is forwarded. This prevents future queue increases. Both algorithms are examined for adherence to the L4S framework.

3. Fundamental Concepts

The objective of this section is to present the conceptual foundations concerning AQM and L4S. We aim to describe how L4S-capable AQM mechanisms could potentially be integrated into programmable data plane hardware, offering insights into their implementation possibilities.

3.1. Active Queue Management

The main objective of an AQM algorithm is to effectively manage network congestion and prevent queues from reaching their maximum buffer capacity. The primary methods for conveying congestion conditions to senders include packet **marking** using Explicit Congestion Notification (ECN) bits and selective packet **dropping**.

Utilizing ECN for packet marking plays a role in congestion control by explicitly indicating to CC mechanisms that they should reduce their transmission rates. We can observe that CC mechanisms take advantage of the ECN bit marking covering DCTCP [Alizadeh et al. 2010] and, more recently, TCP Prague [Briscoe et al. 2018].

On the other hand, in the scenario of probabilistic packet discards by an AQM mechanism, CC stands to gain from the process of fast retransmission. In this instance, the sender detects the occurrence of duplicate acknowledgments (typically three) and initiates the retransmission of the packet that had been deliberately discarded by the AQM as part of its congestion mitigation strategy within the router's queues [Peterson et al. 2022].

3.2. L4S - Low Latency, Low Loss, and Scalable throughput

As briefly mentioned previously, the L4S architecture shown in Figure 1 introduces incremental changes to both the hosts' CC algorithm and the AQM at the network nodes. The modifications proposed by L4S were motivated by some requirements, such as: L4S-ECN packet identification, accurate ECN feedback, fall-back to Reno-friendly on Loss, fall-back to Reno-friendly on classic ECN bottleneck, reduce RTT dependence, scale down to the fractional window, and detecting loss in units of time [Briscoe et al. 2018].

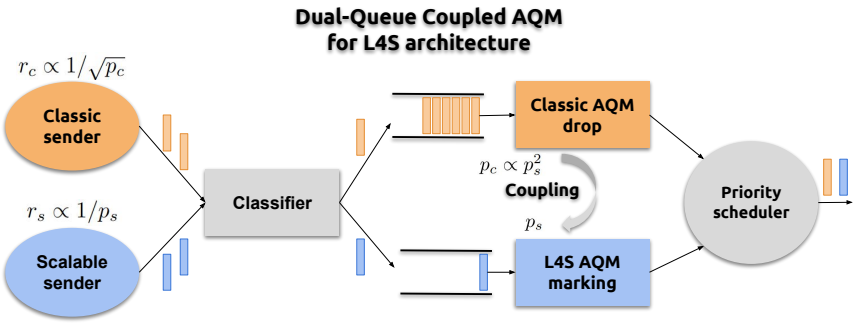


Figure 1. Dual-Queue AQM in L4S architecture. Adapted from [De Schepper et al. 2023].

In this context, L4S introduces two distributed mechanisms that work together to achieve the requirements listed above. The first of these reside in the scope of a host, being the scalable CC algorithm, the TCP Prague[1] [Briscoe et al. 2018]. It is a modified version of DCTCP [Alizadeh et al. 2010] for safe use over the Internet. As is well-known by TCP researchers, DCTCP is suitable only for data centers, where the administrator can arrange the network to work properly for frequent ECN-marking. However, this is not so simple for the public Internet, as DCTCP flows would certainly starve classical flows. To avoid this problem, TCP Prague incorporates minor adjustments from DCTCP.

The second mechanism resides in the network nodes as a Dual-Queue coupled AQM [De Schepper et al. 2023], which is responsible for maintaining a harmonious coexistence between both flavors of CC algorithms, Classic and Scalable. The Dual Queue coupled AQM mechanism, specified in the RFC9332 [De Schepper et al. 2023], was designed to solve the coexistence problem, accommodating flows into separated queues for Classic (larger delay) and Scalable (small delay) CC flavors, as can be seen in Figure 1.

[1]The name is after an ad hoc meeting of IETF in Prague in July 2015.

Despite the use of distinct queues with varying depths (shallow and deeper), bandwidth consumption remains uniform across flows. Achieving equitable resource allocation involves the interplay between the Classic and Scalable queues. This interaction enables the Classic queue to perceive the square of congestion levels in the Scalable queue. This squared level is then offset by the sending rate of the classic sender (r_c) in response to a congestion signal, characterized by $r_c \propto 1/\sqrt{p_c}$, where p_c denotes the loss level of the Classic flow. On the other hand, the Scalable sender rate (r_s) follows an inverse linear approach, characterized by $r_s \propto 1/p_s$, where p_s denotes the loss level of the Scalable flow. This linearity characterizes scalability in response to congestion.

4. AQM implemented in programmable data plane hardware

In this section, we offer a concise overview of the first L4S-capable AQM algorithms currently implemented into programmable data plane hardware, namely iRED and PI2.

4.1. iRED - ingress Random Early Detection

iRED was designed under three fundamental premises: i) Performing probabilistic packet dropping with minimal overhead; ii) Supporting and adhering to current Internet congestion control mechanisms, such as the L4S framework; iii) Being fully implemented in the data plane hardware. In the following paragraphs, we will describe the details and challenges of implementing iRED on the Tofino2 programmable switch[2].

Regarding the first premise, iRED discards packets as soon as possible to minimize overhead on the switch, i.e. in the Ingress block. However, the data (queue metadata) necessary to calculate the drop probability is available after the Traffic Manager, i.e. in the Egress block. In this context, iRED divides this operation into two parts, making it a disaggregated AQM. As can be seen in Figure 2, decisions are made at the Egress Pipeline, while actions are performed at the Ingress Pipeline.

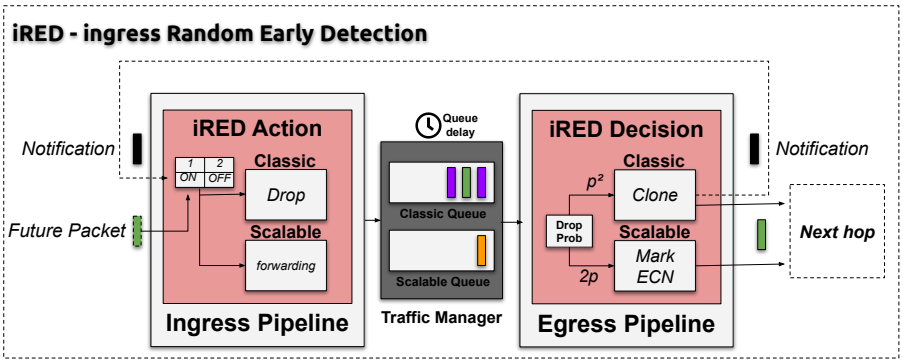


Figure 2. iRED design: disaggregating the action of a drop decision reduces wasted resources.

In alignment with the second premise, iRED provides support for the L4S framework. Initially, the classification process is performed in the Ingress block, in which the logic identifies the type of flow (L4S or non-L4S) and enqueues it to the corresponding output queue. Furthermore, the coupling mechanism is implemented in the Egress block.

[2]The previous version of iRED [de Almeida et al. 2022] was deployed in a software switch environment.

In this scenario, iRED dynamically adjusts the drop probability or marking based on the flow type (Classic or Scalable).

Finally, iRED is fully implemented in hardware, enhancing autonomy by performing AQM functions solely within the data plane, thereby eliminating the need for a control plane or external mechanisms to execute specific tasks. In this context, it is well-established that AQM logic needs the utilization of complex mathematical operations, including multiplications, divisions, and square roots. Furthermore, certain sections of the logic require the implementation of more sophisticated functions, such as exponential moving averages or similar calculations. iRED overcomes these challenges imposed by the architecture and implements the logic entirely in the data plane using available resources, such as *bitshift* to represent mathematical operations and compute the Exponentially Weighted Moving Average (EWMA).

At the Egress block, iRED computes the queue delay EWMA (or queue depth[3]) for each packet, entirely within the data plane. The inherent absence of division and floating-point operations poses challenges in calculating average values within the data plane. To overcome this constraint, as applied in [Busse-Grawitz et al. 2019], iRED employs an approximation method following the Eq. 1:

$$S_t = \alpha \cdot Y_t + (1 - \alpha) \cdot S_{t-1}, \tag{1}$$

where S_t is the updated average queue delay, S_{t-1} is the previous average queue delay, and Y_t is the current queue delay. The constant $\alpha \in [0, 1]$ determines how much the current value influences the average. iRED uses $\alpha = 0.5$, such multiplication can be replaced by bit shifts operations. The output of the EWMA will represent the average queue delay over time. If the value observed (average queue delay) is between a set of min-max thresholds defined, iRED will compute the drop probability according to the RED approach. After that, based on the coupling mechanism, it generates different congestion signal intensities (drop or marking).

Once the iRED decision module (Egress) has detected that a packet must be dropped (Classic), iRED must notify the action module (Ingress) to perform this action. The first challenge in the programmable data plane context is to achieve communication between the Ingress and Egress blocks with minimum overhead. Obviously, iRED will not drop the packet that generated the discard decision, but a future packet [Chen et al. 2019]. Discarding future packets is one of the main features differentiating iRED from other state-of-the-art AQMs. For the congestion notification to reach the Ingress block, iRED creates a congestion notification packet (cloned packet with only 48 bytes) and sends it through an internal recirculation port to reach the Ingress block.

The iRED action module, situated in the Ingress block, maintains the congestion state table on a per-port/queue basis and activates the drop flag (ON) for the corresponding port/queue. The current packet is forwarded to the next hop without introducing any additional delay. Subsequently, future packets intended for the same output port/queue, where the drop flag is set to ON, will be dropped (classic), and the drop flag will be reset to OFF. This mechanism, facilitated by iRED, ensures that the Ingress pipeline can

[3]The programmer can choose whether to use iRED's delay-based or depth-based approach.

proactively mitigate imminent queue congestion.

4.2. PI2

PI2 is a linearized AQM for both classic (TCP Cubic) and scalable (TCP Prague) flows, based on the Proportional Integral (PI) controller [De Schepper et al. 2016]. The PI2 uses queueing information (delay) per packet periodically (T interval) in conjunction with PI gain factors (α and β) to trigger the packet drop policy, as described in Equation 2.

$$p = p + \beta(\tau_{t-1} - \tau_t) + \alpha(\tau_0 - \tau_1) \tag{2}$$

Any alteration in the queue, be it an increase or decrease, is promptly rectified through the application of a proportional gain factor denoted as β. Meanwhile, any persisting deviation from the target, referred to as residual error, is gradually attenuated towards the target through the utilization of an integral gain factor, denoted as α [Gombos et al. 2022]. The output probability of the basic PI controller is squared when dropping classic TCP packets or doubled when marking scalable TCP traffic. PI2 AQM proved that by simply squaring the output PI probability, the PIE auto-tuning and several heuristics could be removed.

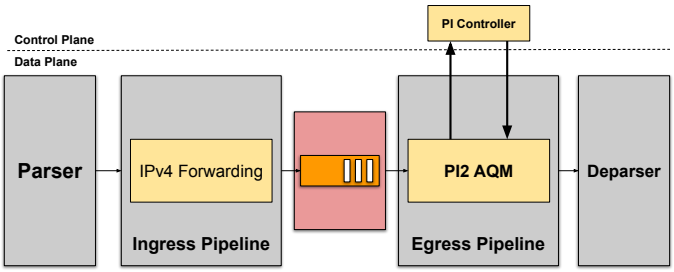


Figure 3. PI2 overview: the PI controller, positioned at the Control Plane, retrieves the queue delay from the data plane to compute the drop probability.

PI2 for P4 is an implementation for TNA [Gombos et al. 2022]. It has part of the logic implemented in a control plane, as detailed in Figure 3, to perform the required complex arithmetic operations that can not be handled by the data plane due to the restricted set of math operations in the programmable data plane architectures. The control plane periodically retrieves the queuing delay from the data plane and uses it in the PI Controller to determine the probability of drop (Classic) or mark (Scalable) [Briscoe et al. 2023].

The direct execution of complex mathematical operations within the programmable data plane remains a challenging task. As elucidated by the authors in [Gombos et al. 2022], these inherent constraints necessitated the utilization of the control plane for the implementation of the PI controller. Consequently, the principal limitation of PI2 manifests itself in its reliance on the control plane, thereby incurring an additional delay in the computation of the PI controller.

5. Evaluation

Environment description. Our testbed is based on a P4 programmable switch (Edgecore DCS810 - Tofino2). The switch connects two Linux hosts, Sender and Receiver, having

25Gbps of link capacity, as shown in Figure 4. Seeking to analyze the coexistence and fairness between different versions of TCP, each end-host sends TCP Cubic and TCP Prague flows. We conducted our experiments over the different network conditions shown in Table 1, varying bandwidth, RTT, and MTU. The bandwidth is emulated by the P4-switch using the port shaping feature. The base RTT is emulated in the Receiver by the *tc netem* tool, delaying the ACKs of TCP flows. The MTU is emulated in the end-hosts (Sender and Receiver) by the *ifconfig* tool. The *iperf* tool is used to generate traffic.

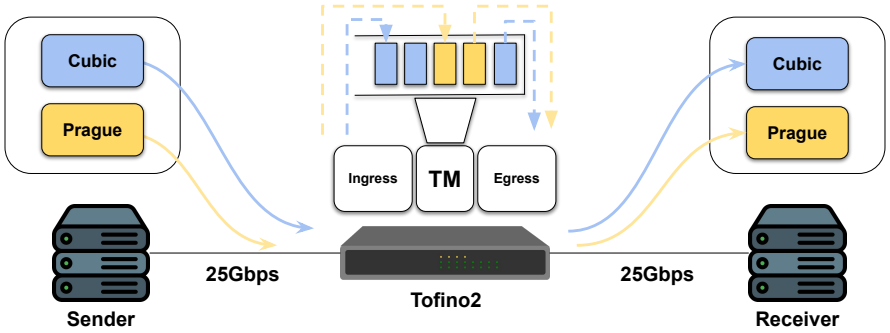


Figure 4. Evaluation setup: Cubic and Prague flows coexist in the same scenario, sharing the programmable switch bandwidth.

Table 1. Configuration parameters.

Configuration	Bandwidth(Mbps)	RTT(ms)	MTU(Bytes)
I	120	10	1500
II	120	50	1500
III	1000	10	1500
IV	1000	50	1500

Load description. The load applied to the experiment comprises 4 phases of 120 seconds each. New flows enter the system in each phase, starting from low to a high load (bottleneck condition), as used in [Gombos et al. 2022]. The number of Cubic and Prague flows are shown in Table 2.

Table 2. Load parameters.

Phase	Relative time	Cubic Flows	Prague Flows
1	0	1	1
2	120	2	2
3	240	10	10
4	360	25	25

AQMs settings. We use a base TARGET DELAY of 20ms for all AQMs. For iRED, we set the minimum and maximum thresholds for queue delay, configuring 20 (TARGET delay) and 40 ms respectively, following the rule of thumb to set the maximum threshold as at least twice the minimum [Floyd and Jacobson 1993]. For PI2, we set the

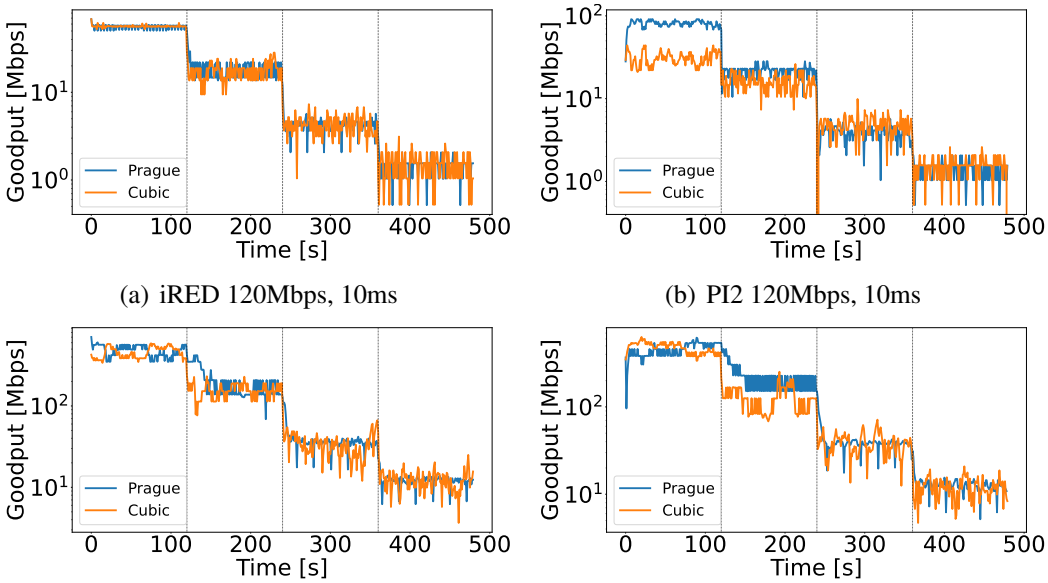
TARGET delay (20ms), INTERVAL (15ms), α (0.3125) and β (3.125), following the parameters used in [Gombos et al. 2022].

5.1. Results

In this particular experiment, our main goal is to evaluate the support and adherence to the L4S framework. The solutions evaluated were iRED [de Almeida et al. 2022] and PI2 [Gombos et al. 2022]. Additionally, we aim to evaluate the harmonious coexistence between non-L4S flows, conventionally referred to as classic (Cubic); and L4S-compliant flows, denoted as Scalable (Prague). We used the setup described in Figure 4.

In alignment with the methodology outlined by [Gombos et al. 2022], our experimental configuration involved traffic intensity loads comprising four discrete phases, each spanning a 120-second duration. Within each of these phases, we introduced new flows with specific flow pairs (1-1, 2-2, 10-10, 25-25) into the system. This sequential introduction of flows allowed us to initiate the load with lower intensity and progress toward a high-load scenario.

In the results for a 10ms baseline RTT and a bandwidth of 120 Mbps, respectively illustrated in Figure 5(a) and Figure 5(b), becomes evident that a more equitable coexistence between flows is achieved with the implementation of the iRED. Conversely, flows employing the PI2 exhibit a relative disadvantage, with improved fairness only becoming apparent in the latter half of the experiment, specifically during phases 3 and 4.



(a) iRED 120Mbps, 10ms

(b) PI2 120Mbps, 10ms

(c) iRED 1Gbps, 10ms

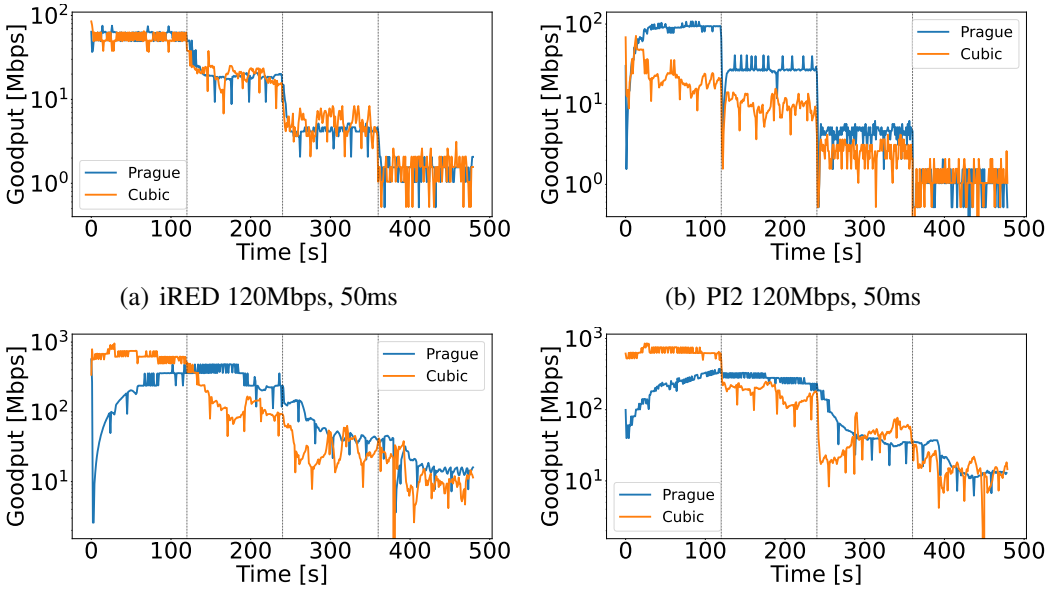
(d) PI2 1Gbps, 10ms

Figure 5. Coexistence evaluation of Cubic and Prague flows (RTT base 10ms).

When we examine the evaluation outcomes for a 1 Gbps bandwidth and a base RTT of 10 ms in Figure 5(c), it remains evident that the equitable distribution of shared bandwidth among flows persists across all phases of the experiment when utilizing iRED. In the case of PI2, despite the initial appearance of fairness in the coexistence of flows during the initial phase of the experiment shown in Figure 5(d), this equilibrium does not endure into phase 2.

The overarching conclusion drawn from our analysis of the results in Figure 5 suggests that, in the case of PI2, the intensity (i.e. the probability of marking the ECN bit) required to mark packets from the Prague flow is insufficient during the initial phases of the experiment. This deficiency in marking intensity becomes apparent due to the TCP Prague flow, as its bandwidth consumption characteristics tend to dominate and not facilitate a fair coexistence with the TCP Cubic flow. Moreover, we believe that employing a single queue to handle both scalable and classical flows presents a challenge for PI2 in achieving fairness in bandwidth allocation across all experiment phases. Furthermore, iRED's practice of not immediately discarding the packet indicating congestion but rather the subsequent packet results in a more efficient utilization of bandwidth by classical flows.

Figure 6 presents the results from scenarios in which the baseline RTT is configured to 50 ms, a value commonly encountered in long-distance networks. With a bandwidth of 120 Mbps and an RTT of 50 ms, the observed outcomes closely parallel those obtained with an RTT of 10 ms. Specifically, the iRED continues to exhibit superior fairness in the coexistence of Cubic and Prague flows, as seen in Figure 6(a), while the PI2 attains fairness only in the later stages of the experiment, as can be seen in Figure 6(b).



(a) iRED 120Mbps, 50ms

(b) PI2 120Mbps, 50ms

(c) iRED 1Gbps, 50ms

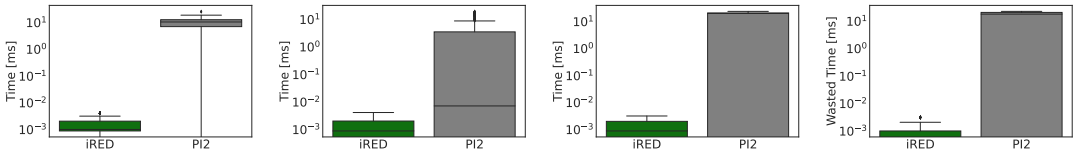
(d) PI2 1Gbps, 50ms

Figure 6. Coexistence evaluation of Cubic and Prague flows (RTT base 50ms).

However, in the case of 1 Gbps and an RTT of 50ms, the two approaches exhibited a parallel pattern of behavior, as can be seen in Figure 6(c) and Figure 6(d). There was a notable reduction in the performance of the Prague flow during the initial phase of the experiment, followed by a more equitable coexistence between flows in the subsequent three phases. In this particular scenario, we conjecture that the delayed feedback (ACK) to the Prague TCP flow resulted in a slower initial ramp-up, as it is notably more dependent on this metric [Briscoe et al. 2018]. This sensitivity likely contributed to the observed behavior where Prague TCP experienced a significant drop in performance during the initial phase of the experiment.

To comprehend the underlying factors behind the outcomes depicted in Figure 5 and Figure 6, we delved deeper into our assessment by examining the duration taken to discard a packet. In the case of PI2, this time is defined by the **queue delay computed for each discarded packet**. In other words, it means the time that a given packet stayed in the output queue before being dropped. However, in TNA, there is no intrinsic metadata to represent the queue delay. In this case, the traditional way [Gombos et al. 2022] to do it is to compute the difference between *egress global timestamp (egTstmp)* and *ingress global timestamp (igTstmp)*. This difference represents the sum of the time spent in: Ingress parser latency; Ingress processing latency; Ingress deparser latency; and Traffic Manager latency. We create an internal bridge header to carry the *igTstmp* from Ingress to Egress, and when the packet reaches the Egress block, we get the *egTstmp* to calculate the queue delay. In the iRED case, the discarded packets are not sent to the output queue, so **the queue delay is always zero**. However, the congestion notification needs to be carried to the Ingress block, to which iRED uses recirculation. In this case, this time is defined by the **recirculation time for each notification packet sent from Egress to the Ingress**.

Figure 7 depicts these time intervals on a millisecond scale through a boxplot graph. It includes the quartiles q_1, q_2, and q_3, as well as the lower (l_i) and upper (l_s) limits. Additionally, outliers are marked where applicable. Overall, it is noticeable that across all scenarios analyzed in this study, the time taken to drop a packet was consistently shorter in iRED compared to PI2. Such difference arises from iRED being a disaggregated AQM, wherein the decision-making process is distinct from the action taken. This characteristic ensures that dropped packets do not occupy bandwidth within the programmable switch. Conversely, PI2 combines both decision-making and packet discarding exclusively in the egress block, leading to bandwidth inefficiencies within the switch pipeline.



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

Figure 7. Drop delay for each approach: recirculation time for iRED and queue delay for PI2.

Table 3 outlines the average percentage disparity in bandwidth consumption between Prague and Cubic flows. Here, the smaller the difference, the fairer the bandwidth division will be and, consequently, the coexistence of flows will be more harmonious. It's evident that iRED exhibits a variance between 0.03% (best case) - 6.47% (worst case), underscoring fairness in bandwidth distribution. Conversely, PI2 exhibits a difference between 2.06% (best case) - 25.0% (worst case), demonstrating a lack of fairness in the division of bandwidth.

6. Conclusions

In this study we evaluate two main L4S-capable algorithms, iRED and PI2, implemented in programmable data plane hardware, specifically utilizing a P4-capable switch Tofino2. We highlight that the flexibility of hardware-programmable data planes allows for customization to meet various application needs and adapt to technological advancements

Table 3. Average percentage difference for bandwidth consumption for Prague and Cubic flows.

Configuration	Algorithm	Percentual Difference
I	iRED	0.03%
I	PI2	17.06%
II	iRED	0.7%
II	PI2	25.0%
III	iRED	2.59%
III	PI2	2.76%
IV	iRED	6.47%
IV	PI2	10.81%

quickly. The evaluation conducted in a local testbed environment aims to confirm the adherence of these algorithms to the L4S framework and assess their coexistence with non-L4S flows. It focuses on the "goodput" metric under varying network conditions, such as bandwidth and delay. Results provide insights into the fairness exhibited by iRED and PI2 under different circumstances. Our findings indicate that iRED demonstrates superior fairness in bandwidth allocation between flows compared to PI2. Future work may include investigating the performance of iRED and PI2 in larger-scale network environments with diverse traffic patterns. Additionally, exploring the adaptability of these algorithms to evolving network technologies and optimizing them for specific application requirements could be valuable. Further research could also examine their interaction with other congestion control mechanisms beyond traditional L4S frameworks.

References

Alizadeh, M., Greenberg, A., Maltz, D. A., Padhye, J., Patel, P., Prabhakar, B., Sengupta, S., and Sridharan, M. (2010). Data Center TCP (DCTCP). In *Proceedings of the ACM SIGCOMM 2010 Conference*, SIGCOMM '10, page 63–74, New York, NY, USA. Association for Computing Machinery.

Alli-Oke, R. O. (2022). On the validity of numerical simulations for control-theoretic AQM schemes in computer networks. *Mathematics and Computers in Simulation*, 193:466–480.

Barefoot/Intel (2021). P416 Intel Tofino™ Native Architecture – Public Version.

BoruOljira, D., Grinnemo, K.-J., Brunstrom, A., and Taheri, J. (2020). Validating the Sharing Behavior and Latency Characteristics of the L4S Architecture. *SIGCOMM Comput. Commun. Rev.*, 50(2):37–44.

Briscoe, B., De Schepper, K., Bagnulo, M., and White, G. (2023). RFC 9330: Low Latency, Low Loss, and Scalable Throughput (L4S) Internet Service: Architecture.

Briscoe, B., Schepper, K. D., Albisser, O., Tilmans, O., Kuhn, N., Fairhurst, G., Scheffenegger, R., Abrahamsson, M., Johansson, I., Balasubramanian, P., Pullen, D., Bracha, G., Morton, J., and Täht, D. (2018). Implementing the ' Prague Requirements ' for Low Latency Low Loss Scalable Throughput (L 4 S). In *Netdev 0x13, THE Technical Conference on Linux Networking*.

Busse-Grawitz, C., Meier, R., Dietmüller, A., Bühler, T., and Vanbever, L. (2019). pForest: In-Network Inference with Random Forests. *CoRR*, abs/1909.05680.

Chen, X., Feibish, S. L., Koral, Y., Rexford, J., Rottenstreich, O., Monetti, S. A., and Wang, T.-Y. (2019). Fine-Grained Queue Measurement in the Data Plane. In *Proceedings of the 15th International Conference on Emerging Networking Experiments And Technologies*, CoNEXT '19, page 15–29, New York, NY, USA. Association for Computing Machinery.

de Almeida, L. C., Matos, G., Pasquini, R., Papagianni, C., and Verdi, F. L. (2022). iRED: Improving the DASH QoS by dropping packets in programmable data planes. In *2022 18th International Conference on Network and Service Management (CNSM)*, pages 136–144.

De Schepper, K., Bondarenko, O., Tsang, I.-J., and Briscoe, B. (2016). PI2: 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, page 105–119, New York, NY, USA. Association for Computing Machinery.

De Schepper, K., Briscoe, B., and White, G. (2023). Dual-Queue Coupled Active Queue Management (AQM) for Low Latency, Low Loss, and Scalable Throughput (L4S). RFC 9332.

Floyd, S. and Jacobson, V. (1993). Random early detection gateways for congestion avoidance. *IEEE/ACM Transactions on Networking*, 1(4):397–413.

Gombos, G., Mouw, M., Laki, S., Papagianni, C., and De Schepper, K. (2022). Active Queue Management on the Tofino programmable switch: The (Dual)PI2 case. In *ICC 2022 - IEEE International Conference on Communications*, pages 1685–1691.

Jacobson, V. (1988). Congestion Avoidance and Control. In *Symposium Proceedings on Communications Architectures and Protocols*, SIGCOMM '88, page 314–329, New York, NY, USA. Association for Computing Machinery.

Ky, J. R., Graff, P., Mathieu, B., and Cholez, T. (2023). A Hybrid P4/NFV Architecture for Cloud Gaming Traffic Detection with Unsupervised ML. In *2023 IEEE Symposium on Computers and Communications (ISCC)*, pages 733–738.

Malangadan, N., Raina, G., and Ghosh, D. (2023). Synchronisation in TCP networks with Drop-Tail Queues. In *Networking and Internet Architecture (cs.NI); Dynamical Systems (math.DS)*. arXiv.

Nguyen, H. N., Mathieu, B., Letourneau, M., Doyen, G., Tuffin, S., and Oca, E. M. d. (2023). A Comprehensive P4-based Monitoring Framework for L4S leveraging In-band Network Telemetry. In *NOMS 2023-2023 IEEE/IFIP Network Operations and Management Symposium*, pages 1–6.

Peterson, L., Brakmo, L., and Davie, B. (2022). *TCP Congestion Control: A Systems Approach*. Systems Approach.

Srivastava, A., Fund, F., and Panwar, S. S. (2022). Coexistence of delay-based TCP congestion control: Challenges and opportunities. In *2022 IEEE International Workshop Technical Committee on Communications Quality and Reliability (CQR)*, pages 43–48.