P4BS: Leveraging Passive Measurements from P4 Switches to Dynamically Modify a Router's Buffer Size

Elie Kfoury*, Jorge Crichigno*, Elias Bou-Harb†
*Integrated Information Technology, University of South Carolina, Columbia, U.S.A
†Computer Science Department, Louisiana State University, Baton Rouge, U.S.A
Email: ekfoury@email.sc.edu, jcrichigno@cec.sc.edu, ebouharb@lsu.edu

***Abstract*—The performance of networked applications can be dramatically impacted by the size of the buffer at the bottleneck router. Shallow buffers may increase packet losses and decrease link utilization, while deep buffers may increase the queueing delays for latency-sensitive flows. Operators nowadays configure large buffers statically without considering the characteristics of flows or dynamic traffic patterns. This paper presents P4BS, a system that dynamically modifies the buffer size of a legacy router. P4BS leverages programmable switches as passive instruments to measure various metrics that are vital when deciding on buffer size. The measured metrics include the number of long-lived flows and their round-trip times, the packet loss rates, and the queueing delays. Using these measurements, the programmable switch sequentially searches for a buffer size that minimizes the queueing delays and the packet loss rates. The system was implemented on a Tofino hardware switch and the system was tested on a wide range of network scenarios. The results show improvements in the quality of service of various applications including web browsing, video streaming, and voice over IP.**

***Index Terms*—Buffer sizing, P4, Measurements, Bayesian Optimization, Passive Deployment.**

I. Introduction

Packet-switched devices (*e.g.*, routers, switches) are equipped with buffers that help in accommodating transient bursts, absorbing traffic fluctuations, reducing packet drop rates, and ensuring high link utilization [1]. The size of a buffer significantly impacts the performance of network applications [2–4]. Despite the continuous research efforts on buffer sizing throughout the years [5–9], there is still no convention on "*How big should a buffer be?*". One might think that large buffers are favorable to improving the performance of the network. This intuition is incorrect; deep buffers incur unnecessary delays as packets have to wait in a long queue. In contrast, short buffers might decrease link utilization and increase packet drop rates.

For a long time, buffers were sized based on the general rule-of-thumb [5] which recommends a size equal to at least $C \cdot RTT_{min}$, where C is the link capacity and RTT_{min} is the average minimum Round-Trip Time (RTT). The quantity $C \cdot RTT_{min}$ is also referred to as the Bandwidth-delay Product (BDP). In 2004, a seminal paper by Appenzeller et al. [6] demonstrated that the buffer size can be significantly reduced to $\frac{C \cdot RTT_{min}}{\sqrt{N}}$, where N is the number of large flows that are persistent over time. This buffer size rule (known as the Stanford rule) was proposed based on the dynamics of the Additive-Increase Multiplicative-Decrease (AIMD) control law used by early versions of TCP (*e.g.*, NewReno). Since then, numerous works criticized and proposed variations to this rule [10–13].

A recent study [14] analyzed the Alexa Top 20,000 websites to infer which Congestion Control Algorithms (CCAs) are being used. The findings indicate a significant shift in the representation of CCAs over the years. Back in 2001, NewReno (loss-based variant) accounted for 100% of the traffic, but in 2019, it only constituted 1% of the traffic. Cubic, which is the dominant TCP variant, represented 42% of the traffic in 2019. Despite also being a loss-based variant, Cubic necessitates a distinct buffer size compared to NewReno [15]. Another noteworthy development is the emergence of BBR [16], which constitutes 33% of the traffic in 2019. The prevailing buffer sizing rules were designed under the assumption that flows would primarily use NewReno. Consequently, these rules may not align optimally with Cubic's or BBR's requirements. The diversity of CCAs on the Internet and their distributions resuscitate the problem of buffer sizing [9].

Specifying a static buffer size might not yield optimal performance for two main reasons. First, distinct CCAs require different buffer sizes; a recent study [15] analyzed the buffer requirements for various CCAs under different network conditions, and the reported buffer sizes vary significantly. Second, metrics used for buffer sizing such as the RTT and the number of large flows, frequently change over time. The other alternative to static buffer sizing is to leverage live measurements to dynamically modify the buffer size in a closed-loop. The advantage of this approach is that the buffer is changing according to network conditions, and little knowledge is required about the dynamics of CCAs traversing the bottleneck link. The drawback, however, is that the metrics are very hard to estimate in contemporary deployments [17], especially with high traffic rates. Another drawback is the lack of availability of such techniques in today's routers and switches.

In the past few years, programmable switches have emerged as a promising approach to customize the data plane behavior and to accelerate a wide range of applications (*e.g.*, congestion control [18], telemetry [19], load balancing [20], caching [21], machine learning training [22], and many others [23]). Due to their high precision, low cost, and compute power, recent work [24], [25] has investigated using these switches as instruments to process traffic measurements at terabits per second rates.

A. Contributions

This paper proposes P4 Buffer Sizing (P4BS), a cost-efficient scheme that dynamically modifies a legacy router's buffer size after measuring the current network conditions. The traffic is analyzed by tapping on the router's ports and forwarding the traffic to a programmable switch. The programmable switch then calculates various metrics that affect the buffer size such as the number of long flows, the RTTs, the packet loss rates, and the queueing delays, and then derives a buffer size by solving an optimization problem. The calculated buffer size is then configured on the legacy router. The contributions can be framed as follows:

- Devising a dynamic buffer size allocation scheme that relies on programmable switches as passive measurements.
- Identifying large flows, tracking their counts, measuring their RTTs, calculating the packet loss rates, and the queueing delays, entirely in the data plane.
- Deriving a buffer size using Bayesian Optimization, a sequential design strategy used to solve blackbox functions. The derived buffer size minimizes the queueing delays and the packet losses, and maximizes the utilization of the link.
- Improving the quality of service of various user-centric applications including web browsing, Voice over IP (VoIP), and HTTP-based video streaming.

B. Paper roadmap

The paper is organized as follows. Section II provides a background on the existing buffer sizing techniques and motivates the need for P4BS. Section III describes an overview of the proposed system and the design goals. Section IV explains how P4BS calculates and tracks the metrics in both the data and the control planes of the switch. Section V formulates and solves an optimization problem to derive a final buffer size. Section VI discusses the results obtained from experiments conducted on physical routers and switches. Section VII provides some discussions and challenges faced when implementing the system. Finally, Section VIII concludes the paper and highlights potential future work.

II. Background and Motivation

This section provides a short background on different buffer sizing regimes proposed in the literature. Moreover, it provides discussions that motivate the need for the proposed system.

A. Static buffer sizing

The different static buffer size recommendations in the literature assume that a router is carrying long-lived TCP flows that adhere to the AIMD control law.

Rule-of-thumb (BDP). For a long time, buffers were sized based on the general rule-of-thumb [5] which recommends a size equal to at least $C \cdot RTT_{min}$, where C is the link capacity and RTT_{min} is the average minimum RTT. This quantity assumes a single long-lived TCP flow (or many synchronized flows experiencing losses within the same RTT) going through the bottleneck link.

Small buffer rules. In a realistic environment, the bottleneck link may be shared by many flows with different RTTs. The large number of flows and the variation in their RTTs cause desynchronization. In such case, the buffer can be significantly reduced to $\frac{C \cdot RTT_{min}}{\sqrt{N}}$, where N is the number of large flows that are persistent over time [6]. Another rule that recommends even smaller sizes known as the "*Tiny buffer rule*" [26] suggests that the buffer can be further reduced to $B \geq \mathcal{O}(\log W)$, where W is the average window size. In this rule, the buffer size is as small as 20-50 packets, but at the cost of a 10-20% drop in link utilization. This model assumes that flows are not synchronized and the traffic is not bursty (*e.g.*, when *pacing* is enabled on end-hosts stacks, or when bursts are absorbed by core routers which operate at a higher speed than access routers).

Large buffers. According to [9], router vendors recommend larger buffers (up to 200ms of line rate), and operators typically use such buffers to mitigate packet losses and to meet Service-Level Agreements (SLAs). However, with such buffers, flows will suffer from excessive queueing delays.

Discussions. The static rules consider link utilization as the sole constraint to satisfy. This metric is operator-centric (*i.e.*, it makes sure that the network resources are fully utilized) but not necessarily user-centric. The packet loss rate increases with smaller buffers. In such cases, some flows might experience TCP timeouts, and the per-flow throughput of the various competing flows significantly varies. It was shown in [7] that smaller buffers can result in unacceptable loss rates (up to 15%). Furthermore, high loss rates can degrade the performance of interactive media flows. Other buffer sizing rules such as BSCL [13] consider loss rates and queueing delays, but require many parameters to estimate, which is not feasible in contemporary routers.

Active Queue Management (AQMs) Active Queue Management (AQMs) such as CoDel [27], PI2 [28], and dualPI2[29], offer a complementary approach to static buffers. AQMs can monitor the minimum queueing delay and drop packets strategically to maintain low latency. This approach prevents excessive buildup and congestion in the network, reducing the impact of the challenges associated with buffer sizing.

B. Dynamic buffer sizing

The idea of dynamically modifying the buffer started with Flow Proportional Queuing (FPQ) [30]. FPQ adjusts the amount of buffering in proportion to the number of TCP flows. The paper proved that the loss rate increases with the square of the number of flows. The method used the Mathis equation [31] ($w = \frac{0.87}{\sqrt{l}}$, where w is the average window size, and l is the loss rate) to calculate a target queue and a target loss. Both targets require estimating the number of flows, a function that is implemented in FPQ. They also accept the BDP as input. FPQ does not estimate the BDP, but requires operators to hardcode it according to typical RTTs seen in their networks.

Stanojević *et al.* [32] proposed ADT, a method that regulates the queue to a minimum size so that the queueing delay is

TABLE I
COMPARISON OF VARIOUS BUFFER SIZING RULES PROPOSED IN THE LITERATURE WITH THE PROPOSED WORK (P4BS).

Rule		Constraint			Deployability		Thresh	
Mode	**Name**	ρ	d	p	**CC-A**	**CR**	$\hat{d}$	$\hat{p}$
Static	BDP [5]	✓	×	×	×	✓	×	×
	Stanford [6]	✓	×	×	×	✓	×	×
	Tiny [26]	✓	×	×	×	✓	×	×
	BSCL [13]	✓	✓	✓	×	×	✓	✓
Dynamic	FPQ [30]	✓	✓	×	×	×	×	×
	ADT [32]	✓	×	×	×	×	×	×
	ABS [33]	✓	×	✓	✓	×	✓	✓
	N/A [34]	✓	×	×	×	✓	×	×
	P4BS	✓	✓	✓	✓	✓	✓	✓

ρ: Link utilization, d: queueing delay, p: packet loss, CC-A: congestion control-agnostic, CR: current routers, $\hat{d}$: delay bound, $\hat{p}$: loss bound

reduced, while maintaining a 98% link utilization. The method assumes that the network conditions do not vary frequently, and hence, doubles and halves the MIMD parameter (a constant that is multiplied by the current queue size).

Zhang and Loguinov [33] presented ABS, a mechanism that leverages the monotonic relationship between the buffer size and the link utilization, loss rate, and queueing delay, to adjust the buffer size. ABS uses integral controllers to modify the buffer size and gradient-based components to train its parameters.

Discussions. The dynamic buffer sizing techniques are interesting because they are responsive to changing network conditions. They can provide better performance than the static ones since they reap the rewards of smaller buffers, but at the same time, enlarge the buffer when needed [9].

The main issue with the existing dynamic buffer sizing schemes is their lack of availability in contemporary routers. In fact, although proposed more than ten years ago, none of the existing solutions are implemented in today's routers/switches. While it is feasible to implement simplistic schemes (*e.g.*, FPQ and ADT) in the router, their performance limitations are not worth the change in the hardware. More complicated schemes (*e.g.*, ABS) are specifically designed for traditional CCAs with low traffic rates. Hardcoding such an algorithm in the router not only consumes resources but may get obsolete in a short amount of time. Moreover, all of the existing algorithms were tested in simulations, but none have been tested on real equipment with realistic traffic. Another issue is that they only consider loss-based AIMD congestion control. However, emerging congestion control algorithms such as BBR have completely different dynamics [35], and therefore, cannot be used with existing dynamic buffer sizing schemes.

C. Novelty

Table I highlights the differences between P4BS and the existing buffer sizing techniques. Unlike the existing techniques, P4BS can be easily deployed and tested on contemporary routers. It is agnostic to the dynamics of the CCA, considers packet loss and queueing delays when deriving the buffer size, and can be configured with upper bounds on the loss rates and queueing delays whenever possible.

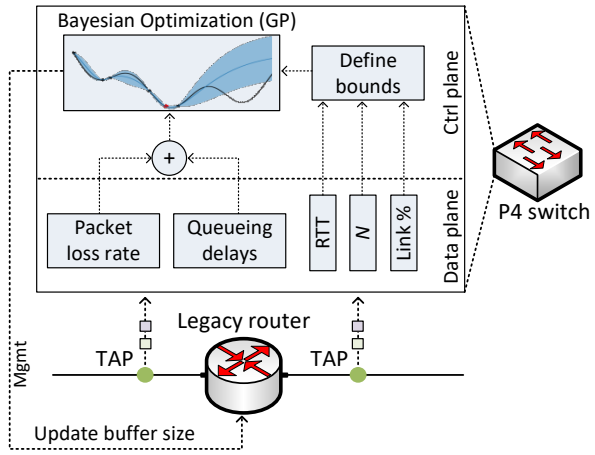

Fig. 1. Proposed system architecture.

III. PROPOSED SYSTEM

This section describes the design goals, scope, and architecture of the proposed system.

A. Design goals

The primary design goals of P4BS are summarized as follows:

Dynamic adaptation to heterogeneous traffic. Traffic traversing networks today is heterogeneous in many aspects. Flows can have different CCAs, RTTs, and/or (in)activity times. For instance, in a small campus network, there is an order of magnitude difference in activity time between weekdays and weekends [36]. When selecting a buffer size, all these dynamic factors must be considered.

SLA-compliance. Service Level Agreements (SLAs) specify performance guarantees that an ISP provides to its customers. It was shown in [37] that while the majority of the buffer sizing rules achieve high throughput and link utilization, other metrics such as packet loss rates and queueing delays are often sacrificed. From an SLA perspective, neither high loss rates nor long queueing delays are acceptable. Thus, limits should be imposed to define the tolerable quantities of such metrics whenever possible.

Smooth integration in existing networks. None of the existing dynamic buffer sizing methods are implemented or tested in current networks. These methods require modifying the proprietary router to keep track of various statistics that are relevant when deciding on the buffer size. The dynamic buffer sizing algorithm should be easily and cheaply integrated into any type of network with proprietary routers.

Extensibility. The traffic profile on the Internet keeps changing over time; new CCAs, higher link speeds, etc. Moreover, switches/routers with small buffers mainly used in data centers are emerging. It is important for a system to accommodate and be adaptable to potential changes in the future.

B. Overview

Consider Fig. 1 which demonstrates the high-level architecture of P4BS. The system continuously and passively

monitors the traffic traversing a legacy router to derive a buffer size. Passive TAP devices are installed on the links of the router, and the traffic is forwarded to a customized packet processing pipeline in the P4 programmable switch that operates at line rate. In the data plane of the programmable switch, metrics such as the RTT, the packet loss rate, the link utilization, and the average queueing delay are computed. Additionally, the number of large flows (N) (*i.e.*, flows that constitute a large portion of the traffic) is determined. Note that quantifying the aforementioned metrics is not possible with general-purpose servers since CPUs often rely on sampling and polling techniques which are only suitable for coarse-grained measurements. The measured metrics are then pushed to the control plane of the programmable switch. The control plane uses Bayesian Optimization (BO), a sequential search strategy, to derive a buffer size in a closed-loop. The goal is to minimize queueing delays and packet losses while maintaining high link utilization.

C. The need for programmable switches

The selection of the buffer size is determined based on metrics such as the number of flows, the flows' RTTs, the loss rate, the queueing delay, etc. There is a need to measure these metrics on-the-fly when devising a dynamic buffer sizing algorithm. Estimating these metrics without a P4 switch is not feasible:

Estimating the number of flows. Spang and McKeown [17] outlined the complications that arise when estimating the number of flows in contemporary networks. Specifically, collecting and analyzing traffic at high rates is not feasible on a general-purpose CPU. Other techniques like packet sampling can lead to errors in the estimations. Ideally, each packet traversing the network is inspected to identify the number of flows. Programmable switches are specifically designed to enable programmers to devise custom algorithms that execute on each packet.

Estimating the RTT. Operators today use active measurements tools [38–41] to measure the RTT [25]. Such tools do not produce RTT samples on the actual traffic; RTT samples are generated for the probes. Besides their inaccuracy, probes can take different paths, leading to incorrect RTT samples. On the other hand, passive tools are only used to measure RTT samples from the three-way TCP handshake [42, 43]. Such measurements cannot capture the delay changes in long-lived flows and can produce biased samples. P4 switches can precisely measure the RTT throughout the lifetime of the flows. This is possible due to their capability of processing each packet entering the switch, and due to the high precision timer provided by the hardware.

Estimating the loss rate and the queue delay. The proposed system creates a closed control loop where the packet loss rate and the queue occupancy measurements are used to calculate a new buffer size. Estimating such metrics in today's network is challenging. Consider Fig. 2 (a) which shows the estimated packet loss on the P4 switch versus the loss obtained through polling from hardware using the Simple Network Management

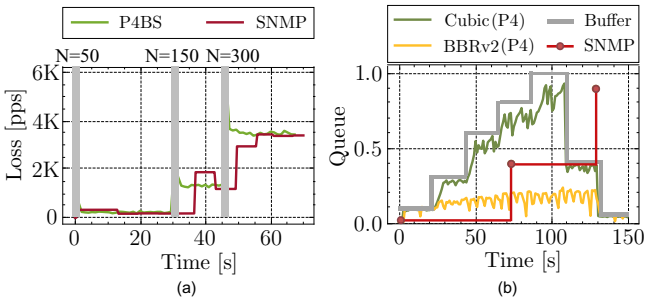

Fig. 2. (a) Estimated loss in P4BS vs. polling (SNMP). (b) Estimated queue occupancy vs SNMP with various CCAs.

Protocol (SNMP). The vertical gray stripes denote newly joined flows. It can be seen that polling is at least five seconds behind whenever the network conditions change (*e.g.*, new flows join, the buffer was modified, etc.). This is problematic for the buffer sizing problem as the algorithm must wait for at least five seconds after each buffer size modification. The loss rate can also be approximated using the Mathis equation; however, this approximation assumes the CCA of the flow to be loss-based.

Fig. 2 (b) shows the estimated queue occupancy in P4BS and SNMP. SNMP only produced two samples throughout the test which lasted 150 seconds. The samples are approximately produced each minute. Note how the SNMP measurements are at least 30 seconds behind, which is not acceptable for a dynamic buffer sizing algorithm.

Fig. 2 (b) also shows flows with different CCAs (Cubic and BBRv2). It also shows the value of the configured buffer size at any given time. Note how Cubic attempts to fill the buffer due to its AIMD dynamics. BBRv2 on the other hand tries to minimize queuing delays, and thus, does not fill the queue. The different behaviors of CCAs motivated the need to measure the queueing delay rather than assuming that the current queueing delay is equal to the configured buffer size.

Buffer tuning timescales. A primary objective is to reach an optimum buffer size as quickly as possible in order to cope with the ever-changing activity times in a busy network. The coarse measurements obtained through polling/sampling (tens of seconds or even minutes) are not only inaccurate, but also become stale. P4BS modifies the buffer size and retrieves statistics at much faster timescales due to the capability of a programmable switch to process packets at a line rate.

D. The need for TAPs

The first step consists of forwarding the traffic traversing a router to a programmable switch for custom packet processing. It is essential to preserve the traffic characteristics when replicating packets. For instance, computing the RTT requires associating packets' acknowledgment numbers with their corresponding sequence numbers. Methods such as port mirroring are inaccurate and cannot be used here. Port mirroring may modify the timing information of traffic, introduce random packet drops, and mirror traffic in an out-of-order manner, even when the switch is experiencing minimal usage [44].

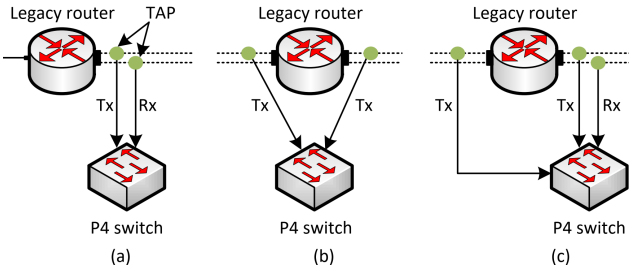

Fig. 3. TAPs positioning. (a) TAPs placed on the egress link in both directions. (b) TAPs placed on the ingress and egress links, unidirectional. (c) Combination of placements (a) and (b).

In contrast, network TAPs are hardware devices that provide an exact duplicate of network traffic passively, even when the network is saturated. Network TAPs operate at the physical layer, and hence require zero changes in the configuration of the upper layers (*e.g.*, IP addressing). It is also worth noting that TAPs are relatively cheap devices, making the whole system cost-efficient.

TAPs positioning. The position of the TAPs in the network is based on the metrics of interest. In Fig. 3 (a), two TAPs are placed on the link of the router, each in a different direction. This placement enables calculating the round-trip time (*e.g.*, [25]), while excluding the queueing delays. The other positioning depicted in Fig. 3 (b) places TAPs at the ingress and egress links of the router, in a unidirectional way. Such placement allows monitoring metrics such as the queueing delay and the number of lost packets due to buffer overflow. P4BS uses the placement of Fig. 3 (c), such that the RTT, the packet loss resulting from buffer overflow, and the queueing delays are calculated.

IV. Metrics Computation

The system computes the buffer size by considering the following metrics: the average RTT, the packet loss rate, the number of large flows, and the average queueing delay. The subsequent subsections describe how each of those metrics is computed.

A. Round-trip time

P4BS adopts the method presented in [25] to calculate the RTT of individual flows. This method relates the sequence numbers (SEQ) of TCP packets with their acknowledgments (ACKs) and uses the precise timing information of the switch to produce an RTT sample (the difference between the timestamps of two packets). The flow identifier (FID) of an outgoing packet is derived by hashing the 5-tuple (source/destination IP, source/destination port, and protocol), and the expected ACK (eACK) is calculated by adding the SEQ number to the length of the payload. The timestamp of the current packet is stored in a register array indexed by the FID and the eACK. Upon receiving an incoming TCP packet, the FID and the ACK number are used to fetch the table for an existing record. If there is a match, the timestamps are subtracted, and an RTT sample is produced. The method uses a timeout threshold to evict the idle records. It also uses multi-stage hash due to the constraints on accessing the data plane memory.

B. Packet loss rate

P4BS also computes the packet loss rate. This metric is important since it strongly affects the selection of buffer size and directly impacts the performance of the network. In addition, it can violate SLAs if it is not controlled properly.

The packet loss rate can be computed in two different methods. The first method checks for packet retransmissions which are produced by the TCP sender on the occurrence of loss events. Because TCP is a reliable transport, all segments lost in the network should be repaired with retransmissions from the sender. A packet is considered retransmitted if its SEQ is lower than the expected next sequence number (eSEQ). Note that eSEQ is the same as the eACK, which was previously computed during the RTT calculation; thus, reusing the same value will save memory on the switch. Additionally, the packet must not be a keepalive packet and its segment length must be greater than zero. After counting the number of retransmitted packets and the total number of packets in the data plane, the control plane can compute the packet retransmission rate. Note that the retransmission rate is only an estimation of the packet loss rate due to factors such as spurious retransmissions or the presence of non-TCP flows sharing the queue. While this method counts the losses that occurred in the whole path, it is easy to consider only those that are dropped due to buffer overflow by using a simple heuristic (*e.g.*, counting only when the queue occupancy reaches the configured buffer). Alternatively, the loss rate can be computed when two copies of each packet (before and after the buffer) are sent to the programmable switch (placement (b) in Fig. 3). Empirically, the first method produced more accurate and consistent measurements than the second. This is because by the time the counts are polled in method 2, outstanding packets in the queue are not counted.

C. Large flows count

Appenzeller et al. [6] proved that short flows (TCP and non-TCP) have a much smaller effect on the buffer size than long-lived TCP flows. Short flows are those that send a few packets and never reach their equilibrium sending rate. When mixed with long-lived flows, short flows will be dominated, and the buffer size can be determined by solely relying on the number of long-lived flows. Therefore, it would make sense to only estimate the number of long-lived flows when deciding on buffer size. There are many schemes that estimate the number of flows in the data plane. Due to the limited memory in the switching hardware, existing schemes (*e.g.*, [45, 46]) largely rely on probabilistic data structures (*e.g.*, sketch, bloom filter, HyperLogLog algorithm, etc.), sampling methods, and top-k counting.

P4BS uses the Count-Min Sketch (CMS) data structure [47] to estimate the number of packets for a given flow. The counts of packets are then compared against a threshold to decide if a flow should be classified as a long flow. Consider Fig. 4 (a). The data structure is constructed using d register arrays that

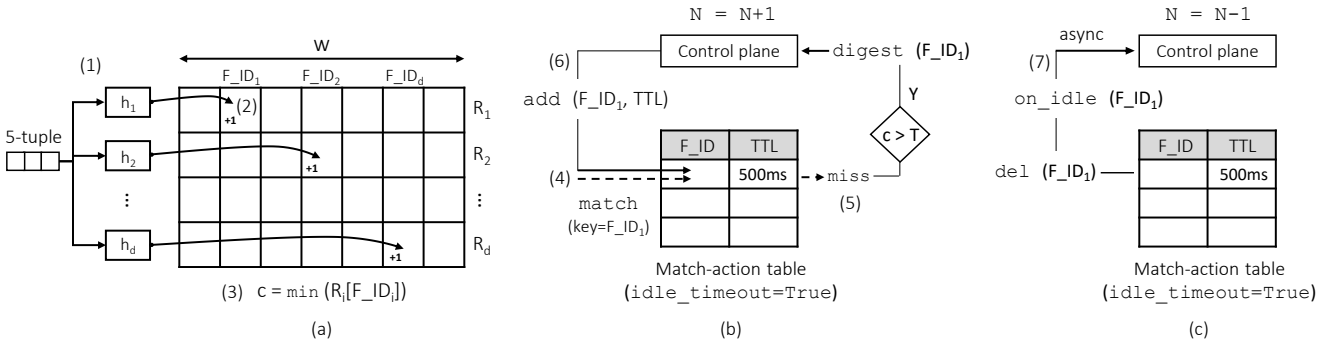

Fig. 4. Long flows count estimation. (a) the Count-Min Sketch (CMS) is used to store the counts of the flows, and the minimum of these counts is calculated; (b) if there is a `miss` on the flow identifier and the calculated count exceeds a predefined threshold, the data plane informs the control plane through a digest about the new long flow. The control plane increments the count of long flows (N) and inserts an entry indexed by the flow ID in a table that implements the idle timeout mechanism; (c) if the long flow is idle (*i.e.*, there is no match on the entry for a period that exceeds the entry's pre-configured TTL), the entry is deleted from the table and N is decremented.

contain w cells each. Thus, the data structure can be seen as a matrix of size $w * d$. The CMS uses d pairwise-independent hash functions $h_i, ..., h_d$ that are applied to the 5-tuple fields in the packet headers. The results of the hash functions correspond to the indices of the counts in the d register arrays; these counts are incremented by one. Calculating the minimum between these counts gives an approximation of the packet counts per flow; note that this is an approximation and not the exact count because collisions might occur, which leads to overestimating the counts. A nice property of the CMS is that it is designed to have provable L1 error bounds for frequency queries:

$$P[\hat{x_i} - x_i \geq \epsilon ||\mathbf{x}||_1] \leq \delta, \tag{1}$$

where $\hat{x_i}$ is the estimated frequency, x_i is the true frequency, ϵ is the error factor, and δ is the error probability. In other words, the estimation error exceeds $\epsilon||x||_1$ with a probability smaller than δ. Given ϵ and δ, it is possible to dimension the sketch with $w = \lceil \frac{e}{\epsilon} \rceil$ and $d = \lceil \ln \frac{1}{\delta} \rceil$. The scalability and the implementation details of the metrics computation within the P4 switch are listed in the Appendix.

After counting the number of packets for a given flow, a match-action table that implements the idle timeout mechanism is applied (Fig. 4 (b)). With idle timeout activated, the table automatically removes an entry when its idle time exceeds a Time-To-Live (TTL) value. This table matches on the flow identifier; if there is a `miss`, and the count of packets extracted from the CMS exceeds a threshold, the data plane sends a digest to the control plane to inform it that there is a new long flow. The control plane then increments its long flow count (N) and inserts into the match-action table an entry indexed with the flow identifier. Subsequent packets belonging to this long flow will `hit` when the table is applied, preventing the control plane from removing the entry. Finally, if the flow is idle (*i.e.*, there is no match on the entry for a period that exceeds the entry's TTL), the control plane decrements N and clears the entry from the table.

D. Queueing delays

P4BS continuously measures the queueing delay at the currently configured buffer. This metric is important because it conveys the share of the buffer that is being used by the current traffic. Furthermore, measuring the queueing delay is essential to make sure that SLAs are not being violated.

Recall that network taps are placed before and after the legacy router. To measure the queueing delay, the data plane must first associate and match incoming packets from both taps. Once there is a match (*i.e.*, the same packet arrives from both taps), the data plane computes the difference between the arrival time of both packets. Associating packets on the data plane and calculating the queueing delay is achieved as follows (see Fig. 5). Let the ingress packet denote the packet arriving before entering the legacy router, and the egress packet denote the packet arriving after leaving the legacy router. (1) When the ingress packet arrives at the P4 switch, a hash of the flow identifier (5-tuple) along with the TCP SEQ number of that current packet is calculated. The resulting hash value is used as an index to an array cell where the arrival timestamp is stored. (2) When an egress packet arrives, the data plane computes the hash of the flow identifier (5-tuple) along with the TCP SEQ number, and retrieves the timestamp stored in the array index by the hashed value. This item in the array is now removed. (3) The data plane computes a queueing delay sample by subtracting the retrieved timestamp from the current timestamp. (4) The queueing delay sample is fed to an

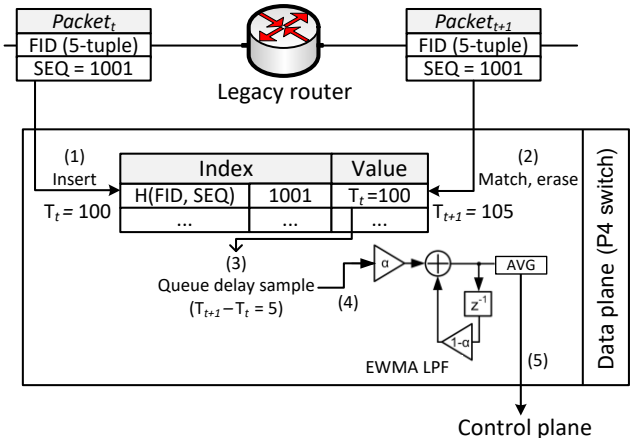

Fig. 5. Queue delay calculation in the data plane.

Exponentially Weighted Moving Average (EWMA) Low Pass Filter (LPF) to compute the average. (5) The average sample is pushed to the control plane. A variation of this method can leverage counting bloom filters (CBF) (as done in [48]) to scalably measure the delay, at the cost of decreasing the accuracy.

V. Buffer modification

Modifying the buffer size consists of tracking the statistics of the measured metrics and calculating a buffer size that could potentially enhance the performance.

A. Statistics tracking

The statistics tracking module resides on the control plane of the programmable switch. It mainly tracks the averages of the computed metrics. It also calculates the performance function which needs to be maximized.

The statistics tracking module continuously tracks the smooth RTT of the existing flows. Upon receiving an RTT sample (RTT_i) from the data plane for a certain flow i, the control plane computes the average RTT using the harmonic mean: $\overline{RTT} = \frac{N}{\sum_{i=1}^{N} \frac{1}{RTT_i}}$, and then computes a smoothed RTT: $SRTT = \alpha \cdot SRTT + (1-\alpha) \cdot \overline{RTT}$, where α is a smoothing factor ($0 \leq \alpha \leq 1$). The value of α = 0.875 is used in this work, following the recommendation of Jacobson [49]. The choice of this specific value is based on empirical observations and aims to strike a balance between responsiveness and stability in estimating the RTT.

The statistics tracking module also gathers the current packet loss p_t and computes a smooth average, denoted as p. It also gathers the average queueing delay d which is computed in the data plane.

B. Buffer size calculation

The end goal is to find a buffer size that minimizes both the average packet loss rate p and the average queueing delay d, while maximizing the link utilization. The buffer should also satisfy the constraints ($\hat{p}, \hat{d}$). Note that in some cases, it might not be feasible to satisfy both $\hat{p}$ and $\hat{d}$. It is important to highlight the intrinsic relationship between these variables. When increasing the buffer, p decreases while d increases, and vice-versa. This relationship has been proven in [6, 13, 30, 33], and is demonstrated in the Appendix.

Let $f_1(\cdot)$ denote the function that returns the average packet loss rate p, and $f_2(\cdot)$ denote the function that returns the average queueing delay d. Both f_1 and f_2 are *unknown* and are being sampled at given times. In addition to being unknown, both functions are also possibly *noisy*. Let $f(\cdot)$ denote the function that combines f_1 and f_2 using the weighted sum method, with weighting coefficients satisfying $\sum_{i=1}^{2} w_i = 1$, $w_i \in [0,1]$, as follows:

$$f(\cdot) = -\Big[w_1 f_1(\cdot) + w_2 f_2(\cdot)\Big] \tag{2}$$

$f(\cdot)$ is the performance function that is also unknown and noisy. $f(\cdot)$ adds a monotonically decreasing function $f_1(\cdot)$ with a monotonically increasing function $f_2(\cdot)$ as the buffer size increases. If both functions $f_1(\cdot)$ and $f_2(\cdot)$ increase/decrease at consistent rates, then $f(\cdot)$ will be a unimodal function[1] However, this is not the case here (see Appendix.B); $f(.)$ is multimodal, and therefore, can have multiple solutions.

Algorithm 1: Shrinking the search space

Input
- Maximum tolerated queueing delay $\hat{d}$
- Link capacity C
- Bandwidth-delay product BDP

begin
- $x \leftarrow$ `BDP`
- `set_buffer(`x`)` ▷ Set buffer size to x
- `wait ( )` ▷ Wait for changes
- $p \leftarrow f_1(x)$ ▷ Obtain current loss rate
- **while** $p > \epsilon_p$ **do**
  - $x \leftarrow x \cdot 2$ ▷ Exponential increase
  - **if** $x > C \cdot \hat{d}$ **then**
    - $x \leftarrow C \cdot \hat{d}$
    - **break**
  - `set_buffer(x)`
  - `wait ( )` ▷ Wait for changes
  - $p \leftarrow f_1(x)$ ▷ Current loss rate
- `bounds` $\leftarrow \left[\frac{C \cdot RTT_{min}}{\sqrt{N}}, x\right]$ ▷ Search space

The goal is to maximize the performance function (i.e., the negative of the delay and the loss) while possibly converging to the optimal configuration $x^* = \text{argmax}_x\, f(x)$.

Note that the link utilization is excluded from $f(.)$ because the minimum allowed buffer size corresponds to a value that was proven to achieve high link utilization, as described next.

Defining bounds. The buffer size x must be large enough to ensure a high link utilization (i.e., $\rho \approx 100\%$). Building on the analysis of [6], a small buffer of size equals to $\frac{C \cdot RTT_{min}}{\sqrt{N}}$ leads to full link utilization, regardless of the CCAs being used. Moreover, this buffer size leads to low queueing delays, and hence, shorter RTTs. Nevertheless, $\frac{C \cdot RTT_{min}}{\sqrt{N}}$ was criticized for inducing excessive packet losses when the number of long flows N is large. Assuming that p is small when $x = \frac{C \cdot RTT_{min}}{\sqrt{N}}$ (*i.e.*, N is small), the value of the optimum buffer x^* is close to x. Accordingly, the smallest x that can ensure high ρ, small d, and in some cases, low p, is $\frac{C \cdot RTT_{min}}{\sqrt{N}}$. On the other hand, the tolerated queueing delay $\hat{d}$, which is specified by the network operator, corresponds to the largest buffer size that can be configured. Therefore, x will be configured over the interval $[\frac{C \cdot RTT_{min}}{\sqrt{N}}, C \cdot \hat{d}]$.

Instead of considering the whole search space $[\frac{C \cdot RTT_{min}}{\sqrt{N}}, C \cdot \hat{d}]$, P4BS attempts to shrink the bounds by finding a buffer where the packet loss rate is very small (below a threshold ϵ_p) as shown in Algorithm 1. The intuition here is that when the packet loss rate is small, increasing the buffer will only increase the queueing delay, and hence, should be avoided. The algorithm continues to double the buffer size as long as

[1] A function $f : [a,b] \rightarrow \mathbb{R}$ is *unimodal* if and only if there exist $x^* \in [a,b]$ such that $f(x^*) \geq f(x), x \in [a,b]$, f is strictly increasing in $[a, x^*]$, and f is strictly decreasing in $[x^*, b]$ [50].

the loss rate is larger than ϵ_p or if the buffer size exceeds the maximum tolerated queueing delay $\hat{d}$ at line rate. In the latter, the upper bound is set to $C \cdot \hat{d}$.

Modeling using Gaussian Process (GP). The performance function $f(.)$, which combines the loss and the delay, is modeled as a GP (a collection of random variables), and this GP is used as an inference engine to predict the value of a buffer size configuration that has not been tested in the environment. Bayesian Optimization (BO) [51] technique uses the GP as the basis routine. BO will suggest the next buffer size to evaluate in the environment through the learned GP model. The GP model is constructed using the performance of the observations from the previous buffer size configurations. GP has been shown to work efficiently and to provide a tradeoff between the modeling accuracy and the inference complexity.

GPs for the buffer sizing problem. GPs are a collection of random variables where a finite number of these are jointly normally distributed. Let the history of observations seen after deploying a sequence of buffer sizes $x(1)$, $x(2)$, ..., $x(n)$ be noted as:

$$o(n) = [\widetilde{f}(x(1)), ..., \widetilde{f}(x(n))]^T \tag{3}$$

where $\widetilde{f}(.)$ is the realization of $f(.)$ for a given buffer size. Using conditioning for normal variable and the GP *posterior* probability, the performance function can be inferred as follows:

$$p(f(x)|o(n)) = \mathcal{N}(\mu_f \underline{+\Sigma_{fo}\Sigma_o^{-1}(o(n)-\mu_o)}, \Sigma_f \underline{-\Sigma_{f_o}\Sigma_o^{-1}\Sigma_{fo}^T}) \tag{4}$$

where $\mu \in \mathbb{R}$ is the mean and Σ is the covariance matrix. The underlined factors in Eq. (4) refer to the corrections in the mean and covariance of f brought by the knowledge of o. The mean vectors μ_f and μ_o are computed using prior mean function $m(.) : \mathcal{X} \to \mathbb{R}$. This function defines for each buffer size x the system's prior belief $m(x)$ of the performance. m can be set to 0 when no prior belief is available. The covariance matrices Σ_{fo}, Σ_o, and Σ_f are computed after constructing a function $C(.,.) : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ defined through kernel function K_θ, where θ is a hyperparameter. In the buffer sizing problem, the selected kernel function is the Radial Basis Function (RBF) defined as:

$$K_\theta(x, x') := a \exp\left(-\frac{||x - x'||}{2b^2}\right) \tag{5}$$

where $\theta = [a, b] > 0$; a is a hyperparameter that regulates the amplitude of the function and b is a hyperparameter that controls the smoothness. RBF was selected after thorough experimentation because it consistently outperformed other functions, such as Matérn [52], with respect to the average improvement per iteration, the final value of the objective function, and the convergence rate. $C(x, x')$ describes how close the performance of the system is when trying configurations x and x', and ideally, $C(x, x') \approx Cov(f(x), f(x'))$. Since the system is subject to noise, C is computed by adding $K(.,.)$ to σ^2, the observation noise variance, only if x and x' are equal.

Once C and m are defined, the mean vectors and the covariance matrices in Eq. 4 can be set as follows.

Algorithm 2: Buffer size searching

```
begin
    Initialize current iteration n = 0
    Define a maximum number of iteration n̄
    Set prior mean function m(.) to 0
    Select a covariance kernel function K_θ(.,.);
    Initialize hyperparameters θ, σ
    Select a termination threshold ε
    while max_{x∈X} u(x | o(n)) ≥ ε and n ≤ n̄ do
        Obtain the next buffer x(n+1) using Eq. (12)
        Deploy x(n+1) and observe f̃(x(n+1))
        Update hyperparameters θ, σ using Eq. (11)
        n ← n + 1
    Stabilize on x* = arg max_{x(i),i=1,...,n} f̃(x(i))
```

$$\mu_f = [m(x)] \tag{6}$$

$$\mu_o = [m(x(1)), ..., m(x(n))]^T \tag{7}$$

$$[\Sigma_o]_{i,j} = C_{\theta,\sigma}(x(i), x(j)) \quad \text{for} \quad i, j = 1, ..., n \tag{8}$$

$$\Sigma_f = C_{\theta,\sigma}(x, x) \tag{9}$$

$$[\Sigma_{fo}]_i = C_{\theta,\sigma}(x, x(i)) \tag{10}$$

The GP then tunes the hyperparameters θ and σ in real-time while seeing new observations by using the maximum likelihood estimation:

$$\underset{\theta,\sigma}{\operatorname{argmax}}\, p(\widetilde{f}(x(1)), ..., \widetilde{f}(x(n))) = \mathcal{N}(\mu_o, \Sigma_o) \tag{11}$$

BO for the buffer sizing problem. The posterior distribution allowed predicting the value of the performance function after seeing the historical observations. BO uses the observations to select the next buffer $x(n+1)$ to be deployed. At each step, an acquisition function $u(, |o(n))$ is optimized by the BO engine:

$$x(n+1) = \underset{x \in \mathcal{X}}{\operatorname{argmax}}\, u(x, o(n)) \tag{12}$$

In the buffer sizing problem, the selected acquisition function is the Expected Improvement (EI) [53] (see Section VI-A):

$$u^{EI}(x|o(n)) = \mathbb{E}\left[f(x) - \max_{i=1,...,n} f(x(i))|o(n)\right]^+, \tag{13}$$

where $[.]^+ = \max(0, .)$. EI has been popular because u^{EI} has a closed form for GPs. More details can be found in [54].

Algorithm 2 shows the buffer size searching steps. The search continues while the expected improvement of the performance exceeds a threshold ϵ and the number of iterations is less than the maximum number of iterations $\bar{n}$. Afterwards, the system stabilizes on the buffer $x^* = \arg\max_{x(i),i=1,...,n} \widetilde{f}(x(i))$ which attained the maximum performance of the system so far.

Since the traffic conditions change with time, the performance function shape might change as well. After stabilizing on x^*, P4BS continues to compute $f(\cdot)$ and uses the standard score (z-score) to identify major changes in the network conditions; if the value is several number of standard deviations away from the moving mean, the change will be signaled and the searching procedure is restarted. A potential alternative to

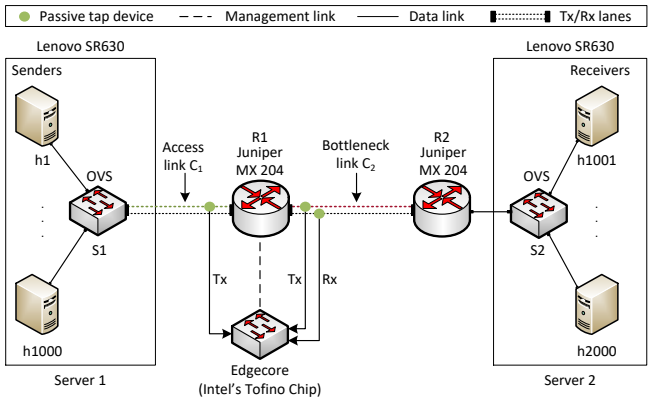

Fig. 6. Topology used for the experiments.

avoid restarting the search is to use the method proposed in [55] which applies BO to time-varying dynamic functions; this is kept for future work.

VI. Experimental Setup and Results

Fig. 6 shows the topology used to conduct the experiments. The topology consists of up to 1000 senders (h1, h2, ..., h1000), each sending data to a corresponding receiver (h1001, h1002, ..., h2000). The hosts in the experiments are network namespaces in Linux. The emulation was carefully designed, and sufficient resources were allocated: each server has 48 Xeon 4214 cores operating at 2.2 GHz and 80GB of RAM. The usage of CPUs was at all times below prudent levels, thus avoiding misleading results. The senders are connected to an Open vSwitch (OVS) (S1) [56], which is bridged to the server's (Server 1) network interface. The server's interface is connected to a Juniper router MX-204 (R1). Edgecore Wedge100BF-32X [57] is the programmable P4 switch used (Intel Tofino). The end-hosts were carefully tuned; the size of the TCP send and receive buffers was set to a large value ($\approx$ 250MB). All tests are configured with a total propagation delay of 20ms, unless otherwise specified. Additionally, random emulated delays [0-15ms] were introduced to each flow to prevent global synchronization. The Network Emulator (NetEm) tool [58] was used to set the delay. The buffer size on the router R1 was modified using the `buffer-size` command under the `class-of-service` configuration in Junos. Every experiment is repeated 10 times and results are averaged for better accuracy. The bandwidth of the links connecting S1 to R1 (C_1) and R1 to R2 (C_2) are configurable.

The source code of P4BS is publicly available online[1].

A. Algorithm's evaluation

Buffer searching dynamics. Fig. 7 demonstrates the iterative process of determining the optimal buffer size in a sample experiment. Each row corresponds to a step in the buffer search, where the left figures depict the acquisition function over time, and the right figures illustrate the GP. The solid blue line in the right figures represents the mean of the GP

[1]https://github.com/ekfoury/p4bs-bo

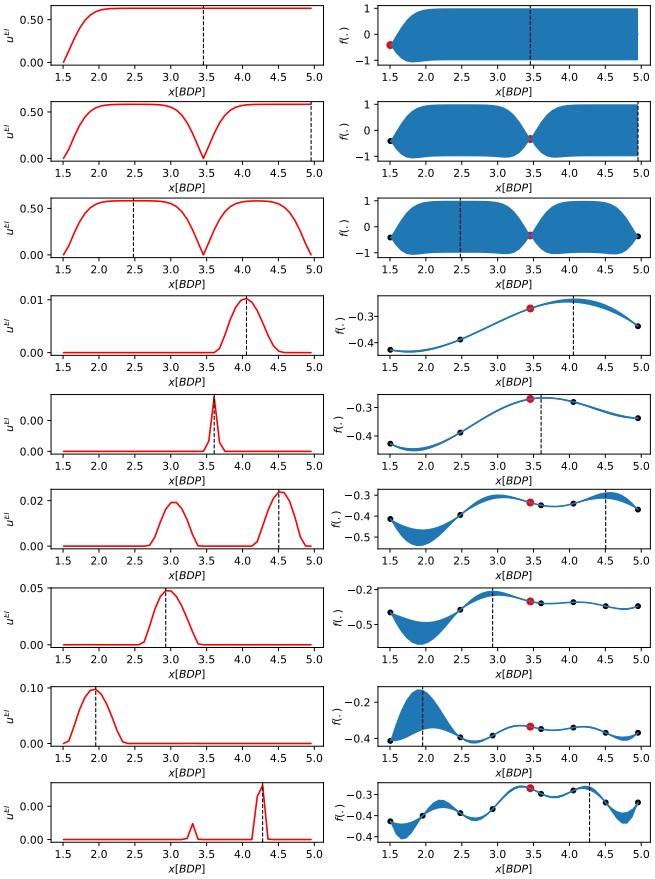

Fig. 7. Buffer searching dynamics in the access network. Left is u^{EI} over time; the solid blue line (right) denotes the mean associated to the GP posterior. The shaded blue region denotes the confidence interval. Each row shows an iteration. The vertical dotted line, which maximizes u^{EI}, shows the location of the next buffer x to be tested.

posterior, while the shaded blue region indicates the confidence interval.

In this experiment, the bottleneck link is shared by 500 large flows, with $C_1 = 40Gbps$ and $C_2 = 1Gbps$, simulating an access network scenario. During the initial iteration (first row), the selected buffer size was approximately 1.5BDP, denoted by the red point in the right figure. Note the considerable uncertainty associated with larger buffer sizes. The acquisition function reaches its maximum value at $x \approx 3.5BDP$ (first row, first column, indicated by the dashed line), determining the buffer size to be evaluated in the subsequent iteration. This process continues until convergence. In BO, the theoretical number of iterations tends to infinity ($T \to \infty$), although in practice, the algorithm is typically executed for a finite number of iterations. The final selected buffer size is the one that maximizes the performance function, approximately 3.5BDP.

Fig. 8 demonstrates the buffer selection process for the same experiment but with $C_1 = 10Gbps$ and $C_2 = 2.5Gbps$, resembling a core network scenario. It is noteworthy that the algorithm favored a smaller buffer size of approximately $2BDP$, contrasting with the suggested buffer size in the access network setting ($3.5BDP$).

Convergence. The searching algorithm should ideally find a

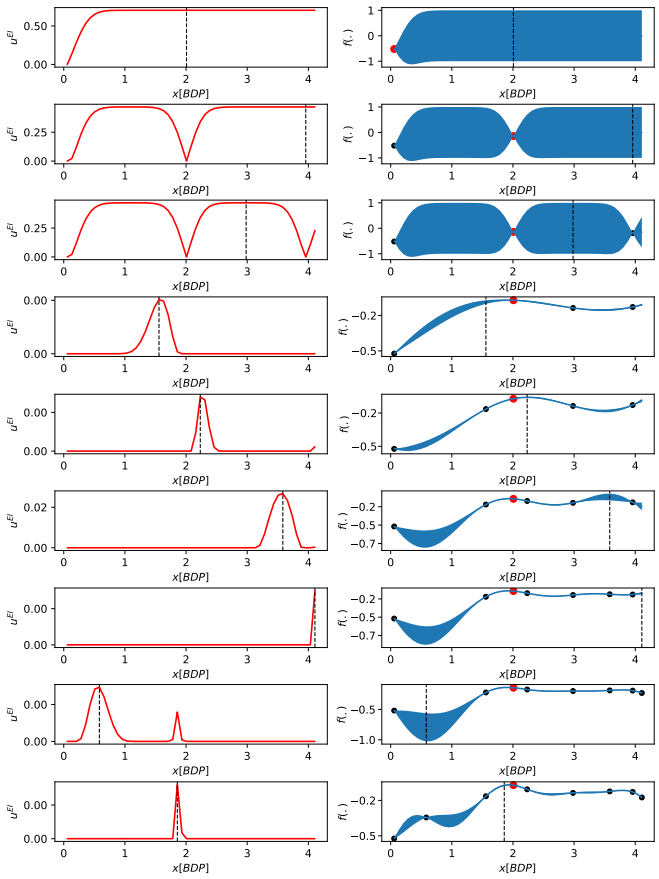

Fig. 8. Buffer searching dynamics in the core network.

buffer with the least number of iterations. Fig. 9 (a) demonstrates the average number of searching iterations considering different N. With smaller N, the search space is small (due to low packet loss rates), and hence, it takes less time to stabilize on a buffer. A larger number of flows on the other hand require more iterations. Fig. 9 (b) shows the average number of iterations as a function of the upper search bound of a given traffic scenario (determined in the last step of Algorithm 1). Similar observations to those discussed for the number of flows can be seen here.

The convergence of BO is commonly measured by using the cumulative *regret* which is defined as $R_T = \sum_{t=1}^{T} r_t$, where $r_t = f(x^*) - f(x_t)$ is the instantaneous regret at time t. According to [59], R_T grows almost at a sublinear rate for the EI function using $y^{max} = \max_{x \in \mathcal{X}} u(x)$ as the incumbent.

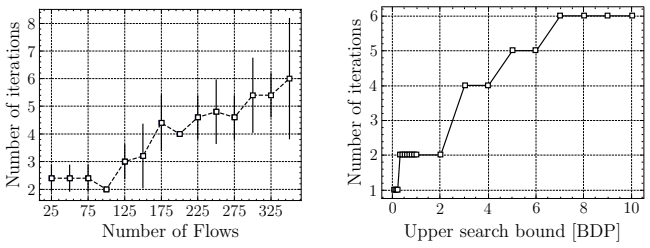

Fig. 9. Average number of iterations: (a) different flow counts, (b) different upper search bounds.

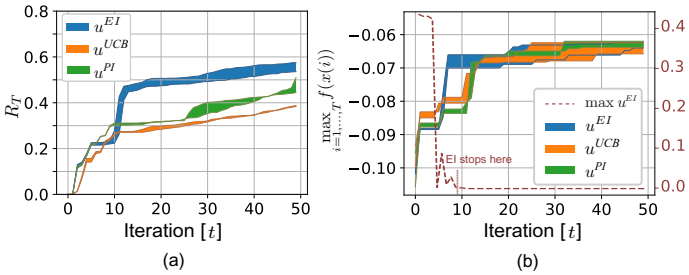

Fig. 10. Convergence for three acquisition functions. (a) Regret R_T over time. All acquisition functions achieved a sublinear regret; (b) Maximum attained value of $f(.)$ over time. EI found a maximum value of $f(.) \approx -0.07$ faster than UCB and PI. While the figure shows 50 iterations, EI stopped at the eighth iteration because $\max_{x \in \mathcal{X}} u^{EI}(x \mid o(n)) < \epsilon$ (see Algorithm 2).

This experiment evaluates the convergence in an access network with 100 Cubic flows. Fig. 10 (a) shows the regret over time for three different acquisition functions (EI, Upper Confidence Bound (UCB), and Probability of Improvement (PI)). It can be seen that the system converges as the regret grows sublinearly with time, regardless of the acquisition function. Although UCB and PI attained a smaller R_T than EI, EI was selected due to its ability to stop the search and prevent unnecessary evaluations when the expected improvement is smaller than a threshold [59]. Fig. 10 (b) shows the maximum value of $f(.)$ with time for the three acquisition functions. Note that EI found a maximum value of $f(.) \approx -0.07$ faster than UCB and PI. EI stopped at the eighth iteration.

B. Network metrics

Fig. 11 shows the average performance function, denoted as $\overline{f(.)}$, in an access network (top row of heatmaps) and a core network (bottom row). The test is executed for 120 seconds, and $f(.)$ is sampled every second[2]. A higher or greener color indicates better performance, as $f(\cdot)$ combines the negatives of loss and delay (see Eq. 2). The heatmaps represent different buffer sizing regimes, with each row within a heatmap corresponding to a CCA, and each column representing a different number of flows N.

The experiment compares the performance of P4BS against several well-known buffer sizing rules, namely the *BDP* rule, the *Stanford* rule, *BSCL*, the *Tiny* buffer rule, and the *Bloated* buffer. The Bloated buffer refers to the buffer size recommended by router vendors, typically at least 200ms [9]. Additionally, the comparison includes ADT, a dynamic buffer sizing algorithm.

When using smaller buffers (such as Tiny, Stanford, and ADT), $\overline{f(.)}$ is low due to excessive packet losses. This observation also holds true for BSCL and BDP when N is large.

The Bloated buffer has lower loss rates but suffers from high queuing delays, leading to a low $\overline{f(.)}$ as well.

In comparison, P4BS improves the packet loss rate compared to all buffer sizing rules except for the Bloated buffer. The queuing delay in P4BS remains low when N is small, but

[2] $\overline{f(.)} = \frac{\sum_{i=1}^{120} \hat{f}_i(.)}{120}$, where $\hat{f}_i(.)$ is the sampled $f(.)$ at time i.

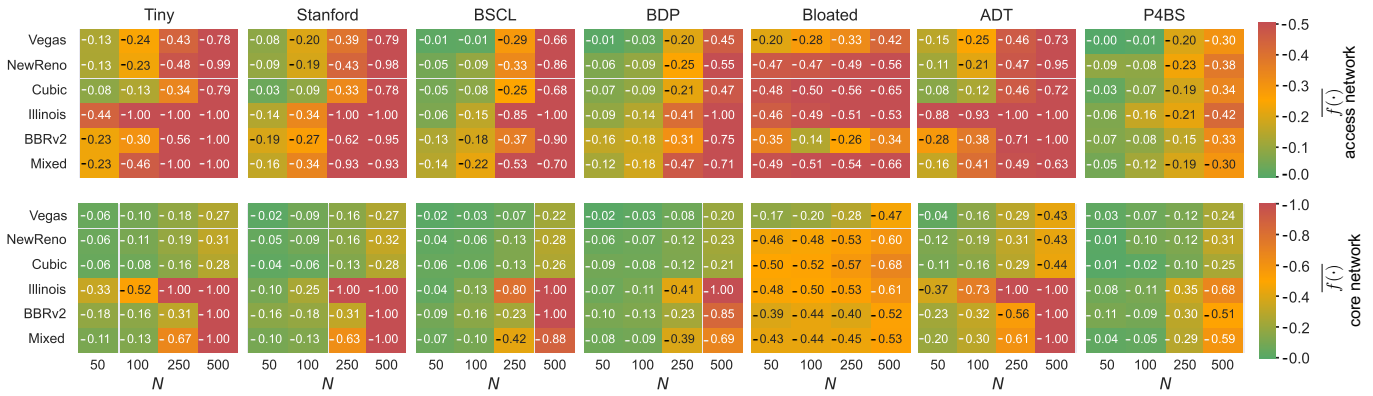

Fig. 11. The average performance function $\overline{f(.)}$ in the access network (top row) and in the core network (bottom row), using various buffer sizing regimes, various CCAs, and various N.

it gradually increases as N grows. Considering the combined performance function $f(\cdot)$ (loss and queuing delay), P4BS outperforms all buffer sizing rules, regardless of N, CCA, or the deployment setting (core and access).

C. Voice over IP traffic

The router buffer size has a significant impact on the Quality-of-Service (QoS) of voice traffic [60]. Specifically, the metrics affecting the QoS include packet losses, jitter, and transit delay. The experiments discussed in this section follow the same structure as those conducted in [60]. The speech signal degradation and the conversational dynamics are often measured to evaluate the QoS of VoIP. The speech signal degradation is assessed through the standardized Perceptual Speech Quality Measure (PESQ) model [61]. PESQ calculates a score (PESQ-MOS) in the range [1, 5] by comparing an error-free audio signal to a degraded one. This score is remapped to [0, 100] using [62] and is now referred to as the $z1$ score. The quality worsens with smaller $z1$ scores. The $z1$ score is affected by the packet losses and the jitter but not the transit delay; thus, PESQ only accounts for the listening quality of audio and not for the conversational dynamics. The conversational dynamics consider the transit delays and are assessed through the E-model delay impairment factor of the ITU-T E. The resulting score is referred to as the $z2$ score. The $z2$ score is in the range [0, 100], and worsens with higher values (unlike the $z1$ score). According to the International Telecommunication Union (ITU G.114) [63], transit delays should be below 150ms or at most 400ms.

As the QoS of voice is affected by both the speech signal degradation and the conversational dynamics, it is important to combine both scores and get a final score that considers all metrics. Since the semantics of $z1$ and $z2$ are reversed, the final score (z) is computed as $z = \max(0, z1 - z2)$ [60]. With such a calculation, if the loss and jitter are negligible ($z1$ is high) but the delays are large ($z2$ is high), then the overall score z is low, reflecting a poor quality, and vice-versa.

Experiment setup. The test consists of 100 VoIP calls playing the 20 reference speech samples (G.711.a (PCMA) narrow-band codec) recommended by the ITU [64] for speech assessment. The SIPp [65] open-source SIP traffic generator is used to establish multiple concurrent sessions and to generate media (RTP) traffic. Additionally, 100 hosts are generating background traffic using iPerf3 to make sure that the link is fully utilized and the buffer is being used.

Results. Fig. 12 shows the results obtained from this experiment. The transit delay was high with the Bloated buffer and short with smaller buffers. Note how P4BS has a higher transit delay than the Stanford, Tiny, and BSCL, but is still much smaller than the perceivable value specified by the ITU (100ms). The PESQ-MOS, which captures the packet losses and the jitter, degrades with smaller buffers or excessively large buffers. P4BS attained the maximum PESQ-MOS score

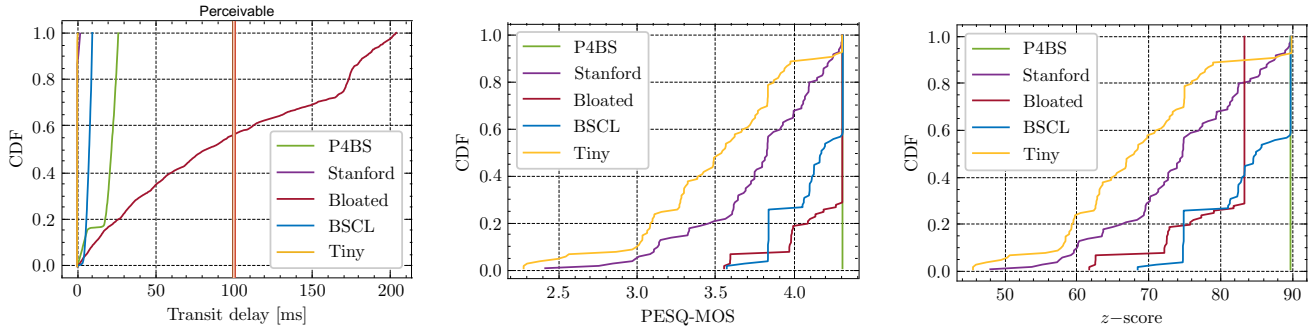

Fig. 12. VoIP metrics for 100 calls playing the ITU's 20 speech samples, using various buffer sizing regimes.

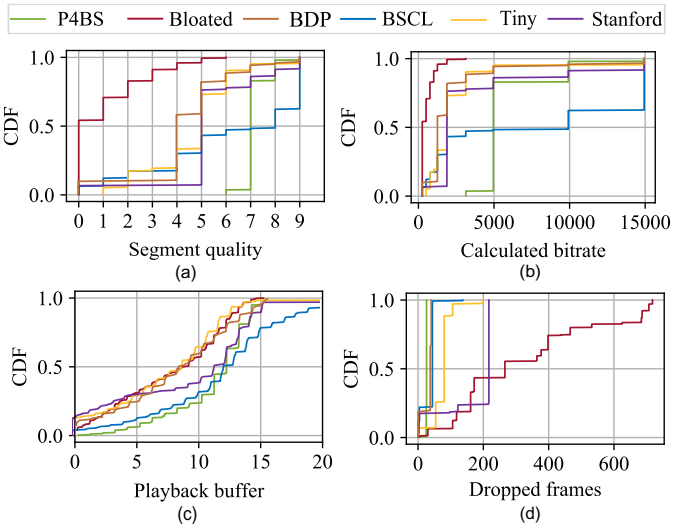

Fig. 13. Performance of DASH using various buffer sizing regimes: (a) number of dropped frames, (b) calculated bitrate, (c) playback buffer, (d) segment quality.

and outperformed all the other rules. Similarly, when combining both scores into the z-score, P4BS achieved the best score among all the other buffer configurations. The reason P4BS achieved such results is due to its ability to sacrifice a small amount of transit delays to mitigate packet losses.

D. Video streaming (DASH)

The majority of video streaming traffic is delivered over HTTP with adaptation protocols such as Dynamic Adaptive Streaming over HTTP (DASH). DASH has been supported by dominant video streaming providers like Netflix and YouTube [66]. Due to the increased popularity of video streaming and DASH, this section evaluates P4BS with such traffic and compares key metrics that affect the QoE for the end-user. In DASH, videos are divided into equal-length segments and hosted on the web server providing HTTP video streaming services. The segments are encoded in various bitrates such that lower bitrates provide lower video quality but with a smaller file size. In addition to the videos, there is a manifest file that describes the metadata of the segments and their bitrates. When the streaming starts, the DASH client downloads the manifest file and learns about the available bitrates. The client then downloads video segments with the goal of maximizing the user's QoE.

Metrics. The key metrics that influence the QoE include 1) Playback buffer size: clients are designed to buffer a specific duration of the video before triggering the stream playback. The larger this duration is, the better; 2) Segment quality: since DASH adapts the current segment quality based on the available bitrate, there could be multiple segment qualities during a single video playback. The probability distribution of the segment qualities is a representative indicator of the QoE for the end-user; 3) Dropped frames: segments are made up of a sequence of pictures. The number of dropped frames, if excessive, can result in a noticeable drop in quality [67]; 4) Bitrate: the bitrates used during video playback; 5) Stall rate: computed as (total playback time – video length) / video length. A stall rate larger than 0 indicates that the user experienced re-buffering. The experiment is executed with the reference open-source DASH player `dash.js`, using the reference video [68] which is encoded into 10 different video quality levels. The total video length is 636 seconds, with 4-seconds encoded segments, resulting in a total of 160 segments. The experiment runs TCP background traffic to make sure that the link is oversubscribed.

Results. Consider Fig 13. Significant improvements have been observed across all metrics. With P4BS, the videos were streamed using segment quality indices ranging from 6 to 9, surpassing the other rules, except for BSCL. It should be noted, however, that BSCL yielded indices below 6 half of the time. The consistent variation in segment quality (witnessed in all but P4BS) has a negative impact on the user experience. Similar observations apply to the bitrate. The playback buffer time is also improved; only P4BS and the BSCL did not suffer from re-buffering, which is witnessed when the playback buffer is 0. The dropped frames are kept below prudent levels with P4BS, in contrast to the other rules. The primary reason behind the improvement achieved by P4BS lies in its ability to maintain small RTTs, striking a balance that minimizes packet drop rates.

E. Web browsing

A major share of the Internet traffic consists of web browsing. This section evaluates the Flow Completion Time (FCT) of traffic that resembles web browsing. To make sure that the buffer will be utilized, this experiment generates background traffic in the form of long flows with various loads. Alongside the long flows, the experiment initiates $5,000$

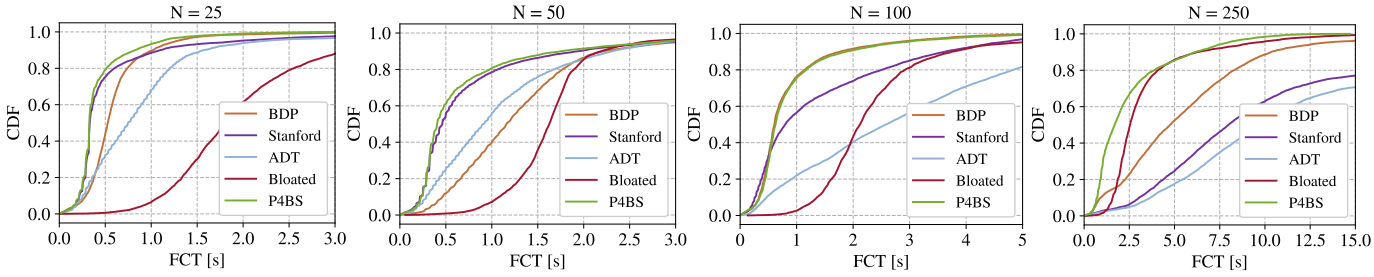

Fig. 14. Flow completion time of $10,000$ short flows sharing the link with long flows. Each figure shows the results with a different number of long flows (N). The CCA used is Cubic.

short flows whose inter-connection times are generated from an exponential distribution with a mean of one second. The web pages are generated to cover a wide range of sizes, from relatively small web pages to large web pages with images and figures [60]. The sizes are in the range [15KB, 2.5MB].

Consider Fig. 14. Regardless of the background traffic load, P4BS attained the lowest FCTs. When the load is small ($N = 25$ or $N = 50$), the packet loss rate is low, and thus, smaller buffers are preferable. This is why the FCTs in P4BS are comparable to those observed with the Stanford rule. The FCTs with the BDP and the Bloated rules are higher because packets are waiting in long queues. ADT selects the smallest buffer while maintaining 98% of link utilization. With such an extremely small buffer, the packet losses are high, leading to retransmissions and timeouts; thus, the flows' FCTs are high with the ADT rule.

When the load increases ($N = 100$), the packet losses increase, and thus, small buffers are no longer preferred. P4BS suggested a larger buffer, close to the BDP. This is why P4BS outperformed Stanford and ADT rules. Note, however, that the Stanford rule had few flows terminating faster; those flows were "lucky" and terminated without retransmissions.

Finally, with $N = 250$, the small buffer rules and the BDP are not preferred due to the large packet loss rates. P4BS suggested a large buffer that mitigates packet losses but is not too large to induce excessively large queueing delays, as with the Bloated rule.

It is important to note that P4BS was able to adapt to the network conditions and produced shorter FCTs regardless of the load. This shows that having a fixed rule will never be optimal in all cases.

F. Real backbone traces testing

Running P4BS over traffic from real-world traces is important to test its capabilities under varying network conditions and heterogeneous flows. The traces are typically available in PCAP format, where packets are stored exactly as they were observed during the capture. Unfortunately, replaying such traces on a per-packet basis (*e.g.*, using a tool such as `tcpreplay`) will not be useful since the closed-loop feedback of TCP sources will not be captured. Changing the buffer size on-the-fly will influence the TCP sources (decreasing the buffer size will cause some packets to be lost; increasing the buffer size will increase the RTT).

The replaying approach used in this test is as follows:

1) The PCAP files are analyzed to extract the TCP flow sizes (i.e., how many bytes a certain flow sent during its lifetime).
2) Only the long flows are selected. The selection is based on comparing the flow size against a predetermined threshold.
3) The inter-arrival times between the long flows are calculated.
4) The distributions of the size of short flows and their arrival times are analyzed.
5) The minimum RTT of a flow is extracted and configured as the flow's propagation delay.

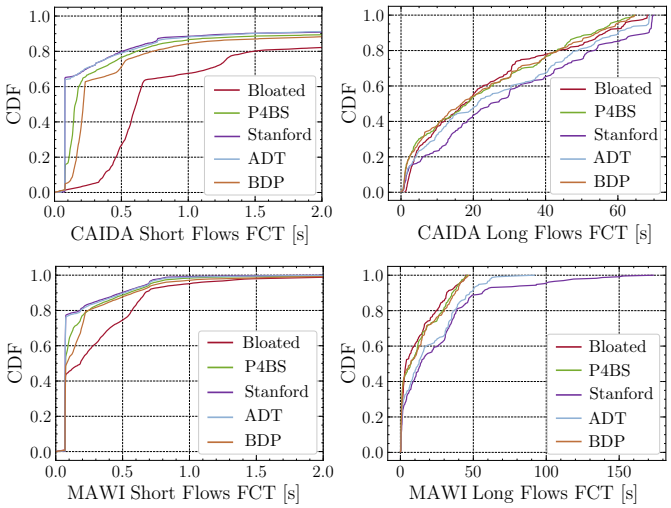

Fig. 15. Flow completion time of short (left) and long (right) flows of the CAIDA (top) and MAWI (bottom) datasets. The CCA used is Cubic.

6) The long flows are replayed based on their sizes and their starting times; the short flows are emulated based on the flow size and starting time distributions observed in the traces.

Two real-world traces are used in this test. The first is from the Center for Applied Internet Data Analysis (CAIDA) [69] which contains packet headers derived from 10Gbps traces on Equinix NYC monitor. The second is from Measurement and Analysis on the WIDE Internet (MAWI) which contains packet headers derived from 1Gbps traces on the transit link of WIDE to the upstream ISP [70]. Both traces are tested on a bottleneck link of 1Gbps.

The test compares the FCTs of short and long flows against the other buffer sizing rules. Fig. 15 shows the results. As expected, the Bloated buffer is the most convenient for the FCT of long flows; however, it increases substantially the FCT for short flows. This increase is significantly noticeable from an end-user perspective as short flows are mostly interactive flows (*e.g.*, web browsing traffic). Moreover, the increase in the RTT can potentially violate the delay limits in the SLAs. The opposite is true for the Stanford rule; while it shortens the FCT for short flows dramatically, those for long flows are greatly impacted. Furthermore, the loss rates imposed by Stanford and the ADT rules are significantly high, potentially violating the loss limits of the SLAs. P4BS was capable of finding a balance between the two such that the FCT of long flows is very close to that of the Bloated buffer and the FCT of short flows is close to that of the Stanford buffer. Additionally, both the loss rate and the queueing delays are minimized, making P4BS an SLA-compliant solution for the buffer sizing problem.

G. Research and Education (R&E) networks testing

Research and Education (R&E) networks carry flows that initiate large data transfers between research institutions. Internet2 is an example of a regional and national backbone R&E network. The traffic behavior in such networks is different from that of campus/enterprise networks; a small number of

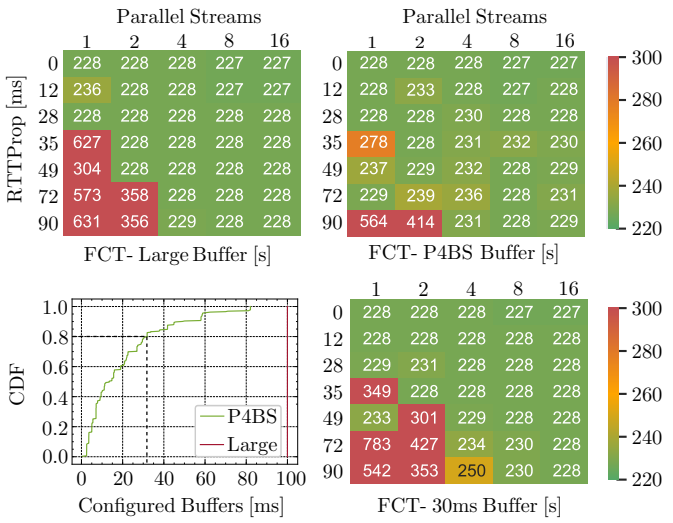

Fig. 16. Top: FCTs of long flows (left is fixed large buffer, right is P4BS); bottom left: buffers selected by P4BS; bottom right: FCTs of long flows using a 30ms buffer.

flows constitute the bulk of science flow traffic, consuming a large amount of bandwidth. Consequently, a large buffer size is preferred to absorb transient packet bursts generated by large flows [71, 72].

This experiment tests P4BS in a R&E network deployment where the bottleneck link (C_2) is 10Gbps and the access link (C_1) is 40Gbps. The Climate dataset which is provided by ESnet for testing the performance of Data Transfer Nodes (DTNs) is used [73]. The dataset is about 245GB in total size. While it is expected for flows to finish faster with a large buffer, the goal of this test is to infer how small can a buffer be to achieve FCTs comparable to those of a large buffer. This is important to understand the buffer requirements in such networks, and thus, plan/dimension smaller buffers in the future to save energy and money.

Fig. 16 shows the FCT of the transfers using a large buffer (top left) and P4BS (top right). The large buffer size is 100ms of line rate, equivalent to ≈125MB on a 10Gbps link. The test includes various propagation delays (RTTProp) which emulate geographically distant transfer nodes (*e.g.*, [0-12ms] RTT corresponds to a local/metro network, [12-49ms] RTT corresponds to a regional network, etc.). The RTTProp values are the same as those used by various throughput tests by ESnet. It can be noted that with parallel streams, the FCTs are barely affected since the losses only affect individual flows. Note how even with large buffers, losses might occur, causing the TCP flows to slow their transmission rates, and thus, increasing the FCT.

Fig. 16 (bottom left) shows the buffers selected by P4BS during the test. Note how 80% of the configured buffers were $<$ 30ms, a 70% decrease from the large buffer. The test is now repeated with a fixed 30ms buffer to see how well the derived buffer is (Fig. 16, bottom right). It can be observed that only negligible differences in the FCTs were witnessed, as long as the transfer uses parallel streams.

These preliminary results suggest that smaller buffers are acceptable as long as parallel streams can be used. Tools used by the R&E community for data transfers (*e.g.*, Globus [74]) are capable of launching parallel streams. While these results are preliminary, and more extensive tests are needed to generalize such a conclusion, the findings indicate that excessively large and Bloated buffers may not be necessary for R&E networks; if this is true, the reduction in the prices of routers can be significant.

VII. Discussions

A. Cost-efficiency and low overhead

P4BS requires installing a P4 switch and passive optical TAPs on the bottleneck router. The cost of such devices is negligible when comparing them to other devices acquired by network operators. A P4 switch is cheaper than most legacy switches used on campus/enterprise networks. Furthermore, it is possible for a single switch running P4BS to serve multiple bottlenecked routers (*e.g.*, in ISP networks). It is worth mentioning that TAPs do not affect the line rate of the original traffic and do not impose memory/storage overheads.

The concept of utilizing passive P4 switches has been successfully applied in production networks, notably in Princeton's P4Campus project [75]. With such a deployment, applications other than buffer sizing can also be implemented. For instance, at P4Campus, microbursts measurements, heavy hitter detection, live traffic anonymization, and operating system fingerprinting have been implemented [76].

B. Playground for buffer sizing and CCAs research

There is a noticeable gap between theory and practice in the buffer sizing literature. The buffer sizing rules published in the past 20 years have been primarily tested on simulators such as NS-2 ([6, 13, 32, 33]). The design introduced in this paper (i.e., the closed-control loop between a legacy router and a P4 switch) can potentially be used by future buffer sizing research to test algorithms against realistic traffic rather than idealistically simulated ones. The authors believe that the ability to devise custom algorithms, empowered by the high-precision timers of the P4 hardware switch, can pave the way to further innovation in buffer sizing research (optimal buffer size choice, buffer size dimensioning, CCA design, etc.).

C. Challenges in testing with real traces

Out of all the surveyed work, only the Stanford rule was deployed for a short time for testing in a production network [4]. It is very difficult to convince operators to change their router's buffer size for experimentation (fear of violating SLA hard limits and paying penalties, fear of network disruption, etc.). To test P4BS against real traces, the authors adopted heuristics to process and replay the PCAP files. While there are some available hardware and software traffic generators (see [77] for a comprehensive survey), they are far from being convenient for buffer sizing research. The trace-driven traffic generators are mostly outdated (*e.g.*, Harpoon [78] relies on NetFlow records and has not been updated since 2005), incompatible (*e.g.*, Swing [79] only works in conjunction with an outdated network emulator), and open-loop (Harpoon does

not consider the closed-loop of TCP). The authors believe that there is an utmost need for a well-maintained, open-source, and reliable traffic generator that can precisely replay traffic from real traces while preserving their characteristics. This would be useful for buffer sizing research and Active Queue Management (AQMs), among others.

D. CCA-agnostic

The majority of buffer sizing research papers assume that the flows are following the AIMD law of loss-based CCAs. A recent paper studied buffer sizing for flows with heterogeneous CCAs, and discovered that substantially different buffer sizes are needed by various network scenarios (*e.g.*, Reno requires a buffer $\approx$ 1BDP while BBR requires a buffer $\approx \frac{1}{4}$BDP). In P4BS, the goal is to minimize the measured losses and delays while maintaining high link utilization, making the solution agnostic to the CCAs being used by the flows (see Fig. 11 how P4BS improved the objective function for all the tested CCAs).

E. Deployment

There are two main reasons why the system considers an inline legacy router instead of a P4-programmable switch:

1) Networks today primarily rely on non-programmable routers, which are prevalent in various network environments including campus, enterprise, and ISP networks. The objective of the proposed system is to enhance the performance of these networks by leveraging the computational power of a P4 switch. To ensure smooth integration with the current infrastructure, the P4 switch is deployed passively, allowing it to compute metrics alongside the non-programmable routers.
2) The shared total buffer size in available P4 programmable switches is small (e.g., Broadcom Tomahawk has 16MB total buffer size; Intel's Tofino has 22MB total buffer size, it has 64 ports with 100Gbps bandwidth (3.52KB per port) [80–82]. With such a small buffer size, these switches are commonly used in access or distribution layers where the traffic volume is relatively low, and the buffer requirements may not be as demanding [81]. Even if P4BS was implemented entirely on the P4 switch (i.e., without the legacy router), the decision loop performance would still be the same as the buffer size of the P4 switch can only be modified from the control plane (i.e., not possible to modify the buffer size from the P4 program).

F. Technology availability

P4BS was implemented using Intel's Tofino. Recently, Intel announced that it will stop the development of the next-generation Intel® Tofino (Tofino 3). However, Intel will continue to sell and support existing Tofino products. It is worth mentioning that P4BS is not limited to Intel's Tofino and can be implemented on any programmable data plane device that supports custom packet processing and stateful memory. With the networking industry transitioning towards programmable SmartNICs, the authors envision a modified version of P4BS where the SmartNIC performs real-time calculations of buffer size-related metrics at line rate, while the server housing the SmartNIC would maintain the statistics and dynamically adjust the router's buffer size. An example of such a SmartNIC is Intel's Infrastructure Processing Unit (IPU) (e.g., Intel® IPU E2000), which uses an ASIC-based programmable pipeline, similar to that of the Tofino-based switch [83].

VIII. Conclusion

This paper presented P4BS, a scheme that dynamically modifies a router's buffer size to adapt to the network conditions. P4BS uses measurements collected and processed by P4 programmable switches and calculates metrics that are vital for deciding on the buffer size. Such metrics are the number of long flows, the RTTs, the queueing delays, and the packet loss rates. After collecting and calculating the metrics, P4BS uses Bayesian Optimization to optimize a blackbox function that combines the packet losses and the queueing delays. P4BS was implemented on a Tofino hardware switch, and the experiments were executed over a Juniper router with configurable buffer size. The tests were executed on a wide range of network scenarios (different numbers of flows, bottleneck link capacities, CCAs, and traffic types), and the results show improvements regardless of the test parameters. The authors plan to explore dynamic BO in the future to avoid restarting the search whenever the conditions significantly change.

Acknowledgment

This work was supported by the U.S. National Science Foundation under grant number 2118311. The authors would like to thank Mariam Kiran from the Scientific Networking Division, Lawrence Berkeley National Laboratory, for providing the dataset used in Section VI-G.

References

[1] Geoff Huston. Sizing the buffer, apnic. https://blog.apnic.net/2019/12/12/sizing-the-buffer/, 2019.
[2] Neda Beheshti, Petr Lapukhov, and Yashar Ganjali. Buffer sizing experiments at Facebook. In *Proceedings of the 2019 Workshop on Buffer Sizing*, pages 1–6, 2019.
[3] Bruce Spang, Brady Walsh, Te-Yuan Huang, Tom Rusnock, Joe Lawrence, and Nick McKeown. Buffer sizing and Video QoE Measurements at Netflix. In *Proceedings of the 2019 Workshop on Buffer Sizing*, pages 1–7, 2019.
[4] Neda Beheshti, Yashar Ganjali, Monia Ghobadi, Nick McKeown, and Geoff Salmon. Experimental study of router buffer sizing. In *Proceedings of the 8th ACM SIGCOMM conference on Internet measurement*, pages 197–210, 2008.
[5] Curtis Villamizar and Cheng Song. High performance TCP in ANSNET. *ACM SIGCOMM Computer Communication Review*, 24(5):45–60, 1994.
[6] Guido Appenzeller, Isaac Keslassy, and Nick McKeown. Sizing router buffers. *ACM SIGCOMM Computer Communication Review*, 34(4):281–292, 2004.
[7] Amogh Dhamdhere and Constantine Dovrolis. Open issues in router buffer sizing. *ACM SIGCOMM Computer Communication Review*, 36(1):87–92, 2006.
[8] Arun Vishwanath, Vijay Sivaraman, and Marina Thottan. Perspectives on router buffer sizing: Recent results and open problems. *ACM SIGCOMM Computer Communication Review*, 39(2):34–39, 2009.
[9] Nick McKeown, Guido Appenzeller, and Isaac Keslassy. Sizing router buffers (redux). *ACM SIGCOMM Computer Communication Review*, 49(5):69–74, 2019.

[10] Konstantin Avrachenkov, Urtzi Ayesta, and Alexei Piunovskiy. Optimal choice of the buffer size in the Internet routers. In *Proceedings of the 44th IEEE Conference on Decision and Control*, pages 1143–1148. IEEE, 2005.

[11] Mihaela Enachescu, Yashar Ganjali, Ashish Goel, Nick McKeown, and Tim Roughgarden. Routers with very small buffers. In *INFOCOM*, 2006.

[12] Ravi S Prasad, Constantine Dovrolis, and Marina Thottan. Router buffer sizing revisited: the role of the output/input capacity ratio. In *Proceedings of the 2007 ACM CoNEXT conference*, pages 1–12, 2007.

[13] Amogh Dhamdhere, Hao Jiang, and Constantinos Dovrolis. Buffer sizing for congested Internet links. In *Proceedings IEEE 24th Annual Joint Conference of the IEEE Computer and Communications Societies.*, volume 2, pages 1072–1083. IEEE, 2005.

[14] Ayush Mishra, Xiangpeng Sun, Atishya Jain, Sameer Pande, Raj Joshi, and Ben Leong. The great internet TCP congestion control census. *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, 3(3):1–24, 2019.

[15] Bruce Spang, Serhat Arslan, and Nick McKeown. Updating the theory of buffer sizing. *Performance Evaluation*, 151:102232, 2021.

[16] Neal Cardwell, Yuchung Cheng, C Stephen Gunn, Soheil Hassas Yeganeh, and Van Jacobson. BBR: Congestion-Based Congestion Control: Measuring bottleneck bandwidth and round-trip propagation time. *Queue*, 14(5):20–53, 2016.

[17] Bruce Spang and Nick McKeown. On estimating the number of flows. In *Stanford Workshop on Buffer Sizing*, 2019.

[18] Yuliang Li, Rui Miao, Hongqiang Harry Liu, Yan Zhuang, Fei Feng, Lingbo Tang, Zheng Cao, Ming Zhang, Frank Kelly, Mohammad Alizadeh, et al. HPCC: high precision congestion control. In *Proceedings of the ACM Special Interest Group on Data Communication*, pages 44–58. 2019.

[19] Arpit Gupta, Rob Harrison, Marco Canini, Nick Feamster, Jennifer Rexford, and Walter Willinger. Sonata: Query-driven streaming network telemetry. In *Proceedings of the 2018 conference of the ACM special interest group on data communication*, pages 357–371, 2018.

[20] Naga Katta, Mukesh Hira, Changhoon Kim, Anirudh Sivaraman, and Jennifer Rexford. Hula: Scalable load balancing using programmable data planes. In *Proceedings of the Symposium on SDN Research*, pages 1–12, 2016.

[21] Xin Jin, Xiaozhou Li, Haoyu Zhang, Robert Soulé, Jeongkeun Lee, Nate Foster, Changhoon Kim, and Ion Stoica. Netcache: Balancing key-value stores with fast in-network caching. In *Proceedings of the 26th Symposium on Operating Systems Principles*, pages 121–136, 2017.

[22] Amedeo Sapio, Marco Canini, Chen-Yu Ho, Jacob Nelson, Panos Kalnis, Changhoon Kim, Arvind Krishnamurthy, Masoud Moshref, Dan RK Ports, and Peter Richtárik. Scaling distributed machine learning with in-network aggregation. *arXiv preprint arXiv:1903.06701*, 2019.

[23] Elie F Kfoury, Jorge Crichigno, and Elias Bou-Harb. An exhaustive survey on P4 programmable data plane switches: Taxonomy, applications, challenges, and future trends. *IEEE Access*, 9:87094–87155, 2021.

[24] Mojgan Ghasemi, Theophilus Benson, and Jennifer Rexford. Dapper: Data plane performance diagnosis of TCP. In *Proceedings of the Symposium on SDN Research*, pages 61–74, 2017.

[25] Xiaoqi Chen, Hyojoon Kim, Javed M Aman, Willie Chang, Mack Lee, and Jennifer Rexford. Measuring TCP round-trip time in the data plane. In *Proceedings of the Workshop on Secure Programmable Network Infrastructure*, pages 35–41, 2020.

[26] Neda Beheshti, Emily Burmeister, Yashar Ganjali, John E Bowers, Daniel J Blumenthal, and Nick McKeown. Optical packet buffers for backbone Internet routers. *IEEE/ACM Transactions on Networking*, 18(5):1599–1609, 2010.

[27] Kathleen Nichols and Van Jacobson. Controlling queue delay. *Communications of the ACM*, 55(7):42–50, 2012.

[28] Koen De Schepper, Olga Bondarenko, Ing-Jyh Tsang, and Bob Briscoe. Pi2: A linearized AQM for both classic and scalable TCP. In *Proceedings of the 12th International on Conference on emerging Networking EXperiments and Technologies*, pages 105–119, 2016.

[29] Olga Albisser, Koen De Schepper, Bob Briscoe, Olivier Tilmans, and Henrik Steen. DUALPI2-low latency, low loss and scalable (L4S) AQM. *Proc. Netdev 0x13 (Mar. 2019)*, 2019.

[30] Robert Morris. Scalable TCP congestion control. In *Proceedings IEEE INFOCOM 2000. Conference on Computer Communications. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies (Cat. No. 00CH37064)*, volume 3, pages 1176–1183. IEEE, 2000.

[31] Matthew Mathis, Jeffrey Semke, Jamshid Mahdavi, and Teunis Ott. The macroscopic behavior of the TCP congestion avoidance algorithm. *ACM SIGCOMM Computer Communication Review*, 27(3):67–82, 1997.

[32] Rade Stanojevic, Robert N Shorten, and Christopher M Kellett. Adaptive tuning of drop-tail buffers for reducing queueing delays. *IEEE Communications Letters*, 10(7):570–572, 2006.

[33] Yueping Zhang and Dmitri Loguinov. Abs: Adaptive buffer sizing for heterogeneous networks. *Computer Networks*, 14(54):2562–2574, 2010.

[34] Elie Kfoury, Jorge Crichigno, Elias Bou-Harb, and Gautam Srivastava. Dynamic router's buffer sizing using passive measurements and P4 programmable switches. In *2021 IEEE Global Communications Conference (GLOBECOM)*, pages 01–06. IEEE, 2021.

[35] Elie F Kfoury, Jose Gomez, Jorge Crichigno, and Elias Bou-Harb. An emulation-based evaluation of TCP BBRv2 alpha for wired broadband. *Computer Communications*, 161:212–224, 2020.

[36] Jorge Crichigno, Elie Kfoury, Elias Bou-Harb, Nasir Ghani, Yasmany Prieto, Christian Vega, J Pezoa, C Huang, and David Torres. A flow-based entropy characterization of a NATed network and its application on intrusion detection. In *ICC 2019-2019 IEEE International Conference on Communications (ICC)*, pages 1–7. IEEE, 2019.

[37] Joel Sommers, Paul Barford, Albert Greenberg, and Walter Willinger. An SLA perspective on the router buffer sizing problem. *ACM SIGMETRICS Performance Evaluation Review*, 35(4):40–51, 2008.

[38] Chuanxiong Guo, Lihua Yuan, Dong Xiang, Yingnong Dang, Ray Huang, Dave Maltz, Zhaoyi Liu, Vin Wang, Bin Pang, Hua Chen, et al. Pingmesh: A large-scale system for data center network latency measurement and analysis. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*, pages 139–152, 2015.

[39] Constantine Dovrolis, Krishna Gummadi, Aleksandar Kuzmanovic, and Sascha D Meinrath. Measurement lab: Overview and an invitation to the research community. *ACM SIGCOMM Computer Communication Review*, 40(3):53–56, 2010.

[40] Cheng Tan, Ze Jin, Chuanxiong Guo, Tianrong Zhang, Haitao Wu, Karl Deng, Dongming Bi, and Dong Xiang. Netbouncer: Active device and link failure localization in data center networks. In *NSDI*, pages 599–614, 2019.

[41] Andreas Hanemann, Jeff W Boote, Eric L Boyd, Jérôme Durand, Loukik Kudarimoti, Roman Łapacz, D Martin Swany, Szymon Trocha, and Jason Zurawski. Perfsonar: A service oriented architecture for multi-domain network monitoring. In *Service-Oriented Computing-ICSOC 2005: Third International Conference, Amsterdam, The Netherlands, December 12-15, 2005. Proceedings 3*, pages 241–254. Springer, 2005.

[42] Richard Cziva, Christopher Lorier, and Dimitrios P Pezaros. Ruru: High-speed, flow-level latency measurement and visualization of live internet traffic. In *Proceedings of the SIGCOMM Posters and Demos*, pages 46–47. 2017.

[43] Bryan Veal, Kang Li, and David Lowenthal. New methods for passive estimation of TCP round-trip times. In *Passive and Active Network Measurement: 6th International Workshop, PAM 2005, Boston, MA, USA, March 31-April 1, 2005. Proceedings 6*, pages 121–134. Springer, 2005.

[44] Jian Zhang and Andrew Moore. Traffic trace artifacts due to monitoring via port mirroring. In *2007 Workshop on End-to-End Monitoring Techniques and Services*, Munich, Germany, May, 2007.

[45] Vibhaalakshmi Sivaraman, Srinivas Narayana, Ori Rottenstreich, Shan Muthukrishnan, and Jennifer Rexford. Heavy-hitter detection entirely in the data plane. In *Proceedings of the Symposium on SDN Research*, pages 164–176, 2017.

[46] Xiaoqi Chen, Shir Landau-Feibish, Mark Braverman, and Jennifer Rexford. BeauCoup: answering many network traffic queries, one memory update at a time. In *Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication*, pages 226–239, 2020.

[47] Graham Cormode and Shan Muthukrishnan. An improved data stream summary: the count-min sketch and its applications. *Journal of Algorithms*, 55(1):58–75, 2005.

[48] Maria Apostolaki, Ankit Singla, and Laurent Vanbever. Performance-driven internet path selection. In *Proceedings of the ACM SIGCOMM Symposium on SDN Research (SOSR)*, pages 41–53, 2021.

[49] Van Jacobson. Congestion avoidance and control. *ACM SIGCOMM computer communication review*, 18(4):314–329, 1988.

[50] Singiresu S Rao. *Engineering optimization: theory and practice*. John Wiley & Sons, 2019.

[51] Peter I Frazier. A tutorial on bayesian optimization. *arXiv preprint arXiv:1807.02811*, 2018.

[52] Bertil Matérn. *Spatial variation*, volume 36. Springer Science & Business Media, 2013.

[53] Jonas Močkus. On bayesian methods for seeking the extremum. In *Optimization techniques IFIP technical conference*, pages 400–404. Springer, 1975.
[54] Lorenzo Maggi, Alvaro Valcarce, and Jakob Hoydis. Bayesian optimization for radio resource management: Open loop power control. *IEEE Journal on Selected Areas in Communications*, 2021.
[55] Favour M Nyikosa, Michael A Osborne, and Stephen J Roberts. Bayesian optimization for dynamic problems. *arXiv preprint arXiv:1803.03432*, 2018.
[56] Ben Pfaff, Justin Pettit, Teemu Koponen, Ethan Jackson, Andy Zhou, Jarno Rajahalme, Jesse Gross, Alex Wang, Joe Stringer, Pravin Shelar, et al. The design and implementation of open vswitch. In *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15)*, Oakland, CA, USA, May, 2015.
[57] Wedge 100BF-32X, 100GbE data center switch, Barefoot Networks, an Intel® company. [Online], Available: https://tinyurl.com/sy2jkqe.
[58] Stephen Hemminger et al. Network emulation with NetEm. In *Linux conf au*, volume 5, page 2005. Citeseer, 2005.
[59] Vu Nguyen, Sunil Gupta, Santu Rana, Cheng Li, and Svetha Venkatesh. Regret for expected improvement over the best-observed value and stopping condition. In *Asian Conference on Machine Learning*, pages 279–294. PMLR, 2017.
[60] Oliver Hohlfeld, Enric Pujol, Florin Ciucu, Anja Feldmann, and Paul Barford. A QoE perspective on sizing network buffers. In *Proceedings of the 2014 Conference on Internet Measurement Conference*, pages 333–346, 2014.
[61] International Telecommunication Union (ITU). ITU-T Recommendation P.862: Perceptual Evaluation of Speech Quality (PESQ): An objective method for end-to-end speech quality assessment of narrow-band telephone networks and speech codecs. [Online], Available: https://www.itu.int/rec/T-REC-P.862.
[62] Lingfen Sun. Speech quality prediction for voice over internet protocol networks. 2004.
[63] International Telecommunication Union (ITU). G.114 : One-way transmission time. [Online], Available: https://www.itu.int/rec/T-REC-G.114-200305-I/en.
[64] International Telecommunication Union (ITU). ITU-T Rec. P.862 annex a: Reference implementations and conformance testing for ITU-T Recs P.862, P.862.1 a.P.862.2. [Online], Available: https://www.itu.int/rec/T-REC-P.862.
[65] R. Gayraud and O. Jacques. Sipp 2.0 reference documentation.
[66] Te-Yuan Huang, Nikhil Handigol, Brandon Heller, Nick McKeown, and Ramesh Johari. Confused, timid, and unstable: picking a video streaming rate is hard. In *Proceedings of the 2012 internet measurement conference*, pages 225–238, 2012.
[67] Jakob Tideström. Investigation into low latency live video streaming performance of WebRTC, 2019.
[68] Ton Roosendaal. Big buck bunny. In *ACM SIGGRAPH ASIA 2008 Computer Animation Festival*, pages 62–62. 2008.
[69] Anonymized Internet Traces 2019. https://catalog.caida.org/details/dataset/passive_2019_pcap. Accessed: 2022-5-13.
[70] MAWI Working Group Traffic Archive. https://mawi.wide.ad.jp/mawi/. Accessed: 2022-5-13.
[71] Energy Science Networks (ESnet), Router/switch buffer size issues. https://fasterdata.es.net/network-tuning/router-switch-buffer-size-issues/. Accessed: 2022-5-24.
[72] Jorge Crichigno, Elias Bou-Harb, and Nasir Ghani. A comprehensive tutorial on science DMZ. *IEEE Communications Surveys & Tutorials*, 21(2):2041–2078, 2018.
[73] Energy Science Networks (ESnet), ESnet Data Transfer Nodes. https://fasterdata.es.net/performance-testing/DTNs/. Accessed: 2022-5-24.
[74] Globus, Research data management simplified. https://www.globus.org/. Accessed: 2022-5-24.
[75] Hyojoon Kim, Xiaoqi Chen, Jack Brassil, and Jennifer Rexford. Experience-driven research on programmable networks. *ACM SIGCOMM Computer Communication Review*, 51(1):10–17, 2021.
[76] P4 campus: P4 applications for campus networks. https://p4campus.cs.princeton.edu/. Accessed: 2023-5-26.
[77] Oluwamayowa Ade Adeleke, Nicholas Bastin, and Deniz Gurkan. Network traffic generation: A survey and methodology. *ACM Computing Surveys (CSUR)*, 55(2):1–23, 2022.
[78] Joel Sommers, Hyungsuk Kim, and Paul Barford. Harpoon: a flow-level traffic generator for router and network tests. *ACM SIGMETRICS Performance Evaluation Review*, 32(1):392–392, 2004.
[79] Kashi Venkatesh Vishwanath and Amin Vahdat. Swing: Realistic and responsive network traffic generation. *IEEE/ACM Transactions on Networking*, 17(3):712–725, 2009.
[80] Intel® Tofino 3.2 Tbps, 4 pipelines. [Online], Available: https://tinyurl.com/yc34ukpa.
[81] Wei Bai, Shuihai Hu, Kai Chen, Kun Tan, and Yongqiang Xiong. One more config is enough: Saving (DC) TCP for high-speed extremely shallow-buffered datacenters. *IEEE/ACM Transactions on Networking*, 29(2):489–502, 2020.
[82] Wenjun Lyu, Jiawei Huang, Jingling Liu, Zhaoyi Li, Shaojun Zou, Weihe Li, Jianxin Wang, and Desheng Zhang. Mitigating port starvation for shallow-buffered switches in datacenter networks. In *2021 IEEE 41st International Conference on Distributed Computing Systems (ICDCS)*, pages 921–931. IEEE, 2021.
[83] Intel® Infrastructure Processing Unit (Intel® IPU). [Online], Available: https://www.intel.com/content/www/us/en/products/details/network-io/ipu.html.
[84] E. Kfoury, J. Crichigno, and Bou-Harb. P4CCI: P4-based online TCP congestion control algorithm identification for traffic separation. In *IEEE International Conference on Communications (ICC)*. IEEE, 2023.
[85] Gajendra Hari Prakash Theagarajan, Sivakumar Ravichandran, and Vijay Sivaraman. An experimental study of router buffer sizing for mixed TCP and real-time traffic. In *2006 14th IEEE International Conference on Networks*, volume 1, pages 1–6. IEEE, 2006.
[86] Charles Audet and Warren Hare. Derivative-free and blackbox optimization. 2017.
[87] Donald R Jones, Matthias Schonlau, and William J Welch. Efficient global optimization of expensive black-box functions. *Journal of Global optimization*, 13(4):455, 1998.

APPENDIX

A. Relationship between buffer and loss/delay

There is an intrinsic relationship between the buffer size and the packet loss and queueing delay. Large buffers reduce packet losses but increase queue delays. On the other hand, small buffers increase the losses but decrease the queueing delays. The reduction/increase rates depend on various factors. Consider Fig. 17 where 100 long-lived flows with a smoothed RTT (SRTT) $\approx$ 20ms and different CCAs are sharing a 1Gbps bottleneck link. The increase/decrease rates of the packet loss and the RTT (which is largely dominated by the queueing delays) depend on the CCA being used. In fact, there are many other factors that affect those rates; for instance, Morris showed that the loss rate increases almost with the square of the number of competing flows (N) [30]. Other factors that affect the loss/queueing rates include the heterogeneity of the flows' RTTs (*e.g.*, flows with long RTTs require larger buffers, increasing the queueing delays for flows with short RTTs) [84], the output/input capacity ratio at a network link (e.g., with a ratio greater than one, no queue will be occupied and no packets are dropped) [12], the traffic type (*e.g.*, bursty

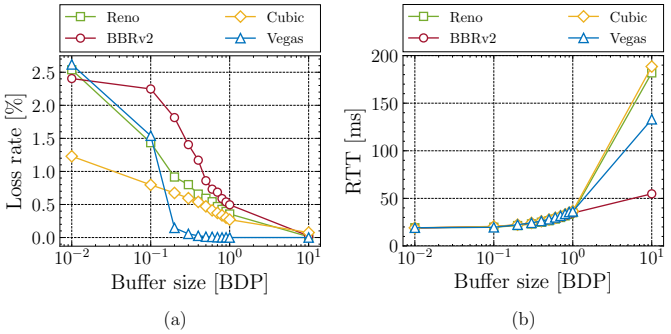

Fig. 17. Impact of buffer size on the packet loss rate (a) and on the RTT (b). The loss decreases and the RTT increases as the buffer size increases. Test parameters: $N = 100$; $C = 1Gbps$; $SRTT = 20ms$.

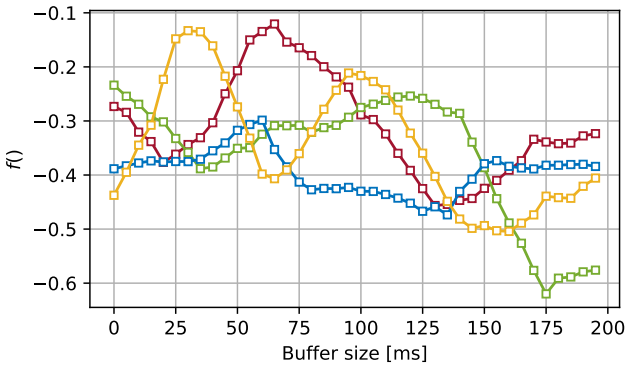

Fig. 18. Four performance functions from different times in the CAIDA trace. These functions were evaluated over a range of buffer sizes, starting from small to large, as part of the sampling process. The functions are multimodal.

real-time traffic can interact with well-behaved TCP traffic, increasing packet losses) [85], etc.

B. Multimodal performance function

Given the black-box nature of the performance function, where there is limited analytical understanding of its behavior, providing formal proof of its multimodality becomes challenging [86]. Consequently, a sampling-based approach is employed [87]. By sampling the performance function at various points, it becomes possible to learn its true values and observe its behavior. Through this sampling process, it can be determined whether the function possesses multiple maxima or not.

Fig. 18 illustrates four performance functions obtained from different times in the CAIDA trace. These functions were evaluated over a range of buffer sizes, starting from small to large, as part of the sampling process. It is evident from the figure that these functions have multiple maxima.

C. Metrics estimation scalability

P4BS declares 4 registers arrays ($d = 4$), each has $2^{17} = 131072$ cells ($w = 131072$). The CMS is reset every 10 seconds (i.e., all cells will have 0).

Consider a real traffic trace from CAIDA: CAIDA's Equinix 10G dataset [69]. The router has a 10Gbps bottleneck link capacity. Fig. 19 shows the number of packets per 10 seconds for various parts of the CAIDA trace. The number of packets is up to ≈ 7 million.

With $d = 4$, $\delta = \frac{1}{e^{d-1}} \approx 5\%$; with $w = 2^{17}$, $\epsilon = \frac{e}{w-1} \approx 0.00002$. Every 10 seconds, after counting ≈ 7 million packets in total, the probability for any estimate to be off by more than $0.00002 * 7,000,000 \approx 140$ packets is less than 5%. This number is much smaller than the threshold used to identify a flow as being long. Thus, even with multiple bottlenecked routers, the proposed system can still reliably estimate the packet counts.

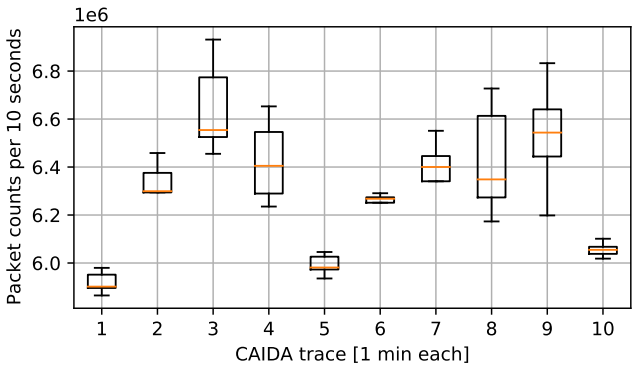

Fig. 19. Number of packets per 10 seconds for various parts of the CAIDA trace. The number of packets is up to ≈ 7 million. Using the CMS, the probability of overestimating the counts by 140 packets is less than 5%.

Elie Kfoury received the Ph.D. degree in Informatics from the University of South Carolina (USC), in 2023. He is currently an assistant professor in the Integrated Information Technology department at USC. As a member of the Cyberinfrastructure Laboratory, he developed training materials using virtual labs on high-speed networks, TCP congestion control, programmable switches, SDN, and cybersecurity. His research interests include P4 programmable data planes, computer networks, cybersecurity, and Blockchain. He previously worked as a research and teaching assistant in the computer science department at the American University of Science and Technology in Beirut.

Jorge Crichigno received the Ph.D. degree in computer engineering from The University of New Mexico, Albuquerque, USA, in 2009. He is a Professor with the College of Engineering and Computing, University of South Carolina (USC), and the Director of the Cyberinfrastructure Laboratory, USC. His work has been funded by private industry and U.S. agencies such as the National Science Foundation (NSF), the Department of Energy, and the Office of Naval Research (ONR). He has over 15 years of experience in the academic and industry sectors. His research interests include P4 programmable switches, implementation of high-speed networks, network security, TCP optimization, offloading functionality to programmable switches, and IoT devices.

Elias Bou-Harb (Senior Member, IEEE) received his postdoctoral training at Carnegie Mellon University and his Ph.D. degree in computer science from Concordia University, Montreal, Canada. He is currently a tenured associate professor with the department of computer science at Lousiana State University, specializing in cyber security and data science as applicable to national security challenges. Previously, he acted as the director of the cyber center for security and analytics at the University of Texas at San Antonio, where he led and organized university-wide cyber security research, development, and training initiatives. Dr. Bou-Harb has authored more than 150 refereed publications in leading venues and has acquired significant state and federal cyber security research grants. His research and development activities focus on operational cyber security, cyber forensics, critical infrastructure security, empirical data analytics, digital investigations, network security, and network management. He is the recipient of five best research paper awards, including the ACM's best digital forensics research paper.