P4CCI: P4-based Online TCP Congestion Control Algorithm Identification for Traffic Separation

Elie Kfoury
College of Engineering and Computing
University of South Carolina, USA

Jorge Crichigno
College of Engineering and Computing
University of South Carolina, USA

Elias Bou-Harb
Cyber Center For Security and Analytics
University of Texas at San Antonio, USA

***Abstract*—Congestion Control Algorithms (CCAs) regulate the sending rates of hosts to avoid congestion in the network. Studies have shown that when flows belonging to different CCAs co-exist on the same link, their shares on that link are significantly different. If the CCAs of active flows can be determined on live traffic, then flows belonging to the same CCA can be allocated into a dedicated queue. Unfortunately, identifying the CCA at line rate is not straightforward since the CCA is not advertised in the header fields of a packet. Moreover, with Gigabits per second (Gbps) traffic crossing a network, analyzing each packet to infer the CCA is not possible, especially with general-purpose CPUs. This paper proposes P4CCI, a system that detects the CCA of a flow at line rate by leveraging Programmable Data Planes (PDP). The PDP computes and extracts the flow's bytes-in-flight and sends them to a Deep Learning model for classification. Once classified, the flows are allocated into dedicated queues based on their CCA type. The system was implemented and tested on real hardware that uses Intel's Tofino ASIC. The experiments were executed on traffic provided by CAIDA. Results show that P4CCI can detect the CCAs with high accuracy. Furthermore, the performance of the network is greatly improved when the flows are separated by their CCAs.**

***Index Terms*—Programmable data plane, P4, TCP, congestion control algorithm, Deep Learning.**

I. Introduction

TCP Congestion Control Algorithms (CCAs) attempt to determine the available capacity in the network in order to regulate the sending rates of hosts and avoid congestion. Research in congestion control has significantly evolved throughout the years; early versions of TCP use the Additive Increase Multiplicative Decrease (AIMD) control law. In AIMD, the congestion window is increased by approximately one Maximum Segment Size (MSS) every Round-trip Time (RTT) when the data is acknowledged, and halved for every window of data containing a packet drop. This CCA variant is known as "Reno", and is considered loss-based since it uses the packet loss as a signal of congestion. Since then, many enhancements have been proposed [1], [2]. The most common CCA is "CUBIC", which uses a cubic function for the window growth. CUBIC is currently the default CCA used in Linux Operating Systems (OS).

In 2016, Google released the first version of the Bottleneck Bandwidth and Round-trip Time (BBR) algorithm [3]. The design of BBR diverged from the existing CCAs in that it does not adhere to the AIMD rule. Instead, it measures the bottleneck bandwidth and the RTT, and uses packet pacing to set the sending rate.

Recent studies have shown that CUBIC is currently the dominant TCP variant on the Internet, followed by BBR [4]. Unfortunately, when flows using these two variants co-exist on a link, the fairness (i.e., how fair is the capacity of the link being divided among the competing flows) is significantly low [5], [6]. On the other hand, when flows belonging to the same CCA share a bottleneck link, the fairness is typically high.

A potential solution to mitigate the fairness issue is to separate flows into different queues on the router based on their CCAs. Unfortunately, the CCA used by a flow is not advertised, and it is not straightforward to identify it in real time, especially with high traffic rates. Existing works on CCA identification perform offline analysis on packet captures to extract unique features manifested by a specific CCA. Such works aim to discover the distribution of CCAs on the Internet rather than separating the traffic into different queues.

A. Contributions

This paper presents P4CCI, a system that leverages P4 Programmable Data Planes (PDPs) to identify CCAs at line rate. After identification, the flows are assigned to separate queues based on the CCAs they are using. The system does not require deploying PDPs inline, and thus, it is compatible with non-programmable data planes found in most networks nowadays. The contributions of this paper are:

- Devising a scheme that relies on passive PDPs for CCA identification at line rate. The PDP measures the average queueing and computes the bytes-in-flight (BIF) values for the flows during congestion.
- Implementing a Deep Learning model to identify the CCA. The model classifies the CCA using the flow's BIF values.
- Separating the flows belonging to the same CCA into dedicated queues. This separation improves the fairness of TCP flows sharing the bottleneck and minimizing the Flow Completion Times (FCTs) of short flows.

The rest of the paper is organized as follows. Section II describes the existing CCA identification solutions and motivates the need for P4CCI. Section III provides an overview of P4CCI. Section IV describes the experimentation setup and discusses the results obtained with real hardware. Finally,

This work was supported by the U.S. National Science Foundation under grant number 2118311.

Section V concludes the paper and highlights potential future work.

II. Background and Related Work

A. Programmable Data Planes Primer

PDPs allows the programmer to customize the packet processing pipeline. PDPs include a programmable parser, programmable match-action pipeline, and programmable deparser. The programmable parser permits the programmer to define the headers according to custom or standard protocols, and to parse them. The match-action pipeline executes operations over the packet headers and intermediate results. The deparser assembles the packet headers back and serializes them for transmission. PDPs provide high-precision timers (nanosecond granularity [7]), and stateful memories (e.g., registers, counters, meters) that can be accessed at line rate. PDPs have been extensively used for improving the performance of networks [8], [9].

B. Active Identification

Active approaches instantiate connections to a host, request data, manipulate the communication (e.g., force packet drops, introduce latency), and observe the behavior of TCP. TBIT [10] traces the congestion window; CAAI [11] extracts the multiplicative-decrease parameter and the window growth function, then uses ML for the classification; Gordon [4] estimates the congestion window; Inspector Gadget (IG) enhances CAAI and considers more complex network environments with changing RTTs. P4Air [12] uses P4 switches and tracks the queue buildup to identify CCAs. Unlike the other approaches, P4Air does not instantiate new connections to the host, but forces packet drops and introduces latency to the existing traffic. Furthermore, P4Air requires deploying the PDPs inline, and thus, cannot be used with contemporary non-programmable devices.

C. Passive Identification

Passive approaches do not interact with the destination host, but rely on previously collected packet traces. Oshio et al. [13] used cluster analysis considering the congestion window and the RTT. Kato et al. [14] approximate the sequence number (SEQ) to time function and use derivatives to differentiate between the CCAs. DeePCCI [15] uses the packet arrival time for classifying the CCA.

The proposed approach (P4CCI) identifies the CCA passively, but instead of using packet captures for analysis, it operates on live traffic. P4CCI computes BIF samples at line rate, and feeds those samples to a Deep Learning model for classification. Another distinction between the existing work and P4CCI is that the goal of P4CCI is to separate traffic by their CCAs rather than studying the distribution of CCAs on public nodes in the wide area network.

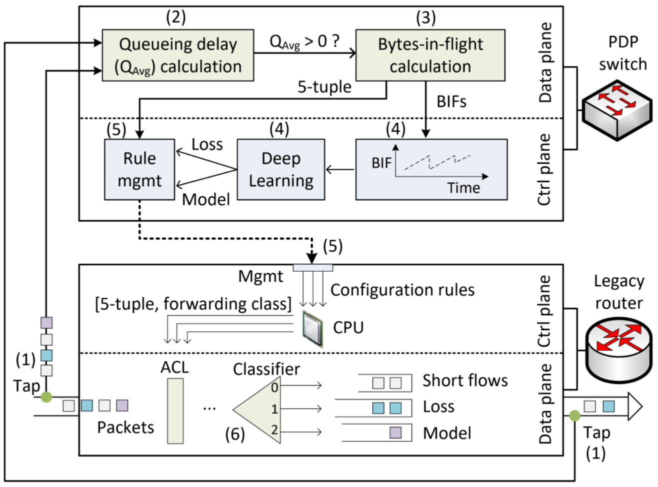


Fig. 1. P4CCI architecture.

III. Proposed System

A. Overview

Consider Fig. 1 which demonstrates the architecture of P4CCI. The steps to identify the CCA and separate the traffic are as follows:

1) The system continuously and passively monitors the traffic traversing a legacy router. Passive Tap devices are installed on the links of the router, and the traffic is forwarded to a customized packet processing pipeline in the PDP switch that operates at line rate.
2) Since the CCAs only exhibit different behaviors during congestion (i.e., the router's queue is being utilized), the data plane of the PDP switch continuously measures the average queueing delay.
3) Upon detecting congestion, the BIF samples for each flow are computed. The BIF samples are then pushed to the control plane of the PDP switch.
4) The control plane organizes the per-flow BIF samples into a time series, and feeds them into a Deep Learning model to classify the flow's CCA.
5) Once classified, a rule is created and pushed to the control plane of the legacy router. The rule consists of the 5-tuple of the flow, and the CCA it belongs to. The rule is configured as an Access Control List (ACL) entry matching on the 5-tuple of the flow.
6) The scheduler of the legacy router assigns the packets to a corresponding queue based on the flow's predicted CCA (i.e., Loss for loss-based CCA such as CUBIC or Reno; Model for model-based CCA such as BBR). The short flows queue is the default queue used by all flows.

B. Queueing Delay Calculation

Consider Fig. 2 which shows the queueing delay calculation process. (1) Upon receiving a packet from `Tap1`, the PDP hashes the packet's 5-tuple fields (i.e., the flow identifier) along with the packet's SEQ. The resulting hash value is used as an index to a register array where the switch's current

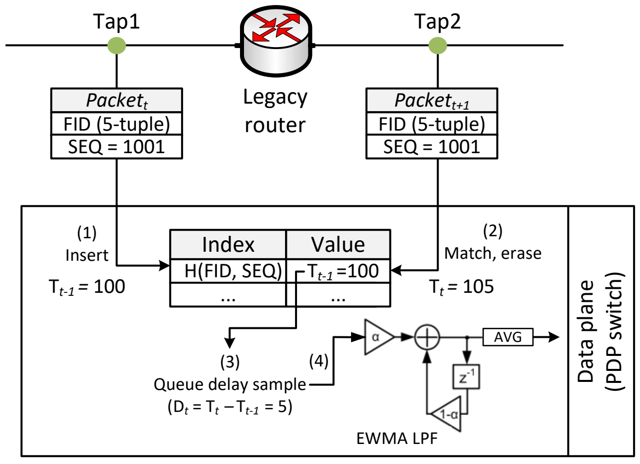


Fig. 2. Queue delay calculation in the data plane.

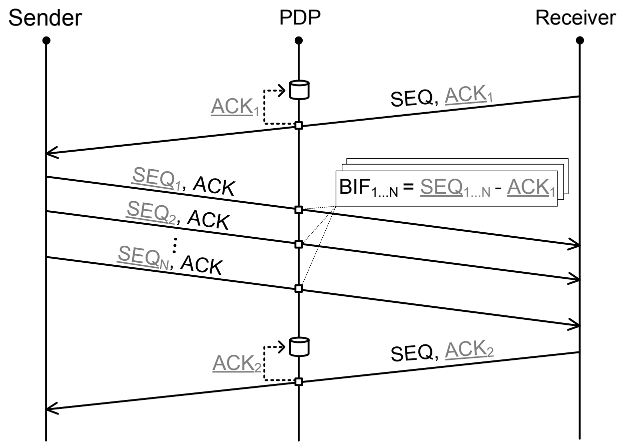


Fig. 3. Bytes-in-flight (BIF) calculation.

timestamp is stored. (2) When a packet arrives from `Tap2`, the data plane computes its hash (i.e., Hash (5-tuple, SEQ)) to retrieve the previous timestamp stored in the array. This timestamp is then removed from the array. (3) The data plane computes a queueing delay sample as $D_t = T_t - T_{t-1}$, where T_t is the switch's current timestamp and T_{t-1} is the extracted timestamp. (4) The queueing delay sample is then fed to an Exponentially Weighted Moving Average (EWMA) Low Pass Filter (LPF) to compute the average.

C. Bytes-in-flight Calculation

BIF is the amount of data that has been sent but not yet acknowledged. This field is always less than or equal to the recipient's receive window. The calculation of BIF is shown in Fig. 3. The PDP stores the last Acknowledgement number (ACK) for a given TCP flow in a register array. The flow's current BIF (in bytes) is calculated as $BIF_i = SEQ_i - ACK$, where SEQ_i is the sequence number of a data packet belonging to the flow, and ACK is the flow's last ACK. The BIF value is then pushed to the control plane of the PDP switch where a time series is constructed. Note that the BIFs values are only pushed during congestion (i.e., when the queue is being utilized). This condition is essential to avoid saturating the control/data plane channel, and to ensure that the CCAs will exhibit different behaviors.

Since calculating BIF requires examining all packets crossing the device, it is not possible to estimate BIFs on a general purpose CPU, especially when the traffic rates are high (in the order of Gbps or Tbps).

D. Time Series Preparation

Two steps are applied to prepare the time series to be used for classification: 1) outliers rejection and 2) normalization.

Outliers Rejection. The robust Z-score method [16], which uses the MAD (Median Absolute Deviation), is used to reject the outliers from the time series. MAD is calculated as:

$$MAD = med\{|x_i - \hat{x}|\}, \tag{1}$$

where x_i is a sample point and $\hat{x}$ is the median of the samples. Then the z-score is calculated as:

$$M_i = \frac{0.6745(x_i - \hat{x})}{MAD} \tag{2}$$

If the score of a point is above 3.5 (the rule-of-thumb cut-off value), the point is considered as an outlier and is removed from the series.

Normalization The time series is preprocessed using z-normalization, where the input vector is transformed into an output vector whose mean is approximately 0 and its standard deviation is in a range close to 1. The transformation is as follows: $x'_i = \frac{x_i - \mu}{\sigma}$, where x_i is a sample point in the time series, μ is the mean of all the values, σ is the standard deviation, and x'_i is the standardized sample point.

E. Deep Learning

Early approaches to Time Series Classification (TSC) consist of using a nearest neighbor (NN) classifier coupled with a distance function such as the Dynamic Time Warping (DTW). More recent approaches use Deep Learning for TSC. Recent studies have shown that Deep Learning outperforms the early NN classifiers [17].

P4CCI uses the Fully Convolutional Neural Networks (FCNs) [18] to classify the univariate time series. FCNs do not use local pooling layers, which makes the length of the time series unchanged throughout the convolutions. The convolution blocks consist of a batch normalization layer and a ReLU activation. Three 1-D kernels with sizes $\{8, 5, 3\}$ are used without striding, and the filter sizes for the convolution blocks are $\{128, 256, 128\}$. In addition, instead of using a fully connected layer after the convolution blocks, FCN uses a global average pooling layer. The output label is produced by a softmax layer.

F. Rule Management

After identifying the CCA for a given flow, the control plane of the PDP switch constructs an ACL entry that matches on the 5-tuple of the flow. This rule is pushed to the control plane

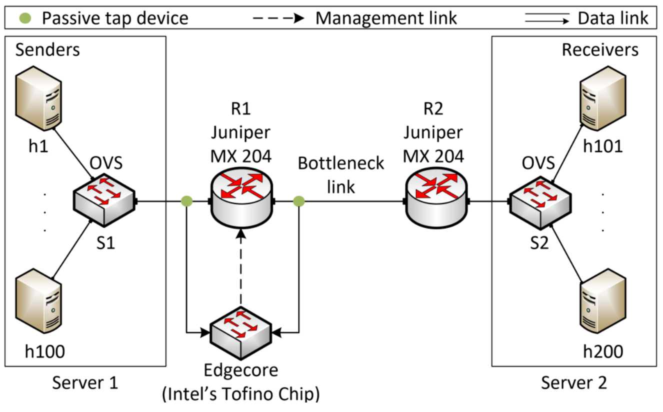


Fig. 4. Topology used for training and testing.

of the legacy router. The rule specifies which queue should be used for the flow. There are three queues: 1) short flows queue: this is the default queue for all flows. Short flows will remain in this queue, while long flows will be allocated in different queues; 2) Loss queue: this queue is used by loss-based CCAs (e.g., CUBIC, Reno); and 3) Model queue: this queue is used by model-based CCAs (e.g., BBR). Note that P4CCI only considers loss-based and model-based CCAs since they represent the majority of the CCAs on the Internet [4].

IV. Experimental Setup and Evaluation

Fig. 4 shows the topology used to train the model and evaluate P4CCI. There are 100 senders (h1, h2, ..., h100) each sending traffic to a corresponding receiver (h101, h102, ..., h200). Mininet was used to emulate the hosts running in network namespaces in Linux. The senders are connected to a virtual switch (Open vSwitch), which is bridged to the physical interface on the server. The server's interface is connected to a Juniper router (MX-204) (R1), and the link's bandwidth is configured depending on the test. The programmable switch uses the Intel's Tofino programmable ASIC that operates at 3.2 Tbps. The end hosts were carefully tuned (e.g., TCP send/receive buffers are set to a large value (200MB)). iPerf3 is the tool used to generate traffic and measure the performance.

When evaluating the system, two scenarios are considered: 1) all flows are mixed in a single queue (i.e., the default behavior of the router), referred to as "w/o separation"; and 2) flows belonging to the same CCA are assigned to the same queue (P4CCI), referred to as "w/ separation".

A. Training

Training the model with realistic traffic is challenging. First, there is a lack of a labeled dataset for CCA identification; the existing works (e.g., P4Air, DeePCCI) generated synthetic data for training and testing. Second, replaying packet traces (e.g., from CAIDA) verbatim will not be useful as the closed-loop dynamics of TCP will not be captured. The P4CCI's model is trained on a real dataset by utilizing the following heuristics:

1) The PCAP file of a CAIDA dataset [19] which contains packet headers derived from 10Gbps traces on Equinix NYC monitor was analyzed. Two PCAP files are considered, each lasting for a minute.
2) The inter-departure times of the long flows and their sizes (the total amount of bytes transferred by the flow) are extracted. A flow is considered long if its size $>$ 10MB.
3) The distribution of the sizes of short flows and their departure times are analyzed.
4) The propagation delay for a flow is configured based on the minimum RTT observed in its lifetime.
5) The long flows are replayed based on their sizes and their starting times; the short flows are reproduced based on the flow size and starting time distributions observed in the traces.
6) The CCAs were assigned to the flows based on the distribution derived from [4]. Another round of training distributed the CCAs evenly (50% CUBIC/Reno, 50% BBR).

TABLE I
Training parameters for the synthetically generated dataset.

Flows	1, 2, 5, 10, 15, 20, 50, 100
Bandwidth [bps]	500M, 1G, 2G, 3G, 4G, 5G, 10G
CCAs	Loss (CUBIC, Reno), Model (BBR)
Packet loss rates [%]	0, 0.1, 0.25, 0.5
Propagation delays [ms]	0, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100
Buffer sizes [ms]	10, 20, 30, 40, 50, 60, 70, 80, 90, 100

In addition to the CAIDA dataset, the model was trained with synthetically generated traffic. The training considered various bottleneck link bandwidths, propagation delays, packet loss rates, router's buffer size, and number of flows. The ranges of these variables are summarized in Table I.

The model was trained with the Adam optimizer, with a learning rate of 0.005. The learning rate was reduced when the optimizer reached a plateau. The loss function is the categorical cross-entropy.

B. Testing

The model was tested against 10 minutes worth of traffic from the remaining CAIDA dataset. The bottleneck bandwidth was configured to 1Gbps, 1.5Gbps, 2Gbps, and 2.5Gbps. The goal is to verify that the model generalizes well, regardless of the bottleneck bandwidth. The model was also tested by generating traffic using a random combination of the parameters used in Table 1. The generated tests contain parameter settings not included in the training (e.g., a bandwidth of 750Mbps,

TABLE II
Classification results.

Dataset	Classes	Precision	Recall	F1-score	Accuracy
CAIDA 1Gbps	Loss	96.2%	93.5%	94.8%	96.1%
	Model	96.0%	97.7%	96.8%	
CAIDA 1.5Gbps	Loss	95.2%	92.0%	93.1%	95%
	Model	95.6%	97.6%	96.6%	
CAIDA 2Gbps	Loss	92.0%	92.5%	92.3%	95.4%
	Model	96.9%	96.4	96.8%	
CAIDA 2.5Gbps	Loss	91.5%	91.0%	91.2%	95.6%
	Model	97.0%	97.1%	97.0%	
Synthetic	Loss	99.2%	99.5%	99.4%	99.4%
	Model	99.5%	99.2%	99.4%	

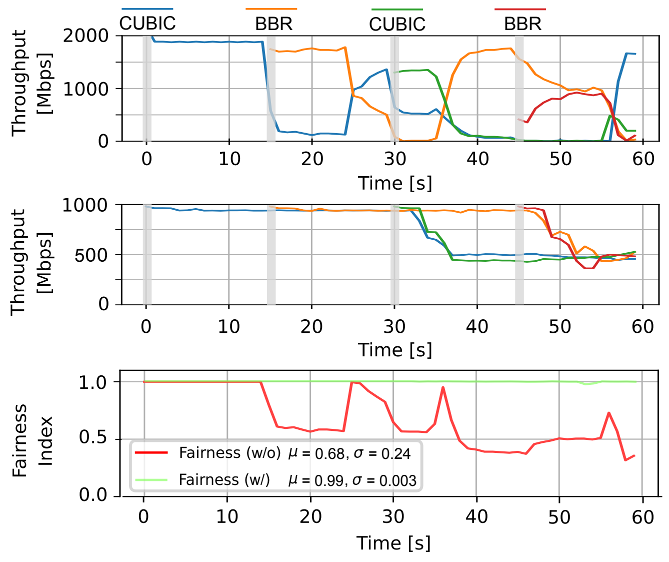


Fig. 5. Per-flow throughput (top: w/o separation; middle: w/ separation) and the fairness index (bottom).

or a loss rate of 0.35). The total number of synthetic tests is 1282, where the CCAs are evenly distributed.

Table II shows the precision, recall, F1-score, and accuracy for each dataset. The model was able to generalize well as the scores are above 95% in the real datasets, and above 99% in the synthetic dataset. Note that the previous works were only evaluated on synthetic datasets. Thus, it is only fair to compare them with P4CCI on synthetic datasets. In such case, P4CCI outperformed all the existing approaches.

C. Fairness Evaluation

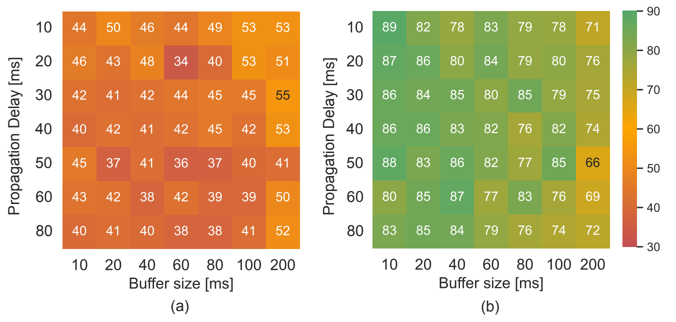
The goal of this test is to measure how fair is the link being shared among flows. Jain's fairness index, described in RFC 5166, is used to quantify fairness. It is computed as follows:

$$\mathcal{F} = \frac{\left(\sum_{i=1}^{n} T_i\right)^2}{n \cdot \sum_{i=1}^{n} (T_i)^2}, \tag{3}$$

where T_i is the throughput of an active flow i among n flows.

Experiment 1. In this experiment, four long flows are generated, 15 seconds apart: flow 1: CUBIC; flow 2: BBR; flow 3: CUBIC; and flow 4: BBR. The bottleneck bandwidth is 2Gbps; in the "w/ separation" scenario, 1Gbps is used for the loss-based CCA queue, and another 1Gbps is used for the model-based CCA queue.

Consider Fig. 5 which shows the per-flow throughput and the fairness index for this experiment. In the "w/o separation" scenario (Fig. 5, top), as soon as the first BBR flow joins (at second 15), the throughput of the active CUBIC flow collapsed (from $\approx$ 2Gbps to $\approx$ 100Mbps), resulting in a decrease in the fairness index to $\approx 60\%$ (Fig. 5, bottom). Note how at second 30, the BBR flow collapsed, but later claimed most of the bandwidth (at second 40). The mean fairness index (μ) is 68% and the standard deviation (σ) is 24%. In the "w/ separation"



Propagation Delay [ms]	10	20	40	60	80	100	200
10	44	50	46	44	49	53	53
20	46	43	48	34	40	53	51
30	42	41	42	44	45	45	55
40	40	42	41	42	45	42	53
50	45	37	41	36	37	40	41
60	43	42	38	42	39	39	50
80	40	41	40	38	38	41	52

Propagation Delay [ms]	10	20	40	60	80	100	200
10	89	82	78	83	79	78	71
20	87	86	80	84	79	80	76
30	86	84	85	80	85	79	75
40	86	86	83	82	76	82	74
50	88	83	86	82	77	85	66
60	80	85	87	77	83	76	69
80	83	85	84	79	76	74	72

Fig. 6. Fairness index considering various propagation delays and various buffer sizes. (a) w/o separation; (b) w/ separation.

scenario (Fig. 5, middle), the throughput was fairly divided among the active flows, resulting in an average fairness index close 100%, with a small σ.

Experiment 2. In this experiment, 10 long flows started at the same time, with alternating CCAs (i.e., Flow1 uses CUBIC, Flow2 uses BBR, Flow3 uses CUBIC, etc.). Various propagation delays $[10, 20, 30, 40, 50, 60, 80]$ (ms) are introduced, and various buffer sizes $[10, 20, 40, 60, 80, 100, 200]$ (ms) are configured on the router R1. In the "w/ separation" scenario, the buffer size is configured on both queues (loss-based and model-based). The experiment lasted 60 seconds.

Consider Fig. 6 which shows the fairness index for each test in this experiment. In the "w/o separation" scenario (Fig. 6 (a)), the fairness index is low, regardless of the buffer size or the propagation delay. On the other hand, the fairness index is approximately two times higher in the "w/ separation" scenario (Fig. 6 (b)).

D. Flow Completion Time Evaluation

Flow Completion Time (FCT) is defined as the amount of time that elapses from when the first packet is sent until the last packet reaches its destination. This test evaluates the impact of separating flows by their CCAs on the FCT of short and long flows.

Short flows. This test measures the FCTs and the RTTs of short flows sharing the bottleneck link with long flows. In this experiment, 100 long flows are generated over a bottleneck link of 3Gbps (1Gbps for each queue for the "w/ separation scenario"). The queue size for the "w/o separation" scenario is 200ms, a size recommended by major vendors [20]. The long flows are 50% CUBIC, and 50% BBR. In addition, 10,000 short flows, whose inter-connection times are generated from an exponential distribution with a mean of one second, are initiated. Fig. 7 shows the Cumulative Distribution Functions (CDFs) of the FCT and the RTT of short flows. Without traffic separation, the FCTs significantly increase; it took more than 0.5s for the majority of the flows to finish. The FCTs increase due to the large RTTs resulting from sharing the oversubscribed queue (Fig. 7 (b)). With traffic separation, the RTTs of the short flows remain small, leading to small FCTs; all flows were completed in less than 0.2s, with a $\mu = 0.05$ and a $\sigma = 0.03$.

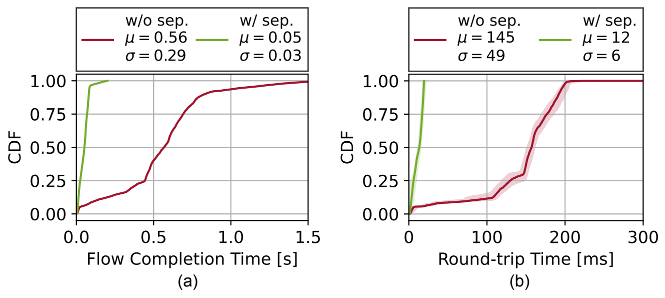


Fig. 7. Flow completion time (a) and round-trip time (b) for short flows. The shaded area demonstrates the CDF of the minimum RTT and the maximum RTT of each connection.

Long flows. This test evaluates the impact of traffic separation on the FCT of long flows. The experiment uses the same settings as those described in Experiment 2 (Section IV.C), but instead of running the test for 60 seconds, each flow transfers a 500MB file[1]. Fig. 8 shows the CDF of the FCTs (a) and the throughput samples (b) of all the tests. In the "w/ separation" scenario, the majority of the flows concluded within [15-25s]. On the other hand, while some flows terminated faster in the "w/o separation scenario", approximately 30% of the flows took more than 30s to complete. This unfairness is also reflected in the throughput samples observed throughout all tests (Fig. 8 (b)).

V. Conclusion

This paper presented a system that uses passive measurements to identify CCAs at line rate. After identifying the CCA, the flow is enqueued into a dedicated queue based on the CCA variant. The experiments were conducted on real hardware, and real datasets were used for testing. The results show that 1) P4CCI is able to identify the CCA with high accuracy; 2) the fairness of the long flows is significantly improved; 3) the FCT of the short flows is also improved when compared to the default enqueeing model used in today's networks. A limitation of P4CCI is that it assumes that the flows are uniformly distributed based on their CCA. The authors plan to solve this queue assignment imbalance problem for future work. The authors also plan to separate the flows based on their RTTs to mitigate the RTT unfairness problem of TCP.

References

[1] Sally Floyd, Tom Henderson, and Andrei Gurtov. The NewReno modification to TCP's fast recovery algorithm. Technical report, RFC, 2004.

[2] Sangtae Ha, Injong Rhee, and Lisong Xu. CUBIC: a new TCP-friendly high-speed TCP variant. *ACM SIGOPS Operating Systems Review*, 42(5):64–74, 2008.

[3] 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.

[4] 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.

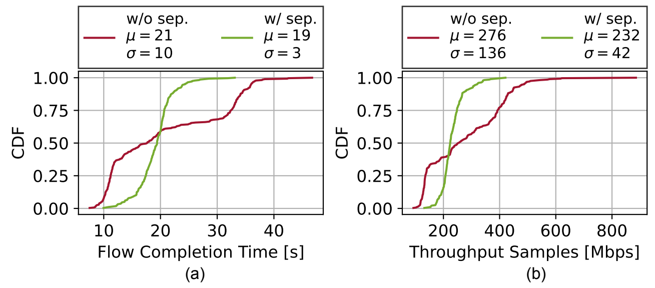


Fig. 8. FCTs (a) and throughput samples (b) of long flows. FCTs/throughput are close to 20s/200Mbps, the ideal values in a fair network.

[5] 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.

[6] Jose Gomez, Elie Kfoury, Jorge Crichigno, Elias Bou-Harb, and Gautam Srivastava. A performance evaluation of TCP BBRv2 alpha. In *2020 43rd International Conference on Telecommunications and Signal Processing (TSP)*, pages 309–312. IEEE, 2020.

[7] 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.

[8] Elie F Kfoury, Jorge Crichigno, Elias Bou-Harb, David Khoury, and Gautam Srivastava. Enabling TCP pacing using programmable data plane switches. In *2019 42nd International Conference on Telecommunications and Signal Processing (TSP)*, pages 273–277. IEEE, 2019.

[9] Jose Gomez, Elie F Kfoury, Jorge Crichigno, and Gautam Srivastava. A survey on TCP enhancements using P4-programmable devices. *Computer Networks*, 212:109030, 2022.

[10] Jitendra Pahdye and Sally Floyd. On inferring TCP behavior. *ACM SIGCOMM Computer Communication Review*, 31(4):287–298, 2001.

[11] Peng Yang, Juan Shao, Wen Luo, Lisong Xu, Jitender Deogun, and Ying Lu. TCP congestion avoidance algorithm identification. *IEEE/ACM Transactions On Networking*, 22(4):1311–1324, 2013.

[12] Belma Turkovic and Fernando Kuipers. P4air: Increasing fairness among competing congestion control algorithms. In *2020 IEEE 28th International Conference on Network Protocols (ICNP)*, pages 1–12. IEEE, 2020.

[13] Junpei Oshio, Shingo Ata, and Ikuo Oka. Identification of different TCP versions based on cluster analysis. In *2009 Proceedings of 18th International Conference on Computer Communications and Networks*, pages 1–6. IEEE, 2009.

[14] Toshihiko Kato, Xiaofan Yan, Ryo Yamamoto, and Satoshi Ohzahata. Identification of TCP congestion control algorithms from unidirectional packet traces. In *Proceedings of the 2nd International Conference on Telecommunications and Communication Engineering*, pages 22–27, 2018.

[15] Constantin Sander, Jan Rüth, Oliver Hohlfeld, and Klaus Wehrle. Deepcci: Deep learning-based passive congestion control identification. In *Proceedings of the 2019 Workshop on Network Meets AI & ML*, pages 37–43, 2019.

[16] Boris Iglewicz and David Hoaglin. How to detect and handle outliers. *The ASQC basic references in quality control: Statistical Techniques*, 16, 1993.

[17] Hassan Ismail Fawaz, Germain Forestier, Jonathan Weber, Lhassane Idoumghar, and Pierre-Alain Muller. Deep learning for time series classification: a review. *Data mining and knowledge discovery*, 33(4):917–963, 2019.

[18] Zhiguang Wang, Weizhong Yan, and Tim Oates. Time series classification from scratch with deep neural networks: A strong baseline. In *2017 International Joint Conference on Neural Networks (IJCNN)*, pages 1578–1585. IEEE, 2017.

[19] Anonymized Internet Traces 2019. https://catalog.caida.org/details/dataset/passive_2019_pcap. Accessed: 2022-5-13.

[20] Nick McKeown, Guido Appenzeller, and Isaac Keslassy. Sizing router buffers (redux). *ACM SIGCOMM Computer Communication Review*, 49(5):69–74, 2019.

[1]In a fair network with a bottleneck of 2Gbps and 10 active flows, each flow is transferring at 200Mbps, resulting in an FCT = $\frac{500MB}{200Mbps} = 20s$.