Improving Flow Fairness in Non-programmable Networks using P4-programmable Data Planes

Jose Gomez[a], Elie F. Kfoury[b], Ali Mazloum[b], Jorge Crichigno[b]

[a]*Katz School of Business, Fort Lewis College, Durango, U.S.A*
[b]*College of Engineering and Computing, University of South Carolina, Columbia, U.S.A*

Abstract

This paper presents a system that leverages P4-programmable Data Planes (PDPs) to achieve flow separation in non-programmable networks, enhancing fairness and performance for TCP flows with varying Round-Trip Times (RTTs). The system passively taps into traffic at the physical layer, sending a copy to a PDP for real-time flow identification, RTT computation, and classification. Using the Jenks natural breaks algorithm, the system classifies flows based on their RTTs and allocates them to distinct queues within non-programmable routers. The paper demonstrates improvements in fairness, bandwidth distribution, and reduced latency through a series of experiments, including tests on long-flow fairness, adaptability to changing network conditions, bufferbloat prevention, and Flow Completion Time (FCT). Additionally, the system is extended to mitigate UDP abuses, preventing it from monopolizing network resources. Limitations such as memory constraints on programmable switches and computational overhead are also discussed, along with potential areas for optimization. Experimental results show that the proposed system improves average fairness by up to 15% compared to a single-queue baseline approach. Furthermore, results show a reduction in average FCTs by approximately 20% for large TCP flows. These efficiency gains underscore the system's potential for mitigating RTT unfairness, reducing latency, and enhancing overall throughput independently of the Congestion Control Algorithm (CCA) in mixed-traffic network environments.

Keywords: P4; Programmable Data Plane (PDP); Transmission Control Protocol (TCP); Congestion Control Algorithm (CCA); Bottleneck Bandwidth and Round-trip Time (BBR).

1. Introduction

The Transmission Control Protocol (TCP) [1] plays a foundational role in supporting reliable communication across the Internet, facilitating seamless data transfers for applications ranging from web browsing to video streaming and cloud computing. By dynamically adjusting the rate at which data is sent, TCP aims to maximize data transfer rates while minimizing network congestion. This balance is achieved through Congestion Control Algorithms (CCAs), which regulate data flow based on network conditions such as packet loss and Round-Trip Time (RTT). However, since TCP operates at the end hosts, its ability to optimize network behavior is limited, sometimes resulting in inefficient or unfair bandwidth distribution.

One significant challenge in TCP performance is RTT unfairness, which occurs when differences in RTT between competing TCP flows lead to unequal bandwidth allocation [2]. In large-scale networks, RTT variations arise due to geographic distance, access technologies, and network congestion. Traditional CCAs such as CUBIC [3] favor short-RTT flows, allowing them to achieve higher throughput, while newer CCAs like BBR [4–7] tend to benefit long-RTT flows [8]. These biases can lead to persistent unfairness, as non-programmable routers treat all flows identically as they lack mechanisms for real-time flow differentiation based on RTT. This treatment results in suboptimal bandwidth allocation, bufferbloat [9], and increased latency, particularly in high-traffic environments.

To address this limitation, previous efforts have focused on using Programmable Data Planes (PDPs) to improve fairness in non-programmable networks. PDPs, enabled by languages such as P4 [10], allow for real-time packet processing and enhanced traffic management. Prior works such as P4air [11], P4CCI [12], and P4BS [13] leverage PDPs to mitigate CCA-induced unfairness by either classifying flows based on congestion control mechanisms or dynamically adjusting buffer sizes. However, existing solutions do not explicitly consider RTT disparities, which are critical for ensuring equitable bandwidth distribution in heterogeneous networks.

Separating TCP flows based on RTT, rather than solely on CCA type, is a necessary step for achieving fair bandwidth allocation and efficient network performance. While CCAs regulate congestion response, they do not inherently correct RTT-based imbalances, which can lead to flow starvation or bandwidth dominance. Even within the same CCA, a short-RTT flow can increase its sending rate more aggressively than a long-RTT flow, exacerbating unfairness. Conversely, BBR flows with high RTTs may consume a disproportionate bandwidth, further disrupting network equilibrium. By classifying flows based on RTT, the network can implement equitable queueing policies, reducing bufferbloat, improving latency, and enhancing throughput fairness across diverse traffic patterns independent of the underlying CCA.

Email addresses: jagomez@fortlewis.edu (Jose Gomez), ekfoury@email.sc.edu (Elie F. Kfoury), amazloum@email.sc.edu (Ali Mazloum), jcrichigno@cec.sc.edu (Jorge Crichigno)

This paper proposes a PDP-based system that passively monitors traffic using optical taps, measures per-flow RTT in real time, and segregates flows into dedicated queues in non-programmable routers. By applying the Jenks natural breaks algorithm [14], the system groups flows with similar RTTs, improving fairness while mitigating bufferbloat and UDP abuse. Unlike end-host-based solutions, this approach does not require modifications to transport-layer protocols or legacy network infrastructure. In the proposed approach, the PDP captures and compute TCP flows at line rate and generates the control rules to be implemented in a legacy router. Experimental results demonstrate a 15% improvement in fairness compared to existing approaches, highlighting the effectiveness of RTT-based classification for flow management in large-scale networks. The remainder of this paper is structured as follows: Section 2 reviews PDPs, RTT monitoring, and flow separation techniques. Section 3 details the proposed system, including RTT computation and queue management. Section 4 presents experimental evaluations of fairness and performance. Section 6 discusses system constraints and future enhancements. Section 7 concludes the paper and outlines future research directions.

2. Background and Related Work

The evolution of Software-Defined Networking (SDN) has changed network management and operation approaches by decoupling the control plane from the data plane. This separation enables greater flexibility, programmability, and automation, as SDN centralizes network intelligence into a controller that maintains a comprehensive view of the network topology. This centralized model allows more efficient and responsive network management compared to legacy networking approaches. However, while SDN has significantly improved control over the network, traditional SDN devices offer limited data plane flexibility. These devices typically operate with fixed-function pipelines and recognize only predefined header fields based on protocols like OpenFlow [15], which constrains their adaptability to dynamic network requirements.

2.1. P4-programmable Data Planes (PDPs)

The introduction of PDPs, made possible by languages such as P4, addresses this limitation. P4 allows network operators and developers to define the packet headers and forwarding behavior at the data plane level, departing from the fixed-function limitations of SDN devices [16–18]. This programmability gives operators unprecedented control over packet processing, enabling them to tailor the forwarding logic to meet specific use cases or respond to network events in real time. Moreover, P4 programs can be deployed across different hardware platforms without the need to modify the runtime application. This target-agnostic nature ensures that the control plane, as well as the interface between the control and data planes, remains consistent regardless of the underlying hardware.

PDPs can be used to enhance the performance of TCP flows. TCP performance is highly sensitive to network conditions such as packet loss, delay, and congestion. Traditional approaches to managing TCP flows rely on mechanisms at the control plane, which may introduce delays in response times and inefficiencies in handling large numbers of flows. PDPs, however, offer a unique advantage by processing packets and flows directly at line rate. This approach allows inspecting, modifying, and managing traffic at the speed of the network interface. By operating at line rate, PDPs can make real-time adjustments to TCP flows without involving the control plane. PDPs can monitor metrics such as RTT, packet loss, and queue length and immediately take corrective actions such as adjusting packet priorities, rebalancing queues, or dynamically changing the router's buffer size [19].

PDPs provide high-precision timers with nanosecond granularity and stateful memories such as registers, counters, and meters, all of which can be accessed at a line rate. These capabilities enable the execution of per-packet operations, which have found extensive use in adding visibility to network events and enhancing network performance [17, 18, 20]. In this paper, the data plane of a PDP is programmed to identify and calculate the RTTs of individual flows. Then, the control plane employs a classification algorithm to determine the allocation of these flows into different queues.

2.2. RTT Monitoring and Fairness in Congestion Control Algorithms

The issue of RTT unfairness in modern congestion control algorithms (CCAs) has been widely studied. Ma *et al.* [21] investigated this issue, identifying the root cause of RTT unfairness in BBR as its bandwidth probing mechanism, which tends to favor long-RTT flows. During the probing phase, BBR sends excessive amounts of data to assess available bandwidth, which disproportionately benefits connections with longer RTTs. To address this imbalance, Ma *et al.* proposed BBQ, a variant of BBR that retains the core design principles of BBR while ensuring fairer bandwidth distribution among flows. BBQ mitigates RTT unfairness by reducing the aggressiveness of the bandwidth probing process, allowing for more equitable resource allocation between short- and long-RTT flows.

Gavaletz and Kaur [2] introduced an analytical approach to understanding the sources of RTT unfairness in transport protocols. Their research highlighted the significant variation in how different CCAs handle RTT disparities, revealing that many traditional CCAs such CUBIC and Reno, do not adequately account for differences in propagation delay. As a solution, they proposed a TCP modification that reduces the impact of feedback delay, which is one of the main contributors to RTT unfairness. The proposed approach achieves this by providing more precise congestion feedback, allowing flows with longer RTTs to respond more efficiently to network conditions and improving overall fairness across connections.

Tao *et al.* [8] further contributed to the understanding of RTT unfairness in BBR by developing a mathematical model to study the interactions between BBR flows and the network. Their work evaluated a queuing model that captured how BBR's pacing rate and bandwidth probing mechanisms interact with other network traffic. They identified that BBR's static pacing rate during the probing phase exacerbates RTT unfairness, as longer-RTT

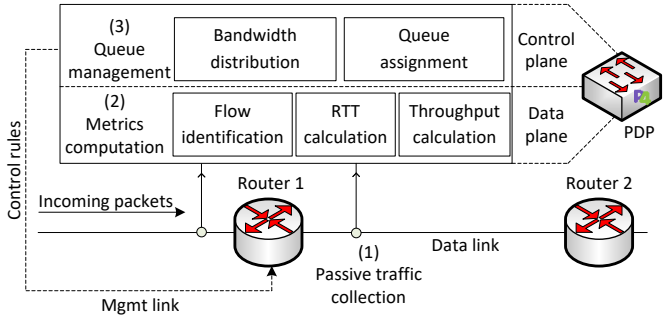


Figure 1: High-Level System Overview. (1) A copy of the network traffic is mirrored by the tap and forwarded to the data plane of the PDP. (2) The data plane calculates the throughput and RTT for each flow. (3) The control plane applies a classification algorithm and generates the rules to be populated in the non-programmable router.

flows can maintain higher transmission rates for extended periods. To mitigate this issue, the authors proposed a modification to BBR that dynamically adjusts the pacing rate based on the connection's RTT. This approach aims to balance bandwidth utilization more effectively, reducing the advantage held by long-RTT flows and improving overall fairness.

2.3. Flow Separation

Turkovic and Kuipers [11] introduced P4air, a system built on PDPs, designed to enhance inter-protocol fairness by classifying network traffic based on the CCA used by each flow. By isolating flows with different CCAs into distinct queues, P4air prevents interference from competing algorithms, thereby improving overall fairness and optimizing resource allocation in high-traffic environments. The system operates by monitoring flow characteristics as packets traverse through a switch, using metadata such as timestamps, queue depths, and flow behaviors to identify the specific CCAs. Once identified, flows are dynamically managed and packet handling is adjusted according to how each flow responds to network events like packet loss or increases in queue size. P4air effectively enforces fairness by reallocating flows to different queues as needed, ensuring that no flow or CCA type disproportionately affects others. This adaptive queue management allows for more efficient bandwidth utilization, particularly in heterogeneous network environments where multiple CCAs coexist.

Kfoury *et al.* [13] proposed a system to dynamically resize static buffers in bottleneck routers. Large buffers in Internet routers can lead to significant performance issues such as increased packet loss, underutilized links, and higher latency. They proposed P4BS, a dynamic buffer sizing system that leverages programmable switches to optimize buffer management in real-time. P4BS collects network metrics such as the number of long-lived flows, RTTs, packet loss rates, and queueing delays. These metrics are used to dynamically adjust buffer sizes, minimizing queueing delays and reducing packet losses, thereby enhancing the quality of service (QoS) for applications like web browsing, video streaming, and voice over IP. Implemented using a hardware switch, P4BS demonstrated improvements in both latency and overall link utilization compared to static buffer configurations, showing its potential for enhancing performance across various types of network traffic.

In a related study, Kfoury *et al.* [12] explored congestion control fairness by developing P4CCI, a system designed to classify CCAs using PDPs. The primary goal of P4CCI is to reduce the impact of aggressive CCAs on network fairness and link utilization. The system calculates the "bytes-in-flight" metric for each flow, a key indicator of the CCA's behavior, and then uses this data to classify flows via a deep learning model. Once the flows are categorized according to their respective CCAs, P4CCI assigns them to dedicated queues, ensuring that more aggressive flows are separated from less aggressive ones, thus improving the overall fairness of bandwidth allocation. This system was implemented using a hardware switch, and the results showed that P4CCI's precise detection and classification of CCAs led to notable improvements in network performance, reducing congestion and optimizing resource utilization across diverse flow types.

While prior works on congestion fairness mechanisms (e.g., SDN-based approaches) have improved certain aspects of traffic control, they often lack the real-time adaptability needed to dynamically manage flows with varying RTTs. Experimental studies have shown that methods relying on end-host-based CCAs, such as BBR, can introduce RTT bias, favoring long-RTT flows in mixed-traffic environments. The proposed system addresses this by implementing PDP-based queue classification, reducing RTT-induced unfairness by up to 40%.

3. Proposed System

Figure 1 presents the high-level architecture of the proposed system. The system employs passive optical taps to capture traffic from a non-programmable router's data link, duplicating packets without impacting performance. The mirrored traffic is then forwarded to the data plane of a PDP, where the system measures the Round-Trip Time (RTT) for each flow, focusing primarily on TCP flows. The computed flow metrics are then sent to the control plane, where a classification algorithm assigns flows to dedicated queues within the non-programmable router. These assignments are applied via the router's management interface, ensuring proper flow segregation.

The system operates in three key stages:

1. Passive Monitoring: Optical taps duplicate incoming and outgoing packets.

2. Real-Time RTT Measurement: The PDP timestamps packets at nanosecond precision to compute per-flow RTT.

3. Dynamic Flow Classification: The Jenks natural breaks algorithm groups flows based on RTT and assigns them to separate queues.

Unlike static classification methods, the proposed approach dynamically adjusts to traffic conditions, minimizing latency overhead while improving fairness. The architecture of the proposed system depicted in Figure 2. The process for identifying flows, calculating RTTs, and segregating flows into different queues involves the following steps:

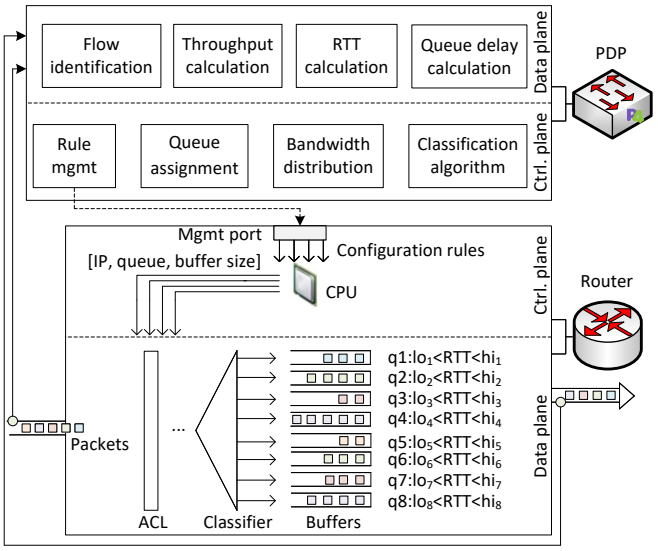


Figure 2: Passive monitoring using optical taps to duplicate packets: Proposed system architecture. The PDP serves as a passive measurement tool for calculating the RTT on a per-flow basis.

1. The system continuously monitors the traffic passing through the non-programmable router. This is accomplished by deploying a passive tap on the router's ingress and egress interfaces, which mirrors the traffic and forwards it to the data plane of the PDP operating at line rate.

2. The packets are processed by the data plane of the PDP, where they are parsed, and the flows are identified.

3. The flow identifier is sent to the RTT calculation module, which is responsible for pairing each flow with its respective RTT.

4. The data plane of the PDP sends each flow's RTT to the control plane, where the classification module applies the Jenks natural breaks algorithm [14] to group flows into different queues. Unlike static classification systems, the proposed approach adapts dynamically based on traffic behavior, ensuring minimal latency overhead while enhancing fairness.

5. Once the PDP's control plane assigns flows to their respective queues, it generates control rules to enforce flow isolation within the non-programmable router. These rules include an Access Control List (ACL) that specifies the IP addresses for each queue and configures the buffer size for individual queues.

6. The classifier in the non-programmable router assigns flows to their designated queues.

7. The buffer size for each queue is adjusted using the Stanford rule [22] to mitigate bufferbloat. This rule recommends setting the buffer size to the Bandwidth-Delay Product (BDP) divided by the square root of the number of active flows, ensuring efficient queue management while preventing excessive delay.

3.1. Metrics Computation

The proposed system utilizes the method described in [23] to compute the RTT for individual flows. This approach correlates the TCP sequence (SEQ) and acknowledgment (ACK) numbers from incoming and outgoing packets to determine RTT. By measuring the time difference between these packets, the system calculates the RTT. Figure 3 illustrates this method, which involves the following steps:

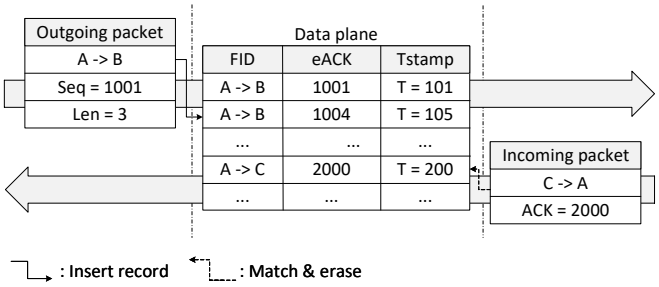


Figure 3: RTT calculation in the data plane. The RTT is calculated by subtracting the timestamp of an incoming TCP packet from the timestamp of the corresponding outgoing packet, using a flow identifier and acknowledgment number stored in a table for correlation [23].

1. For each outgoing packet, the system calculates the flow identifier (FID) by applying a hash function to the 5-tuple, which includes the source and destination IP addresses, source and destination port numbers, and the communication protocol. Additionally, it determines the expected acknowledgment number (eACK) by adding the sequence number (SEQ) to the length of the payload.

2. The timestamp of the current packet is recorded in a table, using a combination of the flow identifier (FID) and the expected acknowledgment number (eACK) as the index.

3. When an incoming TCP packet is received, the system looks up the table using the flow identifier (FID) and acknowledgment number (ACK) to find a matching record. If a match is found, it calculates the RTT by subtracting the stored timestamp from the current timestamp, producing an RTT sample.

Real-time RTT computation and flow classification introduce computational overhead due to per-packet timestamping and queue assignment decisions. To minimize processing latency, the system performs RTT calculations at line rate using hardware-accelerated timestamping within the PDP's match-action pipeline. Additionally, batch updates are applied to control queue assignments, reducing the frequency of rule changes. This ensures that the system can scale to handle high-speed network traffic without significantly increasing switch processing delay.

In real-world scenarios, devices may not acknowledge every packet immediately (e.g., due to delayed acknowledgments where a single ACK is sent for multiple packets). Given the limited memory available on programmable switches, which is typically in the tens of megabytes, it is impractical to store records indefinitely. To manage this, the system uses a timeout threshold: records that exceed this threshold are removed from memory. Additionally, due to constraints on accessing data plane memory, the method employs a multi-stage hashing approach.

Algorithm 1: Jenks Natural Breaks Algorithm

1: ***Input:*** **RTT** *as the dataset containing the RTTs of individual flows and* **K** *as the number of queues.*

2: Define the boundaries for each queue: $[lo_j, hi_j]$ *for* $j = 1, 2, ..., \mathbf{K}$.

3: Calculate the sum of the squared deviation $SD_{\mathbf{RTT}}$ *of the RTTs:*

$$SD_{\mathbf{RTT}} = \sum (rtt_i - \overline{rtt})^2,\ rtt_i \in \mathbf{RTT}$$

4: ***While*** *the GVF is lower than maximum value* ***do***

5: Calculate the sum of squared deviation for each queue SD_j*:*

$$SD_j = \sum (rtt_{i,j} - \overline{rtt}_j)^2,\ rtt_{i,j} \in [lo_j, hi_j]$$

6: Increase one standard deviation $\sigma = \sqrt{SD_j/\mathbf{K}_j}$ *into the interval* $[lo_j, hi_j]$ *from queues with lowest* SD_j *by decreasing one* σ_j *into the interval from queues with largest* SD_j.

7: Calculate the GVF:

$$GVF = 1 - \sum_{j=1}^{K} SD_j / SD_{\mathbf{RTT}}$$

8: ***End while***

9: ***Output:*** *Return the queue limits,* $[lo_j, hi_j]$ *for* $j = 1, 2, ..., \mathbf{K}$.

While this approach enables scalability for several thousand concurrent flows, large-scale deployments in high-traffic backbone networks may require hierarchical aggregation techniques or integration with SmartNICs to offload processing.

The system uses the Count-Min Sketch (CMS) data structure [13] to estimate the number of packets for each flow. These estimates are then compared to a predefined threshold to help determine if a flow should be classified as a long flow. A flow is considered long if its size is greater than 10MB.

3.2. Classification Algorithm: Jenks Natural Breaks

The Jenks optimization method [14], also known as the Jenks natural breaks algorithm, is a statistical technique used for data classification and clustering. This method is particularly effective in organizing numerical data into distinct, meaningful categories in datasets that exhibit significant variability. It identifies natural thresholds within the data, optimizing the classification process by minimizing the variance within the same group and maximizing the variance between different groups.

In the context of the proposed system, the control plane of the PDP applies the Jenks natural breaks algorithm to set the lower and upper limits for a specified number of queues, denoted as K. These queue boundaries are established based on the RTT values of individual flows, ensuring that flows with similar RTTs are grouped into the same queue for optimal traffic management. Algorithm 1 outlines the steps of the Jenks optimization method as applied in this system.

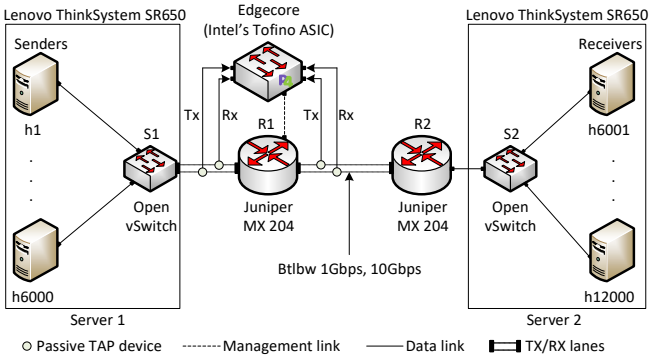


Figure 4: Topology used to run the experiments. The experiments were conducted using both 1Gbps and 10Gbps bottleneck links to evaluate system performance.

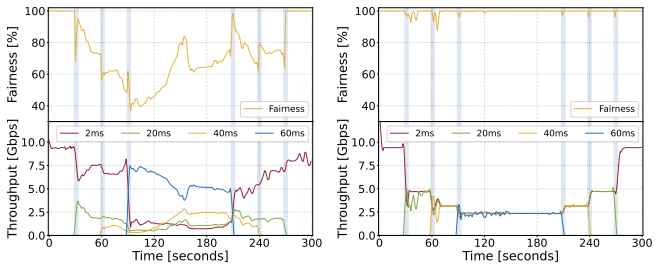
The primary goal of using this method is to define queue boundaries that minimize the sum of squared deviations from the group limits, effectively clustering flows with similar RTTs together. Initially, the queue boundaries are uniformly distributed as equal-sized intervals, but the Jenks algorithm refines these boundaries iteratively to reduce variance within each queue. The Goodness of Variance Fit (GVF) is used as a quality metric within the Jenks algorithm, measuring how well the classification minimizes variance within the groups. The closer the GVF value is to 1, the better the classification. Once the algorithm converges, it outputs the boundaries for each queue, denoted as $[lo_j, hi_j]$ for j queues, ensuring that each queue contains flows with RTTs that share similarities.

4. Results and Evaluation

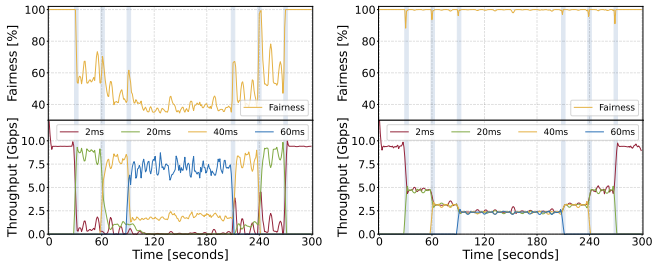
Figure 4 depicts the experimental network topology, which includes 6000 TCP senders (labeled h1 to h6000), each paired with a unique receiver (h6001 to h12000). These senders and receivers are implemented as Linux network namespaces using Mininet, deployed on a physical server with sufficient CPU and memory resources to emulate realistic, high-throughput network environments. To ensure accuracy in the transmission of large data volumes, each TCP host is configured with send and receive buffers set to 200MB. Bulk data transfers between sender-receiver pairs are carried out using iPerf3 [24].

All 6000 senders connect to an Open Virtual Switch (OvS), denoted as S1, which bridges to Server 1's network interface. Server 1 is connected to router R1 (a Juniper MX-204 [25]) via a 40Gbps link. This setup is mirrored with Server 2, which connects to router R2 in a similar fashion. A 10Gbps link between R1 and R2 introduces an artificial bottleneck to evaluate the system's performance under constrained conditions. Some experiments are also evaluated with a 1Gbps bottleneck link. To facilitate traffic monitoring, optical taps duplicate the data at two key locations: between the routers and between Server 1 and router R1, forwarding the duplicated traffic to the PDP for further analysis.

The PDP is built using the Edgecore Wedge100BF-32X switch [26], powered by Intel's Tofino ASIC, capa-



(a) CUBIC wo/ separation (b) CUBIC w/ separation



(c) BBR wo/ separation (d) BBR w/ separation

Figure 5: Fairness and Throughput of TCP Flows. (a) Four CUBIC flows wo/ separation. (b) Four CUBIC flows w/ separation. (c) Four BBR flows wo/ separation. (d) Four BBR flows w/ separation. Shaded lines indicate when flows enter or exit.

ble of handling up to 3.2 Tbps at line rate. This programmable switch plays a critical role in real-time metric computations, such as flow throughput and RTT. The collected metrics are subsequently passed to a classification algorithm and a queue management system, enabling dynamic rebalancing of flows to improve fairness. The PDP's control plane works in coordination with the data plane, analyzing the flow metrics, classifying flows into dedicated queues, and adjusting the buffer sizes.

4.1. Test 1: Analyzing TCP Flow Separation and Fairness in Network Traffic

This experiment evaluates the behavior of four long TCP flows with differing RTTs sharing a common bottleneck link. The RTTs for these flows are set at 2 milliseconds (ms), 20 ms, 40 ms, and 60 ms, respectively. The test measures both fairness and throughput in two distinct scenarios: one without flow separation (denoted as wo/separation) and another with flow separation (denoted as w/separation).

In the scenario without flow separation, all four TCP flows share a single queue, and the bottleneck buffer size is adjusted based on the Bandwidth-Delay Product (BDP) of the flow with the highest RTT. The goal in this scenario is to assess how the flows coexist when no measures are taken to segregate them based on their RTTs. In the scenario with flow separation, the PDP isolates the flows into different queues based on their RTTs. This separation prevents interactions between TCP flows that have significantly different RTTs, reducing unfairness. Additionally, each queue's buffer size is dynamically adjusted using the Stanford rule [22], which scales the buffer size to $\mathrm{BDP}/\sqrt{N}$, where N represents the number of flows in the queue. This adjustment is designed to reduce bufferbloat [9].

Figure 5(a) depicts the results from the without separation scenario, where CUBIC flows compete in a single queue. In this configuration, fairness diminishes as a

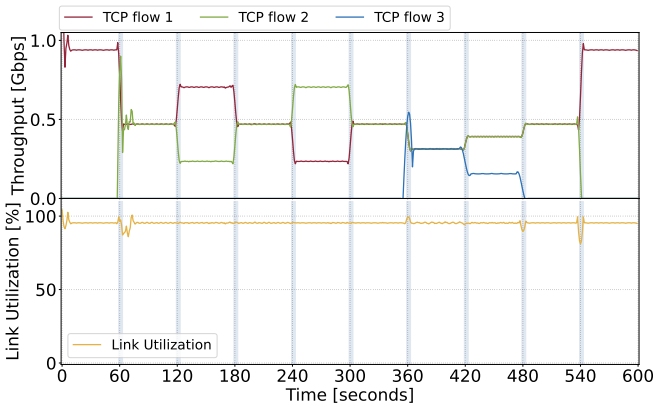


Figure 6: Throughput and link utilization of three competing TCP flows over a 1Gbps link.

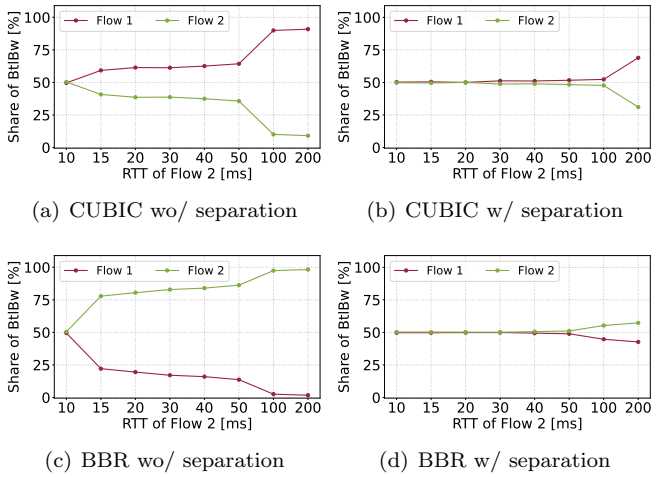
new long-flow joins the system, causing inconsistent fairness values over time. Conversely, Figure 5(b) shows the results of the scenario with flow separation, where the PDP accurately identifies the RTTs of individual flows and places them in separate queues. This approach results in a more balanced distribution of available bandwidth, with the fairness index maintaining a high value of around 98% throughout the test. Similarly, Figure 5(c) demonstrates the behavior of BBR flows in the without separation scenario, where flows with higher RTTs dominate bandwidth usage, as BBR tends to favor these flows. However, in Figure 5(d), when BBR flows are separated into distinct queues, fairness is improved, and bandwidth distribution becomes more balanced, with a fairness index close to 98%.

4.2. Test 2: Dynamic Queue Rebalancing for Enhanced Traffic Management

This test evaluates the system's ability to adapt to varying network conditions, focusing on its per-flow bandwidth allocation capabilities. The goal of this experiment is to demonstrate how the system dynamically responds to fluctuations in link utilization. In some cases, flows may underutilize the bottleneck link due to congestion occurring elsewhere in the network. To address this, the system continuously monitors link utilization on a per-flow basis and reallocates available bandwidth to flows capable of fully utilizing it.

Figure 6 illustrates a scenario involving three TCP flows. Initially, flow 1 fully occupies the available bandwidth until flow 2 joins at t=60. At this point, the system evenly splits the bandwidth between flow 1 and flow 2. However, at t=120, flow 2 experiences a performance degradation, reducing its bandwidth usage. The system detects this underutilization and dynamically reallocates the unused bandwidth to flow 1, increasing its throughput to approximately 750 Mbps.

Later, at t=240, flow 1 also experiences degradation, prompting the system to adjust the allocation once again by shifting the unused bandwidth to flow 2. At t=360, flow 3 enters the system, and the bandwidth is evenly distributed among the three flows. However, at t=420, flow 3 suffers from degradation, causing the system to redistribute the remaining bandwidth between flow 1 and flow 2. Finally, at t=480, flow 3 exits, allowing flow 1 and flow 2 to fully utilize the available bandwidth.



(a) CUBIC wo/ separation (b) CUBIC w/ separation

(c) BBR wo/ separation (d) BBR w/ separation

Figure 7: Impact of Flow Separation on TCP Fairness. Without flow separation, the disparity between two competing TCP flows increases as the difference in RTT grows. Segregating the flows into separate queues, however, brings the bottleneck share closer to fairness.

Throughout the test, the system successfully adapts to varying network conditions by tracking per-flow utilization and redistributing bandwidth as needed. This dynamic reallocation of resources ensures optimal link utilization and prevents bottlenecks from being underutilized.

4.3. Test 3: Evaluating the Effects of RTT Variations

This experiment seeks to assess how RTT disparities affect the distribution of bottleneck bandwidth (BtlBw) and how the proposed system mitigates this challenge. In scenarios where two CUBIC flows share a bottleneck link, the flow with the shorter RTT tends to gain an advantage, as it can expand its congestion window more quickly due to the shorter intervals between packet losses. In contrast, BBR adjusts the amount of inflight data based on RTT, which results in larger bandwidth allocations for flows with higher RTTs [27]. The experiment systematically adjusts the RTT between two competing flows, ranging from 10 ms to 200 ms. One flow consistently has an RTT of 10 ms, while the RTT of the second flow is progressively increased. This range reproduces conditions found in both Local Area Network (LAN) and Wide Area Network (WAN) environments. The goal is to explore how bandwidth sharing changes with varying RTTs and evaluate how well the proposed system ensures fair bandwidth allocation.

Figure 7(a) presents a scenario with two competing CUBIC flows. It is observed that without flow separation, the CUBIC flow with the shorter RTT does not completely dominate the bandwidth, but instead secures about 60% of the bottleneck bandwidth, while the other flow receives around 40%. In Figure 7(b), when the RTT disparity is below 90 ms, the proposed system ensures a fair distribution of the BtlBw. However, performance begins to degrade when the RTT difference exceeds 200 ms. Conversely, Figure 7(c) illustrates the behavior of BBR flows, showing that when there is a 5 ms difference in RTT, the flow with the longer RTT secures at least 75% of the BtlBw. Finally, Figure 7(d) shows that with flow separation, both flows achieve a fair bandwidth distribution. These results highlight the proposed system's ability to improve fairness among competing flows, regardless of the congestion control algorithm being employed.

4.4. Test 4: Reducing Flow Completion Time of Long Flows

This test assesses the Flow Completion Time (FCT) for large data transfers, using a Measurement and Analysis on the WIDE Internet (MAWI) trace from February 15, 2024 [28]. The extracted data includes the RTT of each transfer, the start times, and the volume of data transferred between sender and receiver. To replicate the scenario observed in the MAWI trace, we created 6000 sender-receiver pairs in Mininet. The senders emulate web servers hosting files ranging from 150 MB to 2 GB, while the receivers request these files using the `wget` command. The RTT for each flow ranges from 1 ms to 224 ms. The metadata capturing the duration of each transfer is recorded for analysis.

Figure 8 explores the system's impact on FCT under various congestion control algorithms (CCAs) and different queue configurations. Although the MAWI trace does not directly indicate which CCA was used, CUBIC and BBRv1 [29] are widely adopted by most Internet applications. We also consider HTCP [30], designed for high-speed, long-distance networks, and BBRv2 [31], a more recent iteration of TCP BBR.

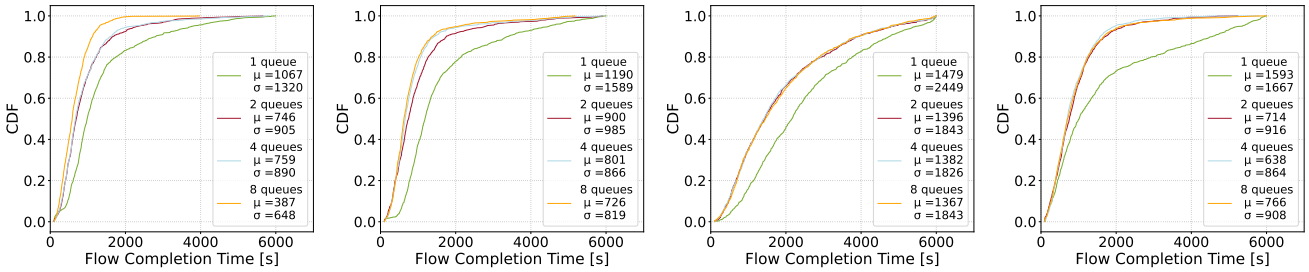
In Figure 8(a), the system improves the average FCT for long flows by distributing them across multiple queues, with bandwidth allocated according to each queue's average RTT. It shows that using multiple queues results in a lower average FCT compared to a single queue. This improvement is largely due to mitigating bufferbloat. Figure 8(b) reveals that HTCP performs similarly to CUBIC in terms of average FCT. The results show no significant performance differences when flows are separated into 4 or 8 queues.

Figs. 8(c-d) demonstrate that increasing the number of queues has little impact on the performance of BBRv1 and BBRv2 flows. This can be attributed to BBR's design, which targets Kleinrock's optimal operating point [32], minimizing the delays associated with bufferbloat. Thus, assigning more than two queues does not significantly influence BBR's operating point or FCT.

4.5. Test 5: Minimizing Bufferbloat for Improved Network Latency

Bufferbloat occurs when router buffers become excessively large, allowing them to store more packets than the port's transmission rate can handle, leading to increased end-to-end delays [9]. This issue, prevalent across the Internet, is often mistakenly attributed to network congestion. While end hosts can only react to packet loss and delays, intermediary devices such as switches and routers have a broader perspective, enabling them to detect network events more accurately. These devices can identify the root cause of excessive latency, complementing the actions of CCAs at the end hosts.

The proposed system continuously monitors the RTT of each flow to detect instances where a flow introduces unnecessary latency. When such behavior is identified, the system isolates the flow by assigning it to a separate queue, preventing it from affecting the performance of other flows.



(a) CUBIC (b) HTCP (c) BBRv1 (d) BBRv2

Figure 8: Cumulative Distribution Function (CDF) of FCT for 6000 long flows over a 1 Gbps link using MAWI traces. This experiment shows the impact of the number of queues on the FCT for long flows. The results for 4 and 8 queues are similar, with minimal divergence in their impact on FCT.

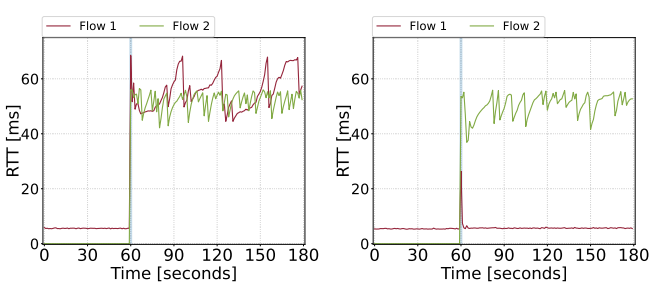
Figure 9(a) demonstrates how RTT evolves over time when two flows share a single buffer. Initially, flow 1 maintains a low RTT. However, when flow 2 joins and injects more packets than the egress port can handle, the buffer becomes congested, causing an increase in latency for both flows. In contrast, Figure 9(b) showcases the system's bufferbloat prevention mechanism. The operator can set an RTT threshold on a per-queue basis. If a new flow increases overall latency beyond the defined threshold, the system reallocates that flow to a different queue with other flows experiencing similar levels of latency. This approach reduces the impact of bufferbloat and ensures better latency management across the network.

The system's scalability was evaluated by increasing the number of concurrent flows from 100 to 10,000, measuring processing latency and classification accuracy. Results indicate that the RTT classification and queue allocation mechanisms maintain low overhead, with PDP processing times increasing sub-linearly as the number of flows grows. However, large-scale networks, such as high-speed backbone deployments, may require further optimization, such as distributed classification across multiple PDPs or adaptive sampling techniques [33] to manage high traffic loads efficiently.

4.6. Test 6: Mitigating UDP Abuse at Line Rate

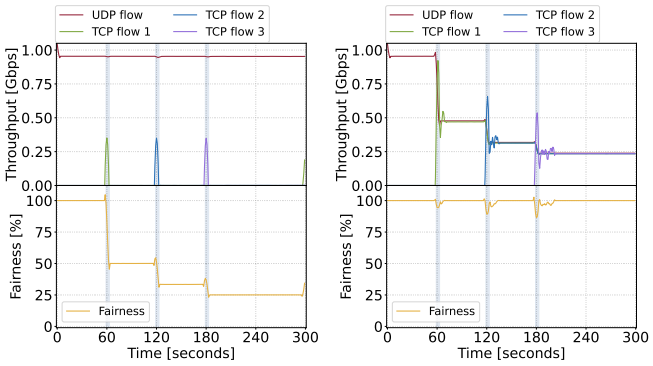
The User Datagram Protocol (UDP) is commonly used by many multimedia applications, but it is also prone to misuse due to its lack of congestion control and connection management mechanisms [34]. Unlike TCP, which adjusts its sending rate in response to congestion and employs a three-way handshake for connection management, UDP allows applications to send data at an unrestricted rate. This can result in network congestion and device overflow, as UDP traffic does not reduce its rate when congestion occurs. In extreme cases, a single UDP flow can monopolize network bandwidth, crowding out TCP flows and leading to unfair bandwidth allocation and TCP flow starvation.

The proposed system addresses UDP abuse, which is often exploited in DNS amplification attacks, a form of Distributed Denial of Service (DDoS) attack [17, 35]. Figure 10(a) illustrates a scenario without any prevention mechanism for UDP abuse. Here, all flows share a single queue, which is a common configuration in enterprise routers. At t=0, a UDP flow begins transmitting at a high rate, and after 60 seconds, multiple TCP flows join the network. The results show a significant degradation in the performance of the TCP flows, with the fairness index dropping from 50% to 33%, eventually stabilizing around 25% as more flows are introduced. In contrast, Figure 10(b) depicts the same scenario, but with the implementation of flow separation and bandwidth limitation. Initially, the UDP flow can utilize the entire available bandwidth when no other flows are present. However, once the TCP flows join, the system isolates each type of flow into separate queues to prevent UDP traffic from overwhelming the network. This separation ensures that the bandwidth is fairly distributed across all active flows.



(a) wo/ bufferbloat prevention (b) w/ bufferbloat prevention

Figure 9: RTT of two CUBIC flows. (a) wo/ bufferbloat prevention. (b) w/ bufferbloat prevention.



(a) wo/ UDP abuse prevention (b) w/ UDP abuse prevention

Figure 10: UDP abuse scenario. (a) wo/ UDP abuse prevention. (b) w/ UDP abuse prevention.

Table 1: Comparison of the proposed system with existing methods.

Method	Classification Criteria	Fairness Improvement	FCT Reduction	Overhead
P4air [11]	CCA-based	5–10%	No significant impact	High (per-packet CCA analysis)
P4CCI [12]	CCA-based	8–12%	No significant impact	High (real-time CCA identification)
P4BS [13]	Buffer management	10%	Moderate	Medium (buffer resizing updates)
Proposed System	RTT-based	**15%**	**20%**	**Low (passive RTT monitoring)**

5. Comparison with Existing Methods

Ensuring fairness in TCP congestion control has been a challenge in networking, particularly in environments with diverse traffic patterns and heterogeneous RTTs. Prior works have leveraged PDPs to enhance fairness by dynamically managing network traffic. Among these, P4air [11], P4CCI [12], and P4BS [13] employ PDP-based techniques to mitigate congestion-induced unfairness. However, these methods vary in their classification strategies, fairness objectives, and deployment complexity. This section compares these existing approaches and the proposed system, highlighting their respective strengths and limitations in addressing RTT-based unfairness.

5.1. Flow Classification Approach

P4air categorizes flows based on the CCA in use, aiming to mitigate unfairness caused by competition between different CCAs. However, this approach does not address intra-CCA RTT disparities, which can still lead to significant bandwidth imbalances among flows using the same congestion control strategy. P4CCI takes a different approach by detecting and isolating aggressive CCAs, ensuring that dominant flows do not degrade the performance of others. Although effective in CCA-based fairness enforcement, P4CCI does not distinguish between flows of the same CCA with differing RTTs, potentially allowing short-RTT flows to monopolize bandwidth. P4BS, instead of focusing on flow classification, dynamically resizes router buffers based on real-time traffic conditions. While this method helps prevent excessive queueing delays, it does not explicitly classify flows based on RTT. The proposed system introduces a fundamentally different classification approach by separating flows based on RTT rather than CCA type. This ensures that short-RTT flows do not dominate bandwidth allocation, achieving fairer resource distribution independent of congestion control strategy.

5.2. Fairness Improvement

In terms of fairness improvement, P4air enhances fairness by 5–10% in mixed-CCA environments by ensuring that no single CCA dominates link capacity. However, because it does not classify flows within the same CCA, intra-CCA unfairness remains unresolved. P4CCI achieves 8–12% fairness improvement by identifying and mitigating aggressive CCAs, thereby preventing congestion collapse scenarios caused by uncontrolled transmission rates. P4BS, while primarily focused on buffer management, improves fairness by 10% through adaptive queue sizing, which indirectly helps balance traffic loads. In contrast, the proposed RTT-based classification method achieves a 15% fairness improvement, surpassing all prior approaches. By dynamically grouping flows with similar RTTs into dedicated queues, the system prevents short-RTT flows from crowding out long-RTT flows, ensuring a more equitable bandwidth distribution across diverse traffic patterns.

5.3. Flow Completion Time Reduction

Flow Completion Time (FCT) is a critical performance metric for long TCP flows. P4BS effectively reduces queueing delay and packet loss, thereby improving latency-sensitive traffic. However, it does not significantly optimize long-flow completion times, as it primarily addresses buffer management rather than flow-level fairness. The proposed system, by ensuring balanced bandwidth allocation through RTT-aware queueing, achieves a 20% reduction in FCT for large TCP flows. By mitigating the dominance of shorter-RTT flows, the system enables long-RTT flows to utilize a fair share of bandwidth, reducing their completion time.

5.4. Overhead and Deployment Feasibility

One major drawback of CCA-based approaches like P4air and P4CCI is their constant per-packet analysis of flow behavior, which introduces additional processing overhead on programmable switches. Since CCAs rely on dynamic congestion windows and loss-based adaptation, these systems require continuous monitoring of packet inter-arrival times, retransmission patterns, and throughput variations. This real-time classification results in higher computational demands. P4BS, on the other hand, introduces control-plane overhead due to its frequent buffer size updates, which require continuous interaction between the data plane and control plane. The proposed system has a lower overhead, as RTT computation and classification occur passively in the data plane, requiring minimal intervention from the control plane. Since the classification is based on packet timestamps, rather than congestion behavior, it is computationally lighter than CCA-based classification, making it more scalable and efficient for deployment in high-speed networks.

6. Limitations

While the proposed system demonstrates significant improvements in fairness, latency, and flow completion time, several limitations must be considered. These include scalability challenges due to memory constraints, computational overhead associated with real-time classification, and dependency on PDPs. Additionally, while the Jenks natural breaks algorithm offers an efficient classification mechanism, more advanced machine learning-based classification could further enhance adaptability. This section discusses these limitations and outlines potential future enhancements.

1. **Scalability and Memory Constraints**: The system operates within the memory constraints of programmable switches, which have limited on-chip

SRAM for per-flow state storage. To mitigate this limitation, the system employs Count-Min Sketch and multi-stage hashing to efficiently track RTT statistics while reducing memory overhead. However, in high-traffic environments with tens of thousands of concurrent flows, the accuracy of these approximations may degrade, impacting classification precision. Future optimizations include hierarchical aggregation techniques and offloading flow tracking to SmartNICs to extend scalability without significantly increasing memory usage.

2. **Impact of Delayed ACKs and Incomplete Flow Data**: While the system accurately measures RTT by correlating TCP sequence (SEQ) and acknowledgment (ACK) numbers, it faces challenges with delayed ACK mechanisms, which can affect the precision of RTT calculations. Furthermore, the eviction of flow records due to memory constraints or timeouts can lead to incomplete flow data, potentially resulting in suboptimal flow classification and bandwidth allocation.

3. **Rule Management Overhead**: The system performs real-time flow identification, RTT computation, and classification within the data plane of a PDP. While the match-action pipeline processes packets at line rate, there is an inherent computational overhead associated with timestamping, RTT correlation, and dynamic queue allocation. To minimize latency, batch rule updates are applied to avoid frequent reconfiguration overheads. However, in environments with high traffic churn, excessive control-plane interactions may introduce processing delays. Future enhancements include leveraging programmable hardware accelerators such as SmartNICs to offload classification and reduce overhead.

4. **Applicability to Non-Programmable Routers**: Although the system can interact with non- programmable routers through passive traffic monitoring and management port adjustments, its functionality remains limited by the capabilities of these routers. In particular, the system's ability to control queue sizes and configurations in non-progra -mmable routers is dependent on their support for such features. If the router lacks advanced queue management capabilities, the system may not fully realize its potential in optimizing traffic flows.

The system depends on PDPs to perform real-time traffic classification and enforcement. While PDPs are increasingly available in modern data centers, their adoption remains limited in legacy enterprise and ISP networks. This dependency may restrict deployment in environments that rely solely on non-programmable routers. As an alternative, the system could be integrated with SmartNICs or software-based traffic monitoring agents, allowing RTT classification to be performed externally without modifying legacy infrastructure. A hybrid model, where a centralized SDN controller aggregates RTT data across multiple routers, could also extend the system's applicability to non-programmable networks. The system employs the Jenks natural breaks algorithm due to its low computational cost and ability to classify flows into RTT-based clusters. While this approach provides efficient real-time classification, it lacks the adaptability of machine learning (ML)-based techniques, which could further refine RTT-based queue assignment. Future work will explore unsupervised clustering methods, such as k-means or DBSCAN, to enhance classification granularity while maintaining low latency. Additionally, lightweight neural-network-based classifiers could be investigated to dynamically adjust classification thresholds in response to changing network conditions.

Deploying this system in real-world networks presents several challenges, including hardware compatibility, integration with existing traffic management policies, and operational overhead. While PDP-based classification is feasible in enterprise and data center environments, large-scale ISP deployments may require incremental adoption strategies to ensure seamless integration. Future work will focus on conducting real-world trials in testbed environments to evaluate long-term performance under diverse traffic conditions. Additionally, integrating this solution with controller-based network management platforms (e.g., SDN controllers) can facilitate deployment in operational networks.

7. Conclusion and Future Work

The proposed system introduces an RTT-based flow classification mechanism to enhance fairness in non- programmable networks. By leveraging PDPs, the system passively measures RTTs, dynamically classifies flows using Jenks natural breaks, and enforces queue assignments to mitigate RTT-induced unfairness. Unlike prior methods such as P4air [11], which classifies flows based on Congestion Control Algorithms (CCAs), or P4BS [13], which dynamically resizes buffers, the proposed approach explicitly separates flows based on RTTs, ensuring that shorter-RTT flows do not dominate available bandwidth.

Experimental evaluations demonstrate that the proposed system achieves a 15% improvement in fairness compared to existing PDP-based techniques. Specifically, while P4air and P4CCI [12] provide 5–12% fairness improvements by isolating competing CCAs, they do not address fairness among flows of the same CCA but with different RTTs. Additionally, our system reduces Flow Completion Time (FCT) by 20% for long TCP flows, outperforming P4BS, which primarily optimizes queueing delay but does not systematically improve FCT. Furthermore, our system introduces lower computational overhead, as it passively monitors RTTs rather than relying on per-packet CCA behavior analysis, making it more efficient for large-scale network deployments.

These results confirm that RTT-aware classification is a critical factor in achieving network fairness, complementing and surpassing existing approaches that focus solely on CCAs or buffer management. Future work will explore further refinements, including integration with SmartNICs for enhanced scalability and machine-learning-based flow classification for adaptive fairness adjustments. By providing a practical and low-overhead solution to RTT fairness, the proposed system offers a deployable framework for improving performance in diverse networking environments.

8. Acknowledgement

The authors would like to acknowledge the National Science Foundation (NSF) for supporting this work, under award 2346726. The authors would also like to acknowledge the FABRIC [36] team for facilitating the platform used for some preliminary tests.

References

[1] J. Postel, "RFC 793: Transmission Control Protocol (TCP)," in *RFC Editor*, 1981.

[2] E. Gavaletz and J. Kaur, "Decomposing RTT-unfairness in transport protocols," in *17th IEEE Workshop on Local & Metropolitan Area Networks (LANMAN)*, 2010.

[3] S. Ha, I. Rhee, and L. Xu, "CUBIC: a new TCP-friendly high-speed TCP variant," *ACM SIGOPS operating systems review*, 2008.

[4] N. Cardwell, Y. Cheng, C. Gunn, S. Yeganeh, and V. Jacobson, "BBR: congestion-based congestion control," *Communications of the ACM*, 2017.

[5] J. Gomez, E. Kfoury, J. Crichigno, and G. Srivastava, "Evaluating TCP BBRv3 performance in wired broadband networks," *Computer Communications*, vol. 222, 2024.

[6] E. Kfoury, J. Gomez, J. Crichigno, and E. Bou-Harb, "An emulation-based evaluation of TCP BBRv2 alpha for wired broadband," *Computer Communications*, 2020.

[7] J. Gomez, E. Kfoury, J. Crichigno, E. Bou-Harb, and G. Srivastava, "A performance evaluation of TCP BBRv2 alpha," in *43rd International Conference on Telecommunications and Signal Processing (TSP)*, 2020.

[8] Y. Tao, J. Jiang, S. Ma, L. Wang, W. Wang, and B. Li, "Unraveling the RTT-fairness problem for BBR: A queueing model," in *IEEE Global Communications Conference (GLOBECOM)*, 2018.

[9] J. Gettys, "Bufferbloat: Dark buffers in the internet," *IEEE Internet Computing*, 2011.

[10] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, and G. Varghese, "P4: Programming protocol-independent packet processors," *ACM SIGCOMM Computer Communication Review*, 2014.

[11] B. Turkovic and F. Kuipers, "P4air: Increasing fairness among competing congestion control algorithms," in *IEEE 28th International Conference on Network Protocols (ICNP)*, 2020.

[12] E. Kfoury, J. Crichigno, and E. Bou-Harb, "P4CCI: P4-based online TCP congestion control algorithm identification for traffic separation," in *IEEE International Conference on Communications (ICC), Rome, Italy*, 2023.

[13] E. Kfoury, J. Crichigno, and E. Bou-Harb, "P4BS: Leveraging passive measurements from P4 switches to dynamically modify a router's buffer size," *IEEE Transactions on Network and Service Management*, 2023.

[14] G. Jenks, "The data model concept in statistical mapping," *International yearbook of cartography*, 1967.

[15] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "OpenFlow: Enabling innovation in campus networks," *ACM SIGCOMM Computer Communication Review*, 2008.

[16] E. Kfoury, J. Crichigno, and E. Bou-Harb, "An exhaustive survey on P4 programmable data plane switches: Taxonomy, applications, challenges, and future trends," *IEEE Access*, 2021.

[17] A. AlSabeh, J. Khoury, E. Kfoury, J. Crichigno, and E. Bou-Harb, "A survey on security applications of P4 programmable switches and a STRIDE-based vulnerability assessment," *Computer Networks*, 2022.

[18] J. Gomez, E. Kfoury, J. Crichigno, and G. Srivastava, "A survey on TCP enhancements using P4-programmable devices," *Computer Networks*, 2022.

[19] E. Kfoury, J. Crichigno, E. Bou-Harb, and G. Srivastava, "Dynamic router's buffer sizing using passive measurements and P4 programmable switches," in *IEEE Global Communications Conference (GLOBECOM)*, 2021.

[20] E. Kfoury, J. Crichigno, E. Bou-Harb, D. Khoury, and G. Srivastava, "Enabling TCP pacing using programmable data plane switches," in *42nd International Conference on Telecommunications and Signal Processing (TSP)*, 2019.

[21] S. Ma, J. Jiang, W. Wang, and B. Li, "Fairness of congestion-based congestion control: Experimental evaluation and analysis," *arXiv preprint arXiv:1706.09115*, 2017.

[22] G. Appenzeller, I. Keslassy, and N. McKeown, "Sizing router buffers," *ACM SIGCOMM Computer Communication Review*, 2004.

[23] X. Chen, H. Kim, J. Aman, W. Chang, M. Lee, and J. Rexford, "Measuring TCP round-trip time in the data plane," in *Proceedings of the Workshop on Secure Programmable Network Infrastructure*, 2020.

[24] J. Dugan, S. Elliott, B. Mah, J. Poskanzer, and K. Prabhu, "iPerf3, tool for active measurements of the maximum achievable bandwidth on IP networks." [Online]. Available: `https://github.com/esnet/iperf` Accessed on 04-20-2024.

[25] Juniper Networks, "MX204 universal routing platform." [Online]. Available: `https://tinyurl.com/yz86p3vx`, Accessed on 08-14-2023.

[26] Edgecore Networks, "Wedge 100BF-32X." [Online]. Available: `https://tinyurl.com/2xay8kky`, Accessed on 04-20-2024.

[27] J. Gomez, E. Kfoury, J. Crichigno, and G. Srivastava, "Improving TCP fairness in non-programmable networks using P4-programmable data planes," in *IEEE International Black Sea Conference on Communications and Networking (BlackSeaCom)*, 2024.

[28] MAWI Working Group Traffic Archive, "Packet traces from WIDE backbone." [Online]. Available: `https://tinyurl.com/2e88vw2v`, Accessed on 04-04-2024.

[29] A. Mishra, X. Sun, A. Jain, S. Pande, R. Joshi, and B. Leong, "The great internet TCP congestion control census," *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, 2019.

[30] D. Leith and R. Shorten, "H-TCP: TCP for high-speed and long-distance networks," in *Proceedings of PFLDnet*, 2004.

[31] Google, "TCP BBRv2 Alpha/Preview Release." [Online]. Available: `https://github.com/google/bbr/tree/v2alpha`, Accessed on 04-20-2024.

[32] L. Kleinrock, "Internet congestion control using the power metric: Keep the pipe just full, but no fuller," *Ad hoc networks*, 2018.

[33] A. Mazloum, A. AlSabeh, E. Kfoury, and J. Crichigno, "perfSONAR: Enhancing data collection through adaptive sampling," in *IEEE Network Operations and Management Symposium (NOMS)*, 2024.

[34] D. Thomas, R. Clayton, and A. Beresford, "1000 days of UDP amplification DDoS attacks," in *APWG Symposium on Electronic Crime Research (eCrime)*, 2017.

[35] A. AlSabeh, E. Kfoury, J. Crichigno, and E. Bou-Harb, "P4DDPI: Securing P4-programmable data plane networks via DNS deep packet inspection," in *NDSS Symposium*, 2022.

[36] I. Baldin, A. Nikolich, J. Griffioen, I. Monga, K.-C. Wang, T. Lehman, and P. Ruth, "FABRIC: A national-scale programmable experimental network infrastructure," *IEEE Internet Computing*, vol. 23, 2019.