Elephants Sharing the Highway: Studying TCP Fairness in Large Transfers over High Throughput Links

Imtiaz Mahmud
Lawrence Berkeley National Laboratory
California, USA
imahmud@lbl.gov

George Papadimitriou
Information Sciences Institute, University of Southern California
California, USA
georgpap@isi.edu

Cong Wang
RENCI, University of North Carolina at Chapel Hill
North Carolina, USA
cwang@renci.org

Mariam Kiran
Oak Ridge National Laboratory
Tennessee, USA
kiranm@ornl.gov

Anirban Mandal
RENCI, University of North Carolina at Chapel Hill
North Carolina, USA
anirban@renci.org

Ewa Deelman
Information Sciences Institute, University of Southern California
California, USA
deelman@isi.edu

ABSTRACT

Escalating bandwidth demand strains high-performance data networks, posing potential performance risks. TCP congestion control algorithms enhance reliability and optimize bandwidth usage. Network performance is influenced by factors such as AQM algorithms and router buffer size. In the context of constrained network resources, understanding how TCP flows share networks and the resulting performance impact is essential.

This paper introduces insights into TCP fairness and performance involving a comparison of TCP CUBIC, Reno, Hamilton, and BBR versions 1 and 2 across real-world networks supporting high bandwidths of up to 25 Gbps. The research explores TCP behaviors with AQM algorithms like FIFO, FQ_CODEL, and RED, alongside diverse buffer sizes. Notably, findings reveal that manipulating buffers and queuing methods yields contrasting outcomes based on bandwidth. BBRv2 emerges as a superior fair algorithm, pivotal for swift transfers, particularly in scientific data scenarios. These results provide crucial guidance for future network design, ensuring equitable performance optimization.

KEYWORDS

TCP Congestion Control, Fairness, Active Queue Management, Buffer Size, FABRIC, High-Bandwidth, High-Speed Internet, Elephant Flows

ACM Reference Format:
Imtiaz Mahmud, George Papadimitriou, Cong Wang, Mariam Kiran, Anirban Mandal, and Ewa Deelman. 2023. Elephants Sharing the Highway: Studying TCP Fairness in Large Transfers over High Throughput Links. In *Workshops of The International Conference on High Performance Computing, Network, Storage, and Analysis (SC-W 2023), November 12–17, 2023, Denver, CO, USA.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3624062.3624594

SC-W 2023, November 12–17, 2023, Denver, CO, USA
© 2023 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-0785-8/23/11.
https://doi.org/10.1145/3624062.3624594

1 INTRODUCTION

The rapid emergence of IoT devices, edge-to-cloud integration and high-performance computational workflows, have brought an exponential growth of demand for reliable data transfer across the wide area networks [3, 14]. Transmission Control Protocol (TCP) is one of the most commonly used transport protocols designed to ensure minimum packet loss and maximum use of available bandwidth (BW) in high volume data transfers [23, 36]. Network researchers have developed innovative TCP variants that try to achieve high throughput with minimal packet loss. However, as networks become congested and resources are limited, links have to be shared among many flows, which requires an understanding of the correct choices for TCP congestion control algorithms (CCA).

The TCP CCA controls network resource utilization by regulating the amount of data being transmitted [2, 29]. Existing variants of CCAs, such as Bottleneck BW and Round-trip propagation time version 1 (BBRv1) [7], BBR version 2 (BBRv2) [8], Hamilton TCP (HTCP) [27], CUBIC [20], and Reno [24], are being extensively used and studied, but they still witness changing network conditions having high retransmissions, significant delay, low throughput, and wastage of network resources [29]. On the other hand, Active Queue Management (AQM) algorithms manage the queue of packets at routers and shape the traffic [1]. AQMs help prevent network congestion by monitoring the queue length and make decisions about which packets to drop. When packets are dropped, CCAs interpret this as a congestion signal, leading to a reduction in the rate at which data is transmitted. Some AQMs implement explicit congestion notifications (ECNs) [16], which provide more specific information to the CCAs about upcoming network congestion. ECNs signal to the CCA that it should reduce the flow of data without waiting for packets to be dropped, leading to more efficient use of network resources. Thus, AQMs are an important tool for ensuring network stability and efficient resource utilization.

Historically, the Research and Education (R&E) community has deployed network devices with deep buffers to avoid packet loss in large science transfers [40]. However, using a buffer size by determining the Bandwidth Delay Product (BDP), proves to be a non-scalable solution, especially as network bandwidth increases up to 400G. Additionally as R&E networks become congested, we see that short-term dynamics of competing high-speed TCP flows can have strong impacts on their long-term fairness leading to severe challenges in co-existence and network deployments [30]. With the growing availability of higher BW and the need to efficiently handle big data, such as in scientific R&E networks, it is crucial to assess the performance of different TCP CCAs in real-world high-BW scenarios. Evaluating how these interact with various AQMs is essential. Additionally, TCP variants are being developed with different goals in mind, which results in different heuristics and overall algorithm behavior. One example is BBR, which was developed to avoid buffer bloat and to provide more stable performance in streaming applications, and is not significantly impacted by queue length like CUBIC. However, BBR can experience performance degradation when multiple TCP flows attempt to share the same network link. A detailed analysis of these relationships and the impact on fairness is needed to improve performance in high-BW scenarios. Based on the above need, we aim to investigate TCP fairness and its effects by examining different AQM algorithms and BW configurations. To achieve this, we conducted controlled experiments using different TCP CCAs and AQM algorithms in real-networks with a wide range of BWs. The FABRIC testbed [4], an inter-continental real-network instrument connecting laboratories from Japan to Europe through USA, was used for this. This paper makes the following contributions:

- We evaluate the TCP CCA fairness in network links through real-world experimentation using a bottleneck topology. We evaluate TCP CCAs (BBRv1, BBRv2, Reno, HTCP) against CUBIC, in different BW scenarios, varying router queue length, and using different AQM algorithms including FIFO, FQ_CODEL, and RED.
- We assess the scaling capability of BBRv1, BBRv2, CUBIC, Reno, and HTCP in TCP sharing experiments in different BW scenarios.
- We identify limitations in the TCP performance particularly with regards to FQ_CODEL and RED's failure to fully utilize the capacity in high-BW links.
- We share a reproducible dataset that contains experimental codes and all end-to-end log files, which can be useful for future research, especially for developing, training, and testing TCP Machine Learning (ML) models. This repository of TCP logs and network performance data contributes to design of future TCP algorithms and efficient network design for high-BW scenarios.

The remainder of the paper is structured as follows: Section 2 provides the motivation behind this work, while Section 3 summarizes the fundamentals of TCP and TCP fairness challenges. Section 4 describes the experimental setup in detail. Section 5 presents the findings and discusses the results. Finally, Section 6 provides concluding remarks.

2 MOTIVATION

Modern large scale science workflows often include movements of large volumes of data generated through scientific instruments, ranging from small datasets to massive amounts of information. The data transfer requirements vary from real-time, short-term transmission for immediate processing, to long term data transfer over inter-continental network links. The variety of data includes experimental results, sensor readings, genomic data, imaging data, and more. Efficient handling of volume, speed, and variety of data is crucial for optimal data transfer performance.

In the R&E community, CUBIC and HTCP have been favourably used for high-speed data transfers within the Science DMZ [9]. These handle very large transfers but need to guarantee loss-less transmissions [40]. Monitoring transmission rates based on network congestion, affecting throughput, latency, and fairness are high priority for these networks. The selection of optimal configurations can impact how science networks are designed and maintained. For instance, applications requiring low latency, real-time collaboration, or fair BW sharing may benefit from algorithms prioritizing responsiveness and fairness. Conversely, applications emphasizing high throughput and efficient bulk data transfer may prefer algorithms that maximize network capacity. Science and research networks are also constrained by resources and face substantial data transfer requirements. While TCP CCAs control end-to-end data transfers, the decision regarding which packets to retain or discard during data transfer is made by the point-to-point AQM algorithms at the routers. These packet drops significantly impact the behavior of TCP CCAs, resulting in a strong interconnection between them. On the other hand, elephant flows are very common in science networks, which is not as common in commercial networks. This disparity makes it more challenging for science networks to manage such elephant flows using traditional CCA and AQM algorithms. Consequently, we are motivated to investigate the interactions between CCA and AQM algorithms and determine the optimal combination. By studying these combined conditions, we can enhance network utilization through efficient design choices.

3 BACKGROUND AND RELATED WORK

3.1 TCP Congestion Control Algorithms

TCP is employed for data transmission and plays a crucial role in averting network congestion by managing the rate at which data is sent. The research community has put forward numerous TCP CCAs. For this experiment, we have opted to focus on the widely utilized and already implemented CCAs found within the Linux Kernel. These include Reno, CUBIC, BBRv1, BBRv2, and HTCP. A brief overview of each is provided in this section.

3.1.1 ***BBRv1.*** BBRv1 was developed by Google in 2016. It estimates the available bottleneck BW and round-trip propagation time of a network path and uses this information to adjust the sending rate to maximize network utilization and minimize congestion. BBRv1 uses a hybrid approach that combines delay-based and loss-based congestion control techniques [7]. One of the main criticisms of BBRv1 is that it tends to overshoot the network capacity, which can result in high packet losses and reduced performance [21][6]. BBRv1 aggressively ramps up the sending rate to the maximum achievable BW at the start of a connection, which can lead to congestion and decreased fairness for later started flows that have not yet had a chance to ramp up their sending rate [21]. Additionally, BBRv1 has been found to cause retransmission timeouts (RTOs) in some cases, which can also result in reduced performance and increased latency [6]. Furthermore, BBRv1 requires accurate round-trip time (RTT) measurements to estimate the available BW, which can be challenging in networks with high latency or fluctuating RTTs, and may not be well supported by all network devices and applications [7]. Finally, the utilization of continuous network modeling necessitates a certain amount of time, potentially leading to adverse effects on short-lived flows. This aspect could result in BBRv1 being better suited as for elephant flows scientific networks.

3.1.2 ***BBRv2***. BBRv2 is the newer version of Google's BBR CCA [8], building upon the delay-based and loss-based hybrid approach of BBRv1. While BBRv1 already includes packet pacing, prioritization, and probing, BBRv2 further refines these features to enhance performance. Pacing in BBRv2 allows for more precise control over the sending rate by regulating the transmission of packets, reducing packet loss, and improving congestion control. Probing in BBRv2 provides a more accurate estimation of available BW by actively probing the network with small bursts of packets. BBRv2 also supports ECN as a congestion signaling mechanism, allowing it to react more quickly and accurately to network congestion. These improvements in BBRv2 contribute to better throughput, reduced retransmissions and latency, and improved fairness in sharing BW among flows, further refining the congestion control capabilities of BBRv1. Moreover, because it is based on BBRv1, it is also expected to show better performance for elephant flows.

While BBRv2 offers many improvements over BBRv1, it also has some potential downsides. For example, the use of packet pacing in BBRv2 can result in increased latency, especially for short flows that may not fully utilize the available BW [37]. Song et al. [37] reported that BBRv2 enhances fairness and reduces aggressiveness in small buffers of less than 1 BDP. Although BBRv2 shows improvement in terms of fairness in sharing BW and lower rates of packet retransmissions, certain challenges such as RTT unfairness, coexistence with loss-based algorithms, and synchronization between BBRv2 flows persist.

3.1.3 ***CUBIC***. TCP CUBIC, the default CCA in Linux, estimates the number of packets that can be sent without causing congestion in the network using a cubic function based on the current RTT and the number of packets sent since the last congestion event [20]. It responds to packet loss by reducing the congestion window (CWND) size and entering a recovery phase where it uses a binary search algorithm to find the appropriate window size. TCP CUBIC is scalable to high-speed and high-BW networks. While TCP CUBIC has been proven to exhibit fairness towards all flows within a network, it can still display aggressive tendencies under certain conditions. This aggressiveness may lead to issues such as bufferbloat and unfairness in the overall network dynamics [42]. CUBIC generally demonstrates equitable behavior across all flows, for elephant flows, a well-balanced output is still anticipated from this algorithm.

3.1.4 ***Reno***. TCP Reno is a loss-based CCA that reacts to packet loss and manages network congestion using slow start, congestion avoidance, fast retransmit, and fast recovery mechanisms [24]. When packet loss is detected, TCP Reno reduces its sending rate by half and enters the fast recovery state. If it receives an acknowledgement (ACK) for the lost packet, it exits the fast recovery state and resumes congestion avoidance. However, if it does not receive the ACK within a specified time, it assumes that the congestion persists and enters the slow start state [18]. TCP Reno is effective in managing congestion in various network environments but has some drawbacks, such as poor performance in high-BW networks and fairness issues with other protocols. This is due to its conservative CWND increase strategy, which limits the sending rate to avoid congestion.[7]. Consequently, TCP Reno might not be the optimal choice for handling elephant flows within scientific networks.

3.1.5 ***HTCP***. The Hamilton TCP (HTCP) is a CCA designed for use in high-speed networks with large buffer sizes [27]. It is based on a feedback control system that uses a combination of additive increase and multiplicative decrease to adjust the sending rate of the TCP flow. It is designed to increase the aggressiveness of TCP in high-BW, high-latency networks while maintaining fairness and compatibility with other TCP CCAs. HTCP dynamically adjusts its aggressiveness by increasing the rate of additive increase as the time since the previous loss increases, allowing it to quickly respond to congestion events and improve throughput. HTCP provides a fast convergence to a stable sending rate and maintains low queuing delay in high-speed networks [28]. Additionally, HTCP uses a simple feedback control mechanism that requires minimal computational overhead and is easy to implement. Furthermore, HTCP is compatible with existing TCP flows, so it can coexist with other CCAs without causing interference. Therefore, HTCP can be considered as one of the best candidate to provide high network utilization in scientific networks, especially for elephant flows. However, it may not be suitable for networks with a large number of flows or non-TCP traffic and is vulnerable to unfairness issues when competing with other CCAs [5].

3.2 Challenges in TCP Fairness

Numerous researchers have extensively examined TCP performance across different network scenarios [33][37][12][41][22][21]. However, the majority of these investigations have taken place within simulated environments [33][37][12]. Real-world experiments provide an authentic portrayal of network dynamics, including real-time flow changes, complex physical networking reactions, and unforeseen anomalies, which cannot be precisely replicated in simulations. Consequently, conducting networking experiments in actual networks holds significant importance and offers distinct advantages over simulations. Interestingly, very few research were conducted involving real-world experiments, often conducted on a smaller scale [33][37][41]. Moreover, only a limited number of studies have delved into the performance of TCP CCAs within high-BW scenarios surpassing Gbps [37][22] [21]. Notably, the exploration of TCP behavior within the context of scientific networks, particularly regarding elephant flows, remains a relatively unexplored area. While the dynamics of TCP fairness in conventional internet settings have sparked considerable debate [11], it is imperative to prevent a single TCP flow from monopolizing resources and compromising the performance of other concurrent flows.

3.3 Active Queue Management Algorithms

AQMs are implemented in network routers to prevent buffer overflow and packet losses, contributing to the establishment of a more

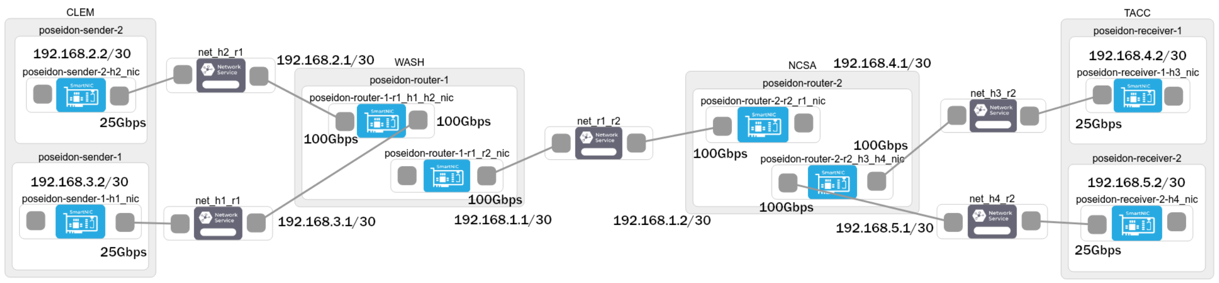

Figure 1: The experimental topology on FABRIC, with nodes located across 4 different locations connected over a Layer 2 network. RTT ≈ 62ms between Clemson and TACC.

dependable network infrastructure. These AQMs also wield significant influence over CCAs, as dropped packets serve as indicators of congestion for CCAs. This symbiotic relationship aids in congestion prevention and ensures the uninterrupted flow of data [15]. While numerous AQM algorithms exist, we will focus on FIFO, RED, and FQ_CODEL due to their prominence and representation of three distinct and widely used categories. Specifically, FIFO serves as a fundamental AQM algorithm, RED employs a probabilistic approach, and FQ_CODEL creates individual queues for each flow while applying the CoDel mechanism to the queues. This section provides a succinct overview of these algorithms.

3.3.1 ***FIFO***. FIFO (First-In-First-Out) is the simplest and oldest AQM algorithm [17], where packets are dropped from the front of the queue as soon as the queue is full. This algorithm does not require any complex processing, making it easy to implement in hardware and software. However, it has several limitations, such as the inability to differentiate between different types of traffic and the lack of support for controlling queue length or delay [32]. Moreover, this algorithm can cause TCP to detect packet loss as a signal of congestion, leading to reduced throughput and unfairness [17]. Despite its limitations, FIFO is still used in many networks due to its simplicity and low overhead.

3.3.2 ***FQ_CODEL***. FQ_CODEL (Fair Queuing with Controlled Delay) is a combination of the fair queuing (FQ) and CoDel (controlled Delay) algorithms, designed to provide fair and low-delay packet transmission in a multi-flow environment. In this algorithm, each flow is assigned to a virtual queue and processed in round-robin fashion to ensure fair BW allocation [39]. CoDel is used to manage the buffer delay by monitoring the time spent by packets in the queue and dropping packets before the buffer becomes full [31]. This algorithm provides significant benefits over FIFO, such as low latency, less packet loss, and better support for multiple flows. However, it requires more processing power and is more complex to implement than FIFO.

3.3.3 ***RED***. RED (Random Early Detection) is an AQM algorithm that drops packets from the queue randomly before it becomes full, based on the total length of the queue and the available average queue length [17]. This algorithm aims to provide congestion avoidance by controlling the queue length and preventing it from becoming too large. RED monitors the queue length and drops packets probabilistically, with a higher probability of dropping packets when the queue is close to the maximum threshold [13]. This algorithm provides benefits over FIFO by reducing the likelihood of congestion and improving the fairness of BW allocation. However, RED requires careful tuning of its parameters to ensure optimal performance, and it may not work well in environments with bursty traffic [35].

4 EXPERIMENTAL SETUP

4.1 Experimental Infrastructure

We used the FABRIC testbed to perform the experiments [4], which is a nationwide instrument funded by the National Science Foundation (NSF) of the USA to enable large scale experimentation within an isolated, yet realistic environment. It provides compute and storage resources on multiple sites that are interconnected by high-speed, dedicated optical links. FABRIC also offers “everywhere programmability”, as all of its resources can be adapted and customized for each experiment’s needs. Finally, an experiment on FABRIC can be designed using FABlib, a Python API that exposes FABRIC’s resources as Python objects, allowing for an intuitive and highly reproducible experimental setup.

We designed a dumbbell topology that contains 6 Linux virtual machines across 4 FABRIC sites, depicted in Figure 1. We spawned 2 client nodes generating traffic at the Clemson University (CLEM), 1 router in Washington (WASH), 1 router at the National Center for Supercomputing Applications (NCSA), and 2 server nodes receiving traffic at the Texas Advanced Computing Center (TACC). All the client and server nodes were equipped with 26 vCPUs, 32 GB RAM and 1 dedicated Mellanox ConnectX-5 NIC with 25 Gbps Ethernet. The nodes for routing traffic contained 24 vCPUs, 32 GB RAM and 2 dedicated Mellanox ConnectX-6 NICs with 2 100 Gbps Ethernet ports. The connectivity of our topology was established over a Layer 2 network service and the RTT between Clemson and TACC was measured to be about 62 milliseconds. We applied our own Layer 3 network with 5 different subnets, and to establish successful connections between client and servers, we enabled packet forwarding on the routing nodes and introduced static routing rules from and to all subnets. All of the nodes were based on Ubuntu 20.04 LTS, and iperf3 version 3.7 had been installed to all the client and server nodes. Finally, all of the nodes were running a custom version of the Linux kernel with BBRv2 support [19].

Table 1 presents the network settings set. We collected statistics for 810 distinct configurations, and for each configuration, the

Table 1: network configuration settings

CCA 1 - CCA 2	AQM	Queue Length	Bottleneck BW
BBRv1 - CUBIC			
BBRv2 - CUBIC	FIFO	0.5 x BDP	100 Mbps
HTCP - CUBIC		1 x BDP	500 Mbps
Reno - CUBIC	FQ CODEL	2 x BDP	1 Gbps
CUBIC - CUBIC		4 x BDP	10 Gbps
BBRv1 - BBRv1	RED	8 x BDP	25 Gbps
BBRv2 - BBRv2		16 x BDP	
HTCP - HTCP			
Reno - Reno			

experiment was repeated and averaged over 5 times. The AQM type, the queue length and the transmission rate on $router_1$ was configured on the interface to the link with $router_2$. We used the Linux Traffic Control tool [38] to measure the performance across the bottleneck BW link.

Additionally, we calculated the queue length of $router_1$ as a function of BDP. We tested CUBIC flows against BBRv1, BBRv2, Reno and HTCP flows (inter-CCA experiments), with equal homogeneous distributions, and we also performed experiments where the flows in the network were of the same CCA (intra-CCA experiments). The iperf3 server processes were spawned on nodes residing in TACC and we were generating traffic from the nodes in Clemson using iperf3 client processes. In each experiment the flows were using jumbo packets of 8900 bytes and were running the test for 200 seconds. For the different bottleneck BW configurations we were also scaling the number of iperf3 processes and the number of parallel flows per iperf3 process, to avoid hitting any single thread performance limits, since iperf3 is single threaded. Table 2 presents the number of total flows, as well as the number of processes and parallel streams used to generate them, for each bottleneck BW scenario.

Table 2: iperf3 configuration per bottleneck bandwidth setting

Bottleneck BW	Total #Flows	iperf3 Configuration
100 Mbps	2	1 iperf3 process/node 1 stream
500 Mbps	10	5 iperf3 processes/node 1 stream each
1 Gbps	20	10 iperf3 processes/node 1 stream each
10 Gbps	200	10 iperf3 processes/node 10 parallel streams each
25 Gbps	500	25 iperf3 processes/node 10 parallel streams each

4.2 Calculating Bandwidth Delay Product

Setting the proper buffer size in a network queue is crucial for optimizing performance and managing congestion. The BDP plays a vital role in determining the appropriate network buffer sizes. It determines the maximum amount of data that can be in transit on a link at any given time [7]. By ensuring that the buffer size is appropriately aligned with the BDP, it enables effective accommodation of the inflight data and helps prevent packet losses [10]. This aids in maintaining smooth data flow, reducing buffer bloat, and minimizing latency. We calculated the BDP using the following equation to ensure proper buffer size configuration in our experiments, where $BW_{bottleneck}$ is measured in bits per second (bps), and RTT in measured milliseconds.

$$BDP = \frac{BW_{bottleneck} * RTT}{8} bytes \tag{1}$$

4.3 Reproducibility and Dataset

All of our experiments were fully automated using the FABlib API. We used a Jupyter notebook to design and deploy our topology on FABRIC, to control the changes of the network settings, and to spawn the iperf3 processes for each experiment run. All the scripts needed to reproduce our results and proof check our experimental method are available on GitHub [34]. Additionally, we provide the raw log files generated by iperf3 and the code to parse and plot the statistics.

5 RESULTS AND DISCUSSION

Here we analyze the performance of the CCAs and AQM algorithms in terms of fairness, per-sender throughput, overall link utilization, and packet retransmissions. We will present the figures that highlight the most significant observations, while the detailed results can be found in the GitHub repository [34]. Additionally, we will provide a concise explanation of the observed events, establish connections between them, and delve into the reasons underlying their behavior.

5.1 Fairness/Per-sender Throughput

Figure 2 presents the throughput achieved by BBRv1, BBRv2, Reno and HTCP against CUBIC while using FIFO as the AQM. Here we observe that BBRv1 throughput varies based on buffer sizes when competing with CUBIC. Similar to Hock et al. [21], the performance reveals equilibrium is achieved based on buffer sizes and BW. BBRv1 utilizes most of the available BW before the equilibrium point, while CUBIC gradually takes over. Notably, the equilibrium point shifts to the right with increasing BW, indicating a higher equilibrium point for higher BW capacity. For example, Figure 2(a)-(e) shows that the equilibrium point lies at a buffer size of 2 BDP for 100 Mbps BW, whereas it shifts to 3.5 BDP for 25 Gbps.

Additionally, we calculated the Jain's fairness index (J_{index}) to observe how effectively the senders share the underlying links [26][25]. J_{index} was computed using the following equation:

$$J_{index} = \frac{(\sum_{i=1}^{n} S_i)^2}{n * \sum_{i=1}^{n} S_i^2} \overset{n=2}{\Longrightarrow} J_{index} = \frac{(S_1 + S_2)^2}{2 * (S_1^2 + S_2^2)} \tag{2}$$

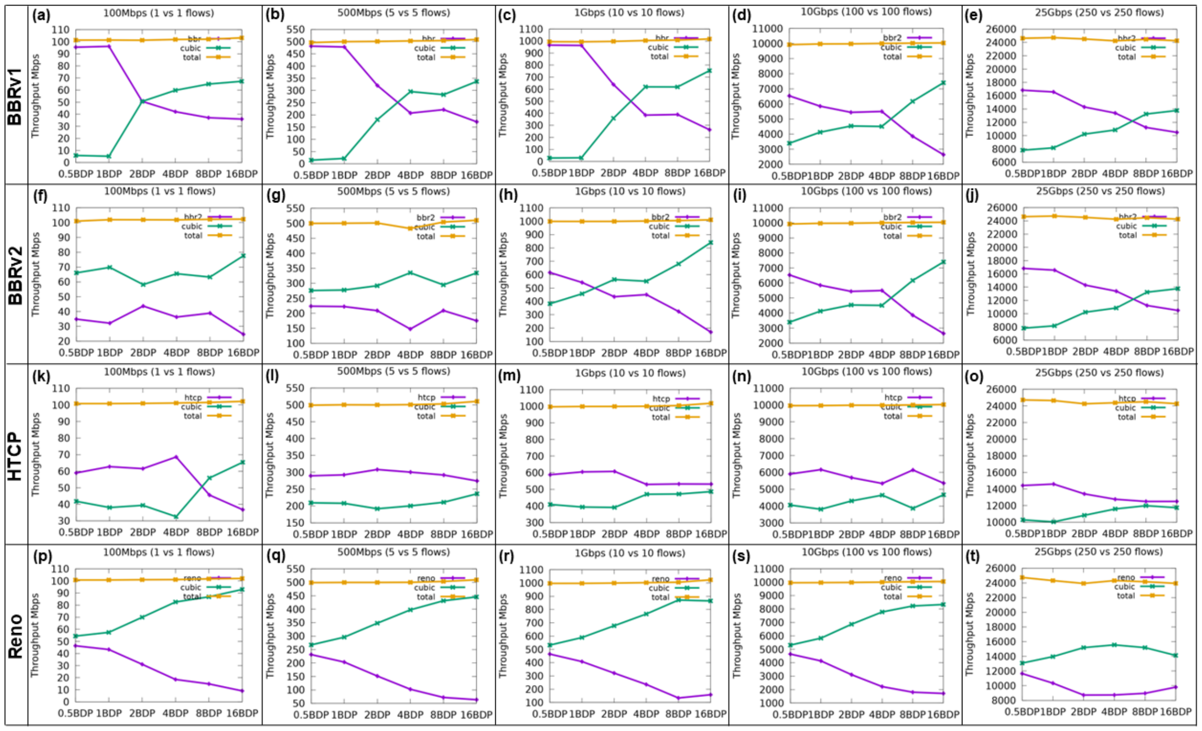

Figure 2: Per-sender throughput of TCP variants with Cubic, AQM=FIFO, (a) – (e) vs BBRv1, (f) – (j) vs BBRv2, (k) – (o) vs HTCP, and (p) – (t) vs Reno.

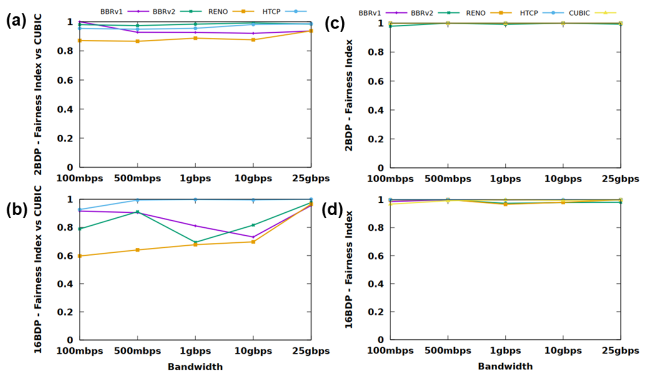

Figure 3: Observed Jain's fairness index for the considered CCAs when the AQM = FIFO: (a) – (b) inter-CCA, buffer size = 2 and 16 BDP, respectively, and (c) – (d) intra-CCA, buffer size = 2 and 16 BDP, respectively.

Here, S_i represents the throughput obtained by sender i, and n is the total number of flows going through the bottleneck. It is important to note that our focus revolves around assessing the equitable distribution of the bottleneck among different senders. In this pursuit, we have computed the per-sender J_{index}, where $n = 2$. The cumulative throughput achieved by each individual sender is represented as S_1 and S_2.

Figure 3(a) – (b) illustrates the J_{index} of BBRv1 competing with CUBIC under 2 BDP and 16 BDP. The fairness index of BBRv1 changes with BW and buffer sizes. This behavior is likely influenced by BBRv1's aggressive startup behavior and inflight cap [21][37]. Moreover, in the case of the 16 BDP buffer, it can be observed that fairness decreases significantly for BWs of 1 Gbps and 10 Gbps. With high BWs and such a large 16 BDP buffer size, CUBIC has a profound chance to overtake BBRv1 by occupying the buffer. Due to the 2 BDP inflight cap, BBRv1 struggles to compete effectively in filling the buffer in these scenarios. This additional buffer occupancy ultimately gives CUBIC the upper hand over BBRv1, as can be seen in Fig. 2(c) to (d). However, in the case of a 25 Gbps link, this gap is significantly smaller (Fig. 2(e)), resulting in better fairness. This could be related to the right shift in equilibrium points and CUBIC's inability to properly utilize the 25 Gbps link and fill the large 16 BDP buffer.

BBRv1's takeover. In contrast to CUBIC's slow start phase, BBRv1 rapidly increases its sending rate to fill the available buffer space and estimate available BW [21][37]. This causes CUBIC to experience early packet drops, leading to its exit from the startup phase and entering the recovery phase. Consequently, CUBIC exhibits significantly lower throughput at small buffer sizes. On the other hand, at large buffer sizes, BBRv1 does not utilize the entire buffer to maintain a 2 BDP inflight cap, while CUBIC gradually fills the available buffer space without such a cap. When using FIFO as the AQM, packet drops occur whenever the queue becomes full, without any separate queue or flow-specific logic. As a result, the ratio of queued packets for CUBIC becomes much higher compared to BBRv1 as the buffer size increases. Although BBRv1 does not directly consider packet loss as a congestion signal and does not reduce the sending rate, it also cannot increase the sending rate due to 2 BDP inflight cap. This ultimately gives CUBIC an advantage over BBRv1 in scenarios with large buffers. Additionally, both BBRv1 and CUBIC require a minimum buffer size to function optimally, and this minimum size increases significantly with higher BW capacities. It is worth noting that BBRv1 and CUBIC achieve almost equal throughput when BBRv1 is the only CCA operating, demonstrating fair BW sharing during intra-CCA experiments. A comparable results can be observed in Figure 3(c) – (d) in terms of J_{index}.

To further explore the results, the figures of per-sender throughput for intra-CCA experiments with FIFO, RED, and FQ_CODEL for all the considered CCAs, as well as the figures for inter-CCA experiments with FQ_CODEL, are available in the GitHub repository [34]. We encourage the readers to refer to the repository for a comprehensive view of the results.

BBRv2's takeover. Similar to BBRv1, BBRv2 exhibits a similar pattern when using FIFO as the AQM. Figure 2(f) – (j) shows that at 1, 10, and 25 Gbps BWs, CUBIC gains an advantage over BBRv2 after reaching buffer sizes of 1.5, 5, and 6 BDP, respectively. Prior to these buffer sizes, BBRv2 outperforms CUBIC, establishing similar equilibrium points. These findings align with Song et al.'s research [37]. However, their observed equilibrium points were around 0.25 and 0.4 BDP for 50 Mbps and 1 Gbps BWs, respectively. Based on their observations, we speculate that the equilibrium for 100 and 500 Mbps BWs lie below 0.5 BDP buffer sizes. Similar to BBRv1, this behavior can be attributed to BBRv2's aggressive startup phase. BBRv2 exhibits the same aggressive startup behavior as BBRv1, but with different criteria for exiting the startup phase. As CUBIC starts less aggressively than BBRv2, we observe similar behavior as with BBRv1. Figure 3(a) – (b) illustrates the changes in fairness index, further supporting these results.

In scenarios with large buffer sizes, BBRv2, like BBRv1, is limited by the 2 BDP inflight cap, which prevents it from fully utilizing the available queue at the bottleneck. In contrast, CUBIC can fill the buffer without such a cap, resulting in a higher ratio of queued packets for CUBIC compared to BBRv2. Since FIFO drops packets when the queue is full, packets from BBRv2 are also dropped, causing the drop rate to exceed BBRv2's 2% threshold. Unlike BBRv1, BBRv2 reacts by reducing its inflight_hi, which is the highest allowable inflight data or inflight cap. This reduction forces BBRv2 to decrease its sending rate in order to minimize packet losses. As a result, BBRv2 performs even worse than BBRv1 when competing with CUBIC at high BDP buffer sizes with FIFO. Similarly to BBRv1, no significant patterns were observed when BBRv2 operated alone, and both senders achieved almost equal throughput. We can obtain a supporting result for this by observing the J_{index} from (Figure 3(c) – (d)). For the same reasons mentioned earlier regarding the J_{index} of BBRv1, a similar behavior in J_{index} is observed for bandwidths (BWs) of 1 Gbps, 10 Gbps, and 25 Gbps.

HTCP's takeover. In the case of HTCP competing with CUBIC using FIFO, shown in Figure 2(k) – (o), shows that as the bottleneck buffer size increase, HTCP's throughput gradually decreases, with CUBIC's throughput increases. The fairness index in Figure 3(a) – (b) also shows this. With FIFO and larger queues, the buffer can become filled with a large number of packets, resulting in excessive queuing delays. Since FIFO treats packets from both CUBIC and HTCP flows equally, they experience similar queuing delays. CUBIC reacts to congestion signals mainly by reducing the CWND in response to packet drops. On the other hand, HTCP employs a different approach by estimating the available BW and adjusting the sending rate accordingly. When faced with increased queuing delays caused by bufferbloat, HTCP interprets these delays as an indication of limited available BW. Consequently, it reduces its sending rate, leading to a gradual decrease in throughput. As a result, HTCP gradually frees up a portion of the buffer that CUBIC occupies, as CUBIC does not rely on any estimation-based methods. Consequently, CUBIC gradually achieves better results. However, as observed in Figure 3(c) – (d), when HTCP operates alone, it can achieve equal throughputs for both senders.

Reno's takeover. Reno also exhibits a pattern when competing with CUBIC and using FIFO as the AQM algorithm. As shown in Figure 2(p) – (t), Reno initially achieves slightly lower throughput than CUBIC at small buffer sizes. However, it gradually loses its fair share, resulting in considerably worse throughput as the buffer sizes increase. This difference in performance can be attributed to their responses to packet losses. When Reno experiences a packet loss, it halves its CWND and then gradually increases it. On the other hand, CUBIC does not employ a fixed halving mechanism. Instead, CUBIC utilizes an adaptive multiplicative decrease approach, followed by a CUBIC function to increase the CWND. This allows CUBIC to occupy more space in the buffer compared to Reno. As the buffer size increases, CUBIC's occupancy in the buffer also increases, enabling it to achieve higher throughput with larger buffer sizes. We can also observe the consequential effects on the fairness index in Figure 3(a) – (b). In contrast, in scenarios where Reno operates alone (Figure 3(c) – (d)), it can ensure fair throughput for both senders.

5.2 Impact of different queuing algorithm

Figure 4 presents the throughput achieved by BBRv1, BBRv2, Reno and HTCP against CUBIC while using RED as the AQM. The throughput obtained when BBRv1 competes with CUBIC using RED, does not exhibit specific patterns. However, it does show a significant imbalance in the obtained throughput. A similar trend was observed for J_{index} (Figure 5(a) – (b)). In fact, RED was worse in fairness for BBRv1, with a J_{index} of just around 0.5. As seen in Figure 4(a) – (e), BBRv1 consumes almost all of the available BW, while CUBIC struggles to maintain a throughput of roughly below 10 Mbps in all the BW scenarios. The interaction between BBRv1 and RED's packet drop probability is the reason behind BBRv1 dominating the BW when RED is the AQM algorithm. RED drops packets randomly based on the queue length and waiting time at the queue, not just when congestion is detected [17]. The drop probability increases as the queue length increases. While this helps prevent bufferbloat and maintain low latency, it negatively impacts the performance of CUBIC. Since BBRv1 continues to send at the rate determined by its network model despite packet losses, it keeps filling up the available queue. RED observes the same arrival rate and waiting time at the queue, which prompts

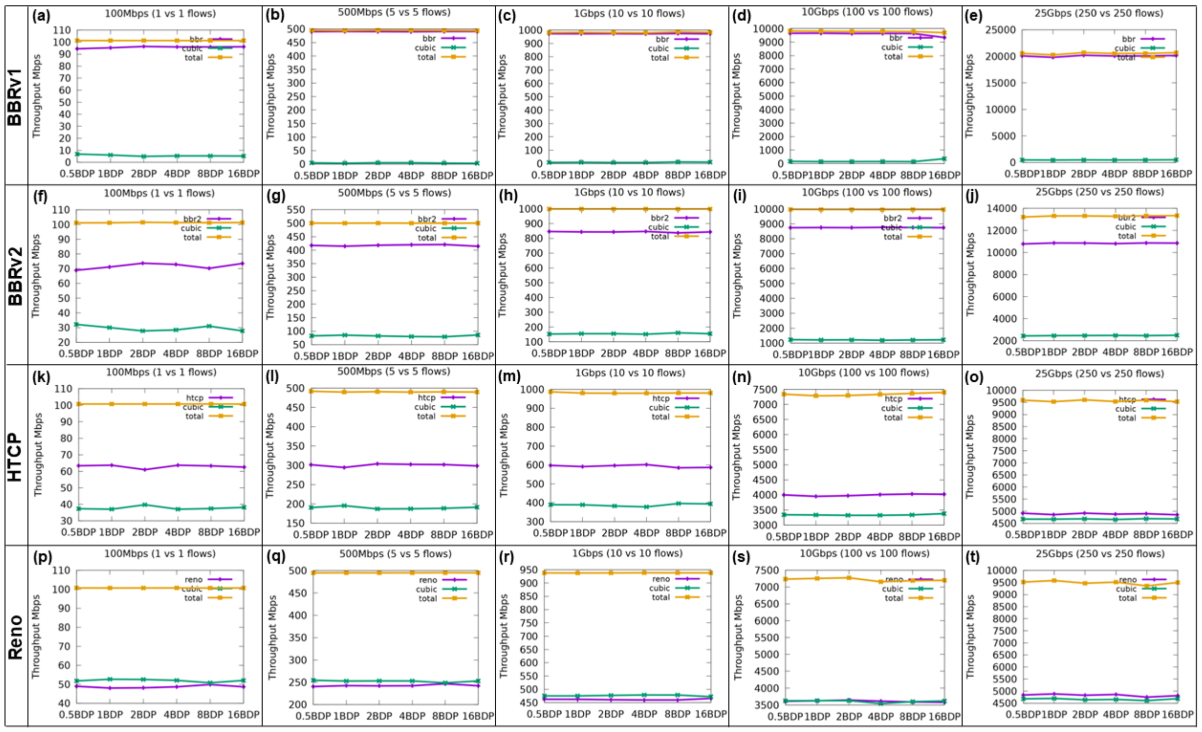

Figure 4: Per-sender throughput of TCP variants with Cubic, AQM=RED, (a) – (e) vs BBRv1, (f) – (j) vs BBRv2, (k) – (o) vs HTCP, and (p) – (t) vs Reno.

it to continue dropping packets at the same or even higher rate. Furthermore, RED does not maintain a separate queue or drop rate for each incoming flow. As a result, CUBIC is severely affected by the packet drops, forcing it to drastically reduce its sending rate. This leads to the pronounced imbalance in throughput. Even when BBRv1 operates alone, it also experiences unpredictable and drastic changes in throughput between the two senders, and we can observe its impact on J_{index} in Figure 5(c) – (d). In scenarios where BBRv1 operates alone, its rigid response to packet losses leads to RTOs, which force BBRv1 to significantly reduce its sending rate. This impact can be arbitrary, affecting either of the flows, resulting in intermittent changes in throughput.

Similar to BBRv1 with RED, BBRv2 consistently takes the majority of the BW when operating with RED, as shown in Figure 4(f) – (j). As explained in the previous section, RED does not solely rely on the available queue size but also maintains a drop rate based on the queue length. Using this rate, it randomly drops packets to reduce overall delay. We believe that this drop rate rarely exceeds the 2% threshold set by BBRv2 for reducing the inflight cap. Consequently, BBRv2 mostly discards small losses without reducing the sending rate. This ultimately worsens the situation for CUBIC, resulting in significantly lower throughput, as explained earlier in the previous section. Interestingly, in contrast to BBRv1, we observe that both senders achieve almost equal throughput with BBRv2 in the intra-CCA experiments, as we can observe the J_{index} in Figure 5(c) – (d). BBRv2 is more adaptable compared to BBRv1 and responds to packet losses by reducing its sending rate when the packet loss

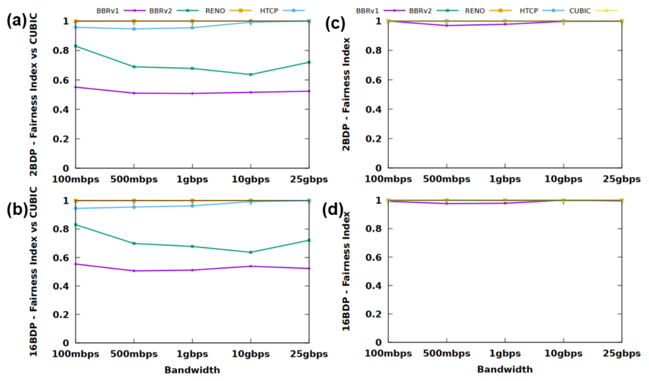

Figure 5: Observed Jain's fairness index for the considered CCAs when the AQM = RED: (a) – (b) inter-CCA, buffer size = 2 and 16 BDP, respectively, and (c) – (d) intra-CCA, buffer size = 2 and 16 BDP, respectively.

rate surpasses the 2% threshold. This approach helps BBRv2 avoid drastic changes in the sending rate and prevents RTOs, allowing it to ensure fair share of the BW for both senders.

Unlike FIFO, when HTCP operates with RED, and while competing with CUBIC, it achieves better throughput compared to CUBIC regardless of buffer sizes, as shown in Figure 4(k) – (o). FIFO lacks sophisticated logic and simply drops packets when the queue becomes full, while RED employs random packet drops based on the arrival

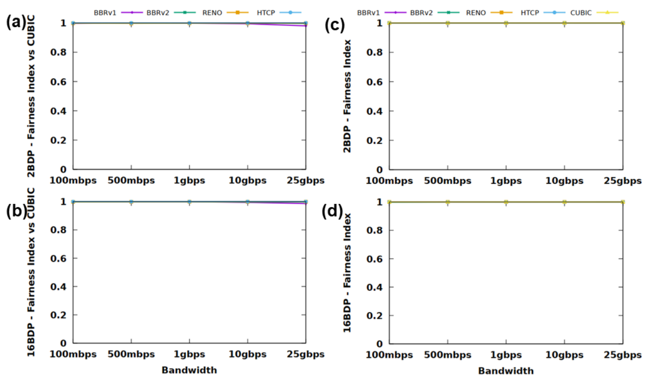

Figure 6: Observed Jain's fairness index for the considered CCAs when the AQM = FQ_CODEL: (a) – (b) inter-CCA, buffer size = 2 and 16 BDP, respectively, and (c) – (d) intra-CCA, buffer size = 2 and 16 BDP, respectively.

rate and waiting time at the queue, aiming to minimize queuing delay. This enables HTCP to better interpret the available BW and adapt its sending rate accordingly. As a result, HTCP consistently outperforms CUBIC, regardless of the impact of increased buffer sizes. However, we observe that the difference in obtained throughput between HTCP and CUBIC decreases as the BW increases. This can be attributed to the interplay between HTCP and RED. With higher BW, more packets are generally queued at the router, resulting in a higher arrival rate. This triggers RED to drop more packets in order to maintain low queuing delay. These increased packet losses have a negative impact on the performance of HTCP, leading to comparatively lower throughput as the BW increases. Additionally, as HTCP falls behind in occupying the queue, CUBIC takes advantage and achieves better throughput compared to scenarios with lower BW. However, in intra-CCA experiments, HTCP successfully ensures almost similar throughput for both senders, following J_{index} in Figure 5(c) – (d).

In the case of Reno with RED, it achieves a balanced throughput for both inter and intra-CCA experiments as shown in Figure 5. As can be observed from Figure 4(p) – (t), the issue of gradual decrease in throughput when competing with CUBIC, observed with FIFO as the AQM algorithm, is not present with RED. This is due to RED's reliance on the arrival rate and queue length, rather than waiting for the queue to become full. RED randomly drops packets based on these factors, providing Reno a better opportunity to achieve equal throughput with CUBIC and ensuring nearly equal performance.

FQ_CODEL demonstrates a different behavior compared to FIFO and RED [39]. By utilizing separate queues for each data flow, it ensures fair and equal treatment of all flows, preventing any single flow from dominating the network BW at the expense of others. This AQM algorithm effectively alters the behavior of the considered CCAs, leading to a more balanced performance where each CCA achieves nearly equal shares of the available BW, both in inter and intra-CCA experiments. The J_{index} index shown in Figure 6 supports this observation.

In the case of CUBIC, when examining its performance under intra-CCA experiments in terms of throughput, it consistently achieves almost equal throughput for both senders across all the AQM algorithms. Considering both intra and inter-CCA results, CUBIC is robust and can adapt well to different AQM algorithms.

5.3 Overall Link Utilization

In this section, we aim to observe the total utilization of available BW by the combination of CCAs and AQM algorithms. We define link utilization (φ) as the normalized total throughput, calculated as follows:

$$\varphi = \frac{\sum_{i=1}^{n} T_i}{\beta_\tau} \tag{3}$$

Here, β_τ represents the total available BW at the bottleneck for a given scenario, and T_i represents the obtained throughput of flow i. The variable n denotes the total number of flows passing through the bottleneck. A value of φ equal to 1 indicates full link utilization, while lower values indicate poorer link utilization. Figure 7 illustrates the observed φ for FIFO, RED, and FQ_CODEL, considering buffer sizes of 2 and 16 BDP.

Interestingly, we found that link utilization primarily depends on the choice of the AQM algorithm. All the considered CCAs were able to achieve nearly full link utilization when FIFO was used as the AQM. Similarly, for FQ_CODEL, the considered CCAs achieved almost full link utilization except for the 25 Gbps BW scenario. However, with RED, the considered CCAs exhibited significant lags in link utilization, especially in scenarios with a BW greater than or equal to 1 Gbps.

FIFO does not consider packet arrival rates or waiting times to manage the congestion. Instead, it simply starts dropping packets when the queue becomes full. This allows CCAs to take full control and fill up the queue according to their needs, enabling them to fully utilize the available capacity.

In contrast, FQ_CODEL and RED employ a technique to control the queue length by dropping packets before the queue becomes full. This preemptive packet dropping gives significant control to AQM algorithms over CCAs, enabling AQMs to actively reduce bufferbloat and improve overall network performance by avoiding congestion. The decision on when to drop packets is highly influenced by the packet arrival rate. As the BW increases, the data arrival rate at the router also rises. It seems that the high rate of data arrival overloads the FQ_CODEL and RED AQM algorithms, causing them to drop packets earlier than in optimal buffer use cases. This contributes to their lower link utilization in high-BW scenarios.

Moreover, RED is too aggressive in maintaining small queuing delays and exhibits high dependency on the arrival rate. RED determines whether to drop packets based on the average queue length and configurable minimum and maximum thresholds. The arrival rate directly influences the average queue length, as it determines the speed at which packets arrive at the router. If the arrival rate is high, the queue length can rapidly increase, potentially causing RED to drop packets more frequently as a response to interpreted congestion. This forms the primary reason for the significantly poor link utilization when RED is used as the AQM algorithm.

Supporting our assumption, we can observe that only BBRv1 was able to achieve a throughput above 20 Gbps for the 25 Gbps BW

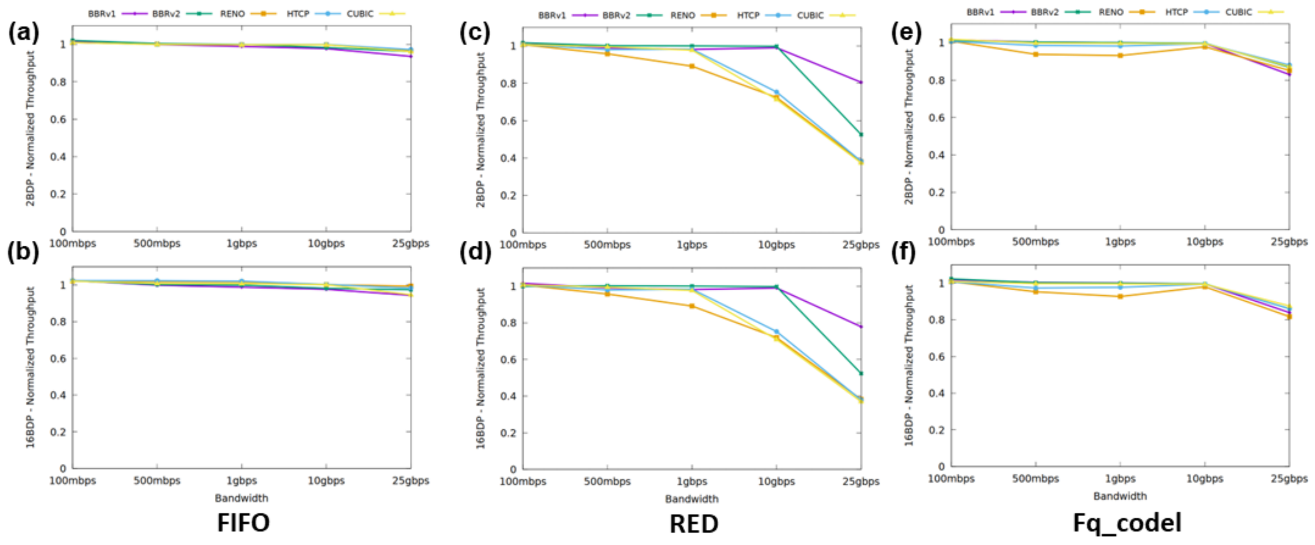

Figure 7: During intra-CCA experiments, observed overall link utilization for: (a) – (b) FIFO, (c) – (d) RED, and (e) – (f) FQ_CODEL.

scenario with RED. This is because BBRv1 does not directly respond to packet losses; it continues to send at the same rate despite losses. However, when there are RTOs, BBRv1 backs off and starts sending at a much lower rate. The RTOs are the main hindrance preventing BBRv1 from reaching full capacity throughput, even with RED and FQ_CODEL AQM algorithms. As for the other considered CCAs that respond to packet losses, they were unable to achieve the same throughput as BBRv1.

Finally, the failure of the RED and FQ_CODEL AQM algorithms does not necessarily reflect their inability to utilize high-BW networks. Rather, this failure is mainly attributed to their internal parameters. For high-BW scenarios, these internal parameters need to be properly optimized to handle such situations effectively. This opens up significant opportunities for future research on optimizing these algorithms to operate in a wide range of BW scenarios, especially considering future Internet.

5.4 Retransmissions

We observe a strong correlation between the number of retransmissions and the algorithm configurations. The performance of the CCAs in terms of number of retransmissions under 2 BDP and 16 BDP buffer sizes for FIFO, RED, and FQ_CODEL are shown in Figure 8. For both RED and FQ_CODEL, the number of retransmissions increases with an increase in BW. However, an increase in BDP does not show any significant impact on the retransmissions. Both RED and FQ_CODEL employ their own methods to keep the queuing delay low by dropping packets before the buffer becomes full, which helps eliminate buffer size dependencies. Moreover, RED exhibits comparatively fewer retransmissions than FQ_CODEL, due to its significant failure to utilize the available BW, as observed earlier. Additionally, with an increase in BW, we observe a scalable increase in retransmissions for each CCAs in both FQ_CODEL and RED. With an increase in BW, the rate of packet arrival also increases. As both RED and FQ_CODEL rely on arrival rate for dropping packets, their retransmissions increase with the BW increase.

Unlike RED and FQ_CODEL, FIFO waits until the buffer becomes full to drop a packet. And as the buffer size increases, the probability to drop packets decreases. Here we observe a direct impact of buffer sizes on packet retransmissions; the retransmissions for all CCAs decrease with increase in buffer sizes. Apart from that, although we observe a similar trend of increased packet retransmissions with the BW increase for FIFO, intermittent low retransmissions are also observed. Even with updated BDP as BW increases, the ratio of available buffer remains the same regardless of BW. On the other hand, with increased BW, the arrival rate also increases, meaning the queue needs to handle more packets simultaneously, leading to quicker buffer filling. As a result, with the increase in BW, we observe an increase in retransmissions, especially for low BDP buffer sizes. We also observed intermittent low retransmissions, due to the large available buffer sizes and FIFO's dependency on queue length. For example, for FIFO in 16 BDP buffer sizes, we can observe that the retransmissions for HTCP, CUBIC, and Reno remains almost in the same range. We also observe significantly low intermittent retransmissions for BBRv1 and BBRv2, which mainly correspond to their 2 BDP inflight cap, restricting them from occupying the entire buffer and resulting in low retransmissions.

Finally, we observe a significantly higher number of packet retransmissions with BBRv1 in all cases, which is attributed to its rigid response to packet losses. BBRv2 generates significantly lower retransmissions than BBRv1 because it responds to packet losses only when the loss rate exceeds a 2% threshold. However, it still produces the second-highest number of packet retransmissions among the considered CCAs. The third highest is HTCP, due to its more aggressive behavior compared to Reno or CUBIC. Reno and CUBIC exhibit almost equal packet retransmissions due to their loss-based nature.

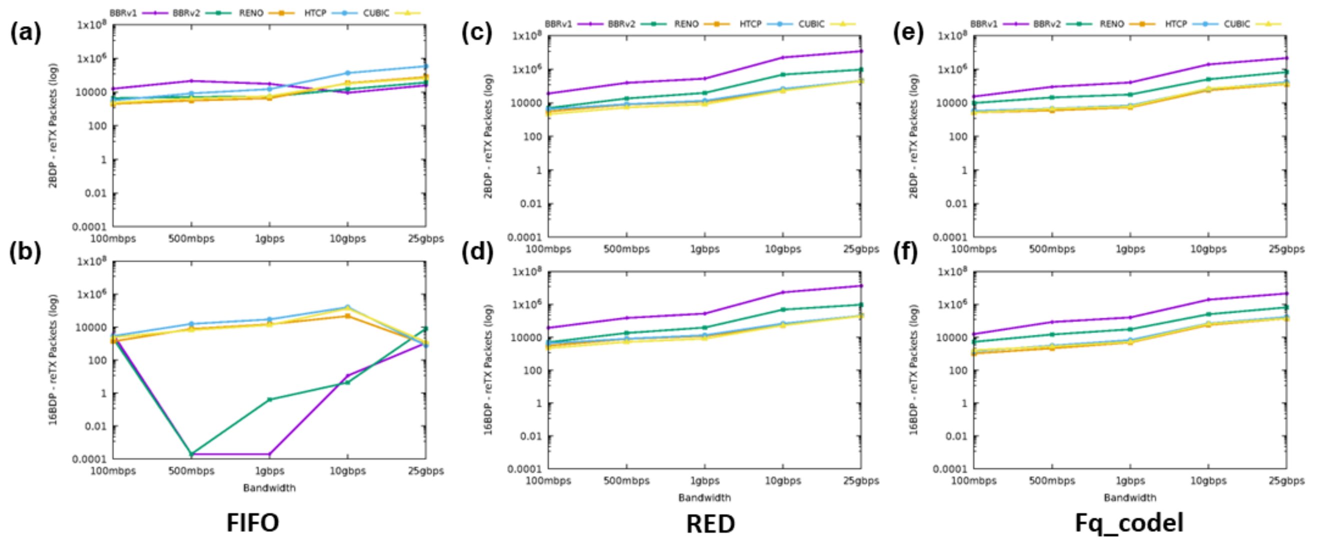

Figure 8: During intra-CCA experiments, observed number of retransmissions for: (a) – (b) FIFO, (c) – (d) RED, and (e) – (f) FQ_CODEL.

5.5 Overall Performance Comparison

In this section, our objective is to analyze the overall performance of the considered CCAs and AQM algorithms for the designated scenarios. To evaluate the overall performance in terms of retransmissions, we begin by analyzing the retransmission data for each scenario, comparing CUBIC vs CUBIC as a standard reference ($R_{cubic\_vs\_cubic}$). Subsequently, we normalize the observed retransmission data for each CCA with respect to $R_{cubic\_vs\_cubic}$, yielding the relative retransmission value (RR) using the formula:

$$RR = \frac{R_{cca1\_vs\_cca2}}{R_{cubic\_vs\_cubic}} \tag{4}$$

For each run, we compute the RR value, calculate the average for each condition under the same CCA and AQM algorithms, and then derive an overall average value - $Avg(RR)$, encompassing all conditions within each combination of CCA and AQM. Similarly, we calculate the average φ and J_{index} across all five runs for each observation scenario, and then determine an overall average among all the BDPs for each combination of CCA and AQM algorithms. We refer to these as the average overall link utilization - $Avg(\varphi)$, and the average fairness - $Avg(J_{index})$, respectively. The resulting average outcomes are presented in Table 3.

Table 3 provides a comprehensive overview of the overall performances of the considered CCA and AQM algorithms in our experiments. BBRv1 results in a significant number of packet retransmissions without providing benefits over other CCAs. Consequently, BBRv1 consumes a considerable portion of network resources. Reno exhibits poor performance both while operating alone and competing with CUBIC. While CUBIC performs well independently, it struggles to maintain performance when in competition with other CCAs, as observed in previous sections. Both HTCP and BBRv2 deliver competitive results. However, BBRv2 slightly outperforms HTCP in terms of average overall link utilization at the cost of increased packet retransmissions. In summary, if a reasonable overall performance is acceptable with minimal network resource consumption, HTCP can be a viable choice. On the other hand, when seeking the highest overall performance and the network can tolerate slightly more retransmissions, BBRv2 emerges as a promising solution, particularly when paired with FQ_CODEL.

Finally, as previously mentioned, we have maintained a fixed RTT for these experiments to simplify results presentation. We believe that our qualitative observations can be replicated using different RTTs. Furthermore, we plan to undertake further investigations involving varying RTTs in our future work.

6 CONCLUSION

In this study, we evaluated the performance of different TCP CCAs and AQM algorithms in real-world networks using the FABRIC testbed. We conducted tests across a wide range of scenarios, including various BWs ranging from 100 Mbps to 25 Gbps and buffer sizes ranging from 0.5 times the BDP to 16 times the BDP. Our evaluation encompassed several CCAs such as BBRv1, BBRv2, HTCP, Reno, and CUBIC, as well as AQM algorithms including FIFO, FQ_CODEL, and RED. We aimed to analyze the interactions between different combinations of TCP CCAs and AQM algorithms, and all our findings are reproducible and publicly accessible.

Our results revealed that both TCP CCAs and AQM algorithms significantly impact overall network performance, particularly in terms of throughput, link utilization, and fairness. The selection of an appropriate combination of TCP CCAs and AQM algorithms can have a substantial influence on performance outcomes. FQ_CODEL consistently demonstrated superior fairness among flows, but it fell short in fully utilizing the capacity of the 25 Gbps link. RED showed

Table 3: Overall performance comparison of the considered *CCA* and *AQM* algorithms in terms of $Avg(\varphi)$, $Avg(RR)$, and $Avg(J_{index})$.

$CCA1$ vs $CCA2$	AQM	$Avg(\varphi)$	$Avg(RR)$	$Avg(J_{index})$
BBRv1 vs BBRv1	FIFO	0.986	23.164	0.995
BBRv1 vs CUBIC		0.997	14.916	0.803
BBRv2 vs BBRv2		0.995	1.141	0.98
BBRv2 vs CUBIC		0.998	1.823	0.934
HTCP vs HTCP		0.999	2.493	1.0
HTCP vs CUBIC		0.997	1.624	0.971
Reno vs Reno		0.997	1.235	0.994
Reno vs CUBIC		0.998	1.01	0.847
CUBIC vs CUBIC		0.995	1.0	0.997
BBRv1 vs BBRv1	RED	0.938	47.687	0.938
BBRv1 vs CUBIC		0.94	41.056	0.522
BBRv2 vs BBRv2		0.903	4.872	0.999
BBRv2 vs CUBIC		0.901	3.675	0.722
HTCP vs HTCP		0.794	1.497	0.999
HTCP vs CUBIC		0.796	1.272	0.979
Reno vs Reno		0.738	1.281	1.0
Reno vs CUBIC		0.766	1.136	1.0
CUBIC vs CUBIC		0.788	1.0	1.0
BBRv1 vs BBRv1	FQ_CODEL	0.971	24.468	1.0
BBRv1 vs CUBIC		0.97	13.986	0.994
BBRv2 vs BBRv2		0.977	4.386	1.0
BBRv2 vs CUBIC		0.975	2.312	0.998
HTCP vs HTCP		0.969	1.135	1.0
HTCP vs CUBIC		0.972	1.057	1.0
Reno vs Reno		0.94	0.852	1.0
Reno vs CUBIC		0.96	0.891	0.998
CUBIC vs CUBIC		0.974	1.0	1.0

the worst overall fairness performance, and it exhibited significant limitations in efficiently utilizing high-BW paths, especially from 1 Gbps and beyond. FIFO exhibited interesting patterns of results for different CCAs regarding fairness, and notably, it was the only successful AQM algorithm that enabled all CCAs to utilize the full capacity of high-BW links. Considering the results, BBRv2 with FQ_CODEL emerged as a promising combination in terms of throughput, fairness, and retransmission. However, further research is necessary to improve FQ_CODEL's capacity utilization in high-BW links. Furthermore, the data we have shared can serve as a valuable starting point for the development of ML-based CCAs to optimize network usage.

In future work, we intend to expand our experiments to include more high-BW scenarios, observe performance under network anomalies (e.g. variable rates of packet loss), and RTTs. Additionally, we plan to capture detailed router logs to gain a clearer understanding of internal parameters and their impact on performance.

ACKNOWLEDGMENTS

This work is funded by the Department of Energy under the Integrated Computational and Data Infrastructure (ICDI) for Scientific Discovery, grant #DE-SC0022328. Experimental data was collected on the FABRIC testbed supported by National Science Foundation.

REFERENCES

[1] Richelle Adams. 2012. Active queue management: A survey. *IEEE communications surveys & tutorials* 15, 3 (2012), 1425–1476.

[2] Mark Allman, Vern Paxson, and Ethan Blanton. 2009. RFC 5681: TCP congestion control.

[3] Subramanian Balaji, Karan Nathani, and Rathnasamy Santhakumar. 2019. IoT technology, applications and challenges: a contemporary survey. *Wireless personal communications* 108 (2019), 363–388.

[4] Ilya Baldin, Anita Nikolich, James Griffioen, Indermohan Inder S. Monga, Kuang-Ching Wang, Tom Lehman, Paul Ruth, and Ewa Deelman. 2019. FABRIC: A National-Scale Programmable Experimental Network Infrastructure. *IEEE Internet Computing* 23, 6 (nov 2019), 38–47. https://doi.org/10.1109/MIC.2019.2958545

[5] Sumitha Bhandarkar, Saurabh Jain, and AL Narasimha Reddy. 2006. LTCP: improving the performance of TCP in highspeed networks. *ACM SIGCOMM Computer Communication Review* 36, 1 (2006), 41–50.

[6] Yi Cao, Arpit Jain, Kriti Sharma, Aruna Balasubramanian, and Anshul Gandhi. 2019. When to use and when not to use BBR: An empirical analysis and evaluation study. In *Proceedings of the Internet Measurement Conference.* 130–136.

[7] Neal Cardwell, Yuchung Cheng, C Stephen Gunn, Soheil Hassas Yeganeh, and Van Jacobson. 2017. BBR: congestion-based congestion control. *Commun. ACM* 60, 2 (2017), 58–66.

[8] Neal Cardwell, Yuchung Cheng, Soheil Hassas Yeganeh, Priyaranjan Jha, Yousuk Seung, Kevin Yang, Ian Swett, Victor Vasiliev, Bin Wu, Luke Hsiao, et al. 2019. BBRv2: A model-based congestion control performance optimization. In *Proc. IETF 106th Meeting.* 1–32.

[9] Eli Dart, Lauren Rotman, Brian Tierney, Mary Hester, and Jason Zurawski. 2013. The Science DMZ: A network design pattern for data-intensive science. In *SC '13: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis.* 1–10. https://doi.org/10.1145/2503210.2503245

[10] Amogh Dhamdhere and Constantine Dovrolis. 2006. Open issues in router buffer sizing. *ACM SIGCOMM Computer Communication Review* 36, 1 (2006), 87–92.

[11] Nandita Dukkipati and Nick McKeown. 2006. Why Flow-Completion Time is the Right Metric for Congestion Control. *SIGCOMM Comput. Commun. Rev.* 36, 1 (jan 2006), 59–62. https://doi.org/10.1145/1111322.1111336

[12] Mulalo Dzivhani and Khmaies Ouahada. 2019. Performance evaluation of TCP congestion control algorithms for wired networks using NS-3 simulator. In *2019 IEEE AFRICON.* IEEE, 1–7.

[13] W-C Feng, Dilip D Kandlur, Debanjan Saha, and Kang G Shin. 1999. A self-configuring RED gateway. In *IEEE INFOCOM'99. Conference on Computer Communications. Proceedings. Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies. The Future is Now (Cat. No. 99CH36320)*, Vol. 3. IEEE, 1320–1328.

[14] Niroshinie Fernando, Seng W Loke, and Wenny Rahayu. 2013. Mobile cloud computing: A survey. *Future generation computer systems* 29, 1 (2013), 84–106.

[15] Victor Firoiu and Marty Borden. 2000. A study of active queue management for congestion control. In *Proceedings IEEE INFOCOM 2000. Conference on Computer Communications. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies (Cat. No. 00CH37064)*, Vol. 3. IEEE, 1435–1444.

[16] Sally Floyd. 1994. TCP and explicit congestion notification. *ACM SIGCOMM Computer Communication Review* 24, 5 (1994), 8–23.

[17] Sally Floyd and Van Jacobson. 1993. Random early detection gateways for congestion avoidance. *IEEE/ACM Transactions on networking* 1, 4 (1993), 397–413.

[18] Sally Floyd and Van Jacobson. 1995. Link-sharing and resource management models for packet networks. *IEEE/ACM transactions on Networking* 3, 4 (1995), 365–386.

[19] Google. 2022. Linux kernel with BBRv2 support. https://github.com/google/bbr/tree/v2alpha.

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

[21] Mario Hock, Roland Bless, and Martina Zitterbart. 2017. Experimental evaluation of BBR congestion control. In *2017 IEEE 25th international conference on network protocols (ICNP).* IEEE, 1–10.

[22] Mario Hock, Maxime Veit, Felix Neumeister, Roland Bless, and Martina Zitterbart. 2019. Tcp at 100 gbit/s–tuning, limitations, congestion control. In *2019 IEEE 44th Conference on Local Computer Networks (LCN).* IEEE, 1–9.

[23] Sami Iren, Paul D Amer, and Phillip T Conrad. 1999. The transport layer: tutorial and survey. *ACM Computing Surveys (CSUR)* 31, 4 (1999), 360–404.

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

[25] Raj Jain, Arjan Durresi, and Gojko Babic. 1999. Throughput fairness index: An explanation. In *ATM Forum contribution*, Vol. 99.

[26] Rajendra K Jain, Dah-Ming W Chiu, William R Hawe, et al. 1984. A quantitative measure of fairness and discrimination. *Eastern Research Laboratory, Digital Equipment Corporation, Hudson, MA* 21 (1984).

[27] Douglas Leith and Robert Shorten. 2004. H-TCP: TCP for high-speed and long-distance networks. In *Proceedings of PFLDnet*, Vol. 2004. Citeseer.

[28] Douglas Leith, R Shorten, and Y Lee. 2005. H-TCP: A framework for congestion control in high-speed and long-distance networks. In *PFLDnet Workshop*.

[29] Josip Lorincz, Zvonimir Klarin, and Julije Ožegović. 2021. A comprehensive overview of TCP congestion control in 5G networks: Research challenges and future perspectives. *Sensors* 21, 13 (2021), 4510.

[30] SÃ¡ndor MolnÃ¡r, BalÃ¡zs Sonkoly, and Tuan Anh Trinh. 2009. A comprehensive TCP fairness analysis in high speed networks. *Computer Communications* 32, 13 (2009), 1460–1484. https://doi.org/10.1016/j.comcom.2009.05.003

[31] Kathleen Nichols and Van Jacobson. 2012. Controlling queue delay. *Commun. ACM* 55, 7 (2012), 42–50.

[32] Rong Pan and Balaji Prabhakar. 2001. *CHOKe, a Simple Approach for Providing Quality of Service Through Stateless Approximation of Fair Queueing*. Stanford University.

[33] Adithya Abraham Philip, Ranysha Ware, Rukshani Athapathu, Justine Sherry, and Vyas Sekar. 2021. Revisiting TCP Congestion Control Throughput Models & Fairness Properties at Scale. In *Proceedings of the 21st ACM Internet Measurement Conference*. New York, NY, USA, 96–103. https://doi.org/10.1145/3487552.3487834

[34] PoSeiDon Workflows. 2023. GitHub Repository. https://github.com/PoSeiDon-Workflows/tcp-conflict-study. Accessed Aug 18, 2023.

[35] K Ramakrishnan and Sally Floyd. 1999. RFC2481: A Proposal to add Explicit Congestion Notification (ECN) to IP.

[36] Kun Tan Jingmin Song, Qian Zhang, and Murari Sridharan. 2006. Compound TCP: A scalable and TCP-friendly congestion control for high-speed networks. *Proceedings of PFLDnet 2006* (2006).

[37] Yeong-Jun Song, Geon-Hwan Kim, Imtiaz Mahmud, Won-Kyeong Seo, and You-Ze Cho. 2021. Understanding of bbrv2: Evaluation and comparison with bbrv1 congestion control algorithm. *IEEE Access* 9 (2021), 37131–37145.

[38] Milan P Stanic. 2001. Tc–traffic control. *Linux QOS Control Tool* (2001).

[39] Dave Taht, Jim Gettys, T Hoeiland-Joergensen, Toke Hoeiland-Joergensen, Eric Dumazet, J Gettys, P McKenney, E Dumazet, and P McKenney. 2018. The flow queue codel packet scheduler and active queue management algorithm.

[40] Brian Tierney, Eli Dart, Ezra Kissel, and Eashan Adhikarla. 2021. Exploring the BBRv2 Congestion Control Algorithm for use on Data Transfer Nodes. In *2021 IEEE Workshop on Innovating the Network for Data-Intensive Science (INDIS)*. 23–33. https://doi.org/10.1109/INDIS54524.2021.00008

[41] Belma Turkovic, Fernando A Kuipers, and Steve Uhlig. 2019. Interactions between congestion control algorithms. In *2019 Network Traffic Measurement and Analysis Conference (TMA)*. IEEE, 161–168.

[42] Jingyuan Wang, Jiangtao Wen, Jun Zhang, Zhang Xiong, and Yuxing Han. 2016. TCP-FIT: An improved TCP algorithm for heterogeneous networks. *Journal of Network and Computer Applications* 71 (2016), 167–180.