Received: 19 December 2019 | Revised: 10 September 2020 | Accepted: 28 September 2020

IET Communications

DOI: 10.1049/cmu2.12061

The Institution of Engineering and Technology WILEY

ORIGINAL RESEARCH PAPER

Study on performance of AQM schemes over TCP variants in different network environments

Salman Muhammad[1] | **Touseef Javed Chaudhery**[2] | **Youngtae Noh**[1]

[1] Department of Electrical and Computer Engineering, Inha University, South Korea

[2] Department of Computer Science, Air University, Pakistan

Correspondence
Youngtae Noh, Department of Electrical and Computer Engineering, Inha University, 100 Inharo, Nam-gu, 22212, Republic of Korea. Email: ytnoh@inha.ac.kr

*Study on Performance of AQM schemes over TCP Variants in Different Network Environments

Funding information
Inha University

Abstract

Increasing the size of memory in network devices leads to the problem of a persistently full buffer (a.k.a, bufferbloat). The objective of this study is to compare the recently introduced Controlled Delay (CoDel) scheme with the traditional method of active queue management, such as Random Early Detection (RED) algorithms over TCP variants. To explore the potential of CoDel over RED, TCP variants have been assessed at three settings: variable congestion and fixed payload (VCFP), variable payload and fixed congestion (VPFC), and high congestion and high payload (HCHP). We assessed the CoDel and RED schemes for active queue management (AQM) using three performance metrics: link utilization, drop rate, and queuing delay. The analytical results show that CoDel outperformed RED in most aspects over variants of TCP because of its auto-tuning and auto-adjustment features. However, RED outperformed CoDel in a few cases. In the VCFP setting, RED recorded a lower drop rate overall TCP variants. Moreover, in the VPFC setting, RED with a payload of 500–1000 bytes performed better in terms of drop rate. Finally, in the HPHC setting, there were two cases where RED, over TCP NewReno and Vegas, performed well in terms of drop rate.

1 | INTRODUCTION

The progressive development of computer applications has continued at pace in the last few years. Applications available nowadays have more features embedded into them than predecessors. Approximately 90% of Internet traffic is estimated to be serviced by the Transmission Control Protocol (TCP) [1]. Moreover, the use of social media, streaming, and files transfer services has witnessed remarkable spikes that have elevated the chances of Internet congestion. Vendors in various sectors of the Internet are attempting to place excess buffers in network devices to prevent packet drop and increase link utilisation [2]. However, the large buffers in network devices conflict with the nature of the TCP. From a technical aspect, the TCP increases its transmission rate continuously until the buffer is filled. As soon as a packet drop is sensed (which is regarded as a signal of congestion), the TCP cuts its sending rate. Consequently, large buffers, primarily designed to prevent packet drops, create excessive queuing latency.

The queuing latency caused by large buffers is referred to as bufferbloat [3], and it is critical for frequently used TCP-based applications, such as the file transfer protocol (e.g. Dropbox, Google Drive) and hypertext transfer protocol (HTTP) etc., because large queues lead to long round-trip times (RTTs) and lower throughput. Increasing the memory size in network devices poses a variety of challenges—for instance, the large buffers that cable and ADSL Internet service-providers (ISPs) use to shape network traffic[1] create long queues that can delay packets for several hundred milliseconds [4]. In the network devices the use of large buffer leads to increase in user-perceived latency [5, 6], and [7]. The delays experienced due to buffers in 3G/4G networks have also been reported in [8]. Similarly, the existing cellular-based networks uses large buffers at the base stations to tackle the bursty traffic. However, such buffers leads to a problem known as long flow completion time, which affects the Quality of Experience (QoE) and has been revealed in [9]. There is no prior communication infrastructure in multi-hop

[1] To cope with the required traffic profile, either all or parts of the datagrams are delayed. Traffic shaping is used for bandwidth management, and is common in packet-switched networks.

wireless networks, and the information is exchanged via a chain of relay nodes. The buffers in the relay nodes lead to queuing latency which degrades the overall performance [10]. The bufferbloat problem is a matter of concern in Wireless LAN too as stated in [11, 12] and [13].

To eliminate bufferbloat, we cannot completely omit the use of buffers in network devices because this will lead to a drastic increase in the number of dropped packets even with minor congestion, and thus the link will suffer from underutilisation. Rather than buffer management, a better approach is to manage the queues actively (i.e. by discriminating bad queues from the good ones). Active queue management (AQM) is an effective way to mitigate bufferbloat. Although bufferbloat and problems generated by it have been known for over three decades [19], AQM has not been deployed for long time owing to the complexity of parameter settings over changing network links.

Fortunately there are several ways to combat with bufferbloat problem. BBR TCP [14] is a transport layer solution proposed by Google LLC (Google) which measures the delivery rate and round-trip time of a connection. Based on this measurement it creates a model with the recent maximum bandwidth and minimum round-trip delay. BBR leverages this model to maximise the amount of data it can allow in flight anytime in the network. Another recently introduced transport layer solution is C2TCP [20]. C2TCP runs over the top of conventional throughput-based TCP. This protocol works flexibly with satisfying the strict delay requirements for variety of applications whilst maintaining the maximum possible throughput. C2TCP meets various target delays regardless of any needful for profiling the network state, channel prediction, and sophisticated rate adjustment. There are numerous solutions available at the network layer such as, RED [21], CoDel [22], PIE [15] ,BLUE [38] and COBALT [35] etc. The Random Early Detection (RED) scheme is an early AQM scheme. The RED algorithm sets its probability of packet drop based on upper and lower bounds to maintain an average queue length. If the average queue length is smaller than the lower threshold, the incoming packets are enqueued without being dropped. On the contrary, if the average queue length starts to exceed the upper threshold, the incoming packet is randomly marked (or dropped), which ultimately sends a congestion signal to the transport layer, which reduces the throughput. As long as the average queue length is in between the upper and lower thresholds, each arriving packet is marked with a probability p, where p is a function of average queue length. If p is large enough, the packet is dropped, and is otherwise enqueued.

As mentioned above, RED was introduced in the 1990s, but its implementation has been challenging due to the complexity of its configuration over dynamic network links. The first weakness of RED is the fluctuation of its average queue length with the different level of congestion as reported in [16, 17], and [18]. That is, at lower congested link, the average queue length is near the lower threshold; and at higher congestion, the average queue length is greater than or equal to upper threshold. Hence the average queuing delay is not predictable beforehand. Another weakness of RED is related to the throughput, which is sensitive to the parameter settings and load on traffic. In other words, the performance of RED deteriorate when the average queue length exceeds the upper threshold, consequently causing an increased drop rate and decreased throughput. In the literature there are several variants of RED available such as Adaptive–RED (ARED) [64], Learning–Automata–Like–RED (LALRED) [23], RED–Exponential (RED_E) [24], Nonlinear RED (NRED) [25], and Effective–RED (ERED) [26]. However, all the variants have similar functionality and senses the congestion based on the queue length.

To overcome the RED algorithm's drawbacks, some auto-tuned AQMs were recently introduced, of which the Controlled Delay scheme (CoDel) is among the most robust. The author have claimed that CoDel is parameterless, controls delay regardless of link rates, traffic loads, and round-trip delays, and adapts to changing link rates [27]. CoDel estimates the sojourn time[2] of each packet and compares it with a certain defined threshold. If the sojourn time is greater than the threshold, the dropping flag switches to high. If the dropping flag remains high for a pre-defined interval, the packet is dropped and the interval is updated according to the control law defined in CoDel; otherwise, the dropping flag is reset and the packet is forwarded to the network. When a packet is dropped by CoDel, the TCP senses the packet drop and adjusts its congestion window.

In any AQM, the common objective is to sense congestion either in the form of queue length (e.g. RED) or queue delay (e.g. CoDel) and inform the TCP of it. The TCP in turn adjusts its congestion window, and the sending rate is thus reduced. There are different types of TCPs, where each has a specific purpose. In addition, each TCP has a different approach to controlling its congestion window. An AQM is highly influenced by the choice of TCP used and the network environment provided. In this study, the influence of three parameters on the performance of the AQM over variants of TCP is investigated: drop rate, link utilisation, and the delay experienced by every packet from ingress to egress. We use three different scenarios to conduct a detailed evaluation of CoDel with varying levels of congestion and changing payload sizes over various TCP variants. The scenarios are (i) variable congestion and fixed payload size (VCFP), (ii) variable payload size and fixed congestion (VPFC), and (iii) high congestion and high payload size (HCHP). The traditional RED AQM is used as a benchmark to compare with CoDel.

The remainder of this paper is structured as follows. Section 2 gives some related work on congestion control using AQM schemes. Section 3 presents our contribution. Section 4 details the network topology, and provides a brief review of each TCP variant and the applied AQM. Section 5 describes the evaluation, including the simulation setup, performance metrics, and a discussion of the results. Section 6 concludes this paper.

2 | RELATED WORK

Augustu et al. [28] has worked on the cross comparison between TCP and AQM algorithms. They have considered two congestion levels (i.e. 16 and 64 flows) to test the capabilities of several TCPs over AQM variants. The simulation is based on evaluating

[2] The time taken by a packet from the enqueuing to the dequeuing process

the TCP performance metric such as goodput and RTT and a network metric, that is, queue occupancy. However this paper lacks other dynamic changes in simulation environment, such as variable packet size which is very common in mixed internet flows. They have not investigated the actual packet sojourn time in network layer, which precisely depicts the packet delay in a bufferbloat. lastly, they claims that at high congestion the general performance trend of different AQMs does not change significantly when moving from a TCP variant to another. However this is not correct and we have proven it in our result section (Figure 20).

In another branch of study [29] a testbed experiment is conducted on some AQM schemes such as CoDel, PIE and Adaptive-RED using various congestion levels (4, 16 and 64 flows) subject to change in target delay (from 1ms to 30ms). CoDel shows stable behaviour in terms of RTT and goodput compared to the other two AQMs, however the assessment is limited to TCP metrics and have not been evaluated for the network layer. Similarly another testbed experiment in [33] evaluates the TCP metrics (RTT and throughput) for several AQMs including CoDel and PIE etc. The main focus of their work is to use different buffer sizes and mitigate the bufferbloat problem by leveraging CoDel algorithms and byte queue limit (BQL) solution. Grazia1 et al. [34] have surveyed a cross-comparison of popular TCP and AQM variants. They have investigated the TCP performance in terms of goodput, RTT, and fairness. However, they did not consider any of the network layer performance metrics.

In [30] the load transient taking place at the edge of network has been investigated over AQM schemes including CoDel, PIE and an aggressive version of RED known as HRED [31]. The authors considered variable congestion flows to assess the queuing delay observed at the bottleneck routers, however, this study lack the detail evaluation of other metric such as link utilisation and drop rate etc.

The authors in [32] have conducted a testbed experiment and evaluated the AQMs in terms of packet drop, latency and throughput. Nonetheless, their result evaluation is very shallow and they have not mapped the TCPs function with each AQM.

Ye et al. [39] has applied some tweak to the existing CoDel scheme (a.k.a. ACoDel-IT) by introducing adoptive tuning for interval and adoptive tuning for target and interval (ACoDel-TIT). They tested the aforementioned algorithms in 5 scenarios with different congestion level and link capacities. The author assessed several performance metric such as queuing delay, packet drop and link utilisation over TCP NewReno. However ACoDel-IT and ACoDel-TIT have not been evaluated over different TCPs. Similarly COBALT [35] (combination of CoDel and Blue) is another recently introduced AQM and has been tested in different traffic scenarios, such as light TCP traffic (5 TCP flows), heavy TCP traffic (50 TCP flows) and mix TCP and UDP traffic (5 TCP and 2 UDP). The performance is compared with CoDel in terms of queue delay and queue occupancy. Yet it is not examined over different TCP variants to further expose its behaviour.

Kennedy et al. [36] have demonstrated the AQMs robustness in some dynamic settings like varying link capacities, propagation delay, and varying load in a wireless environment. They have evaluated four AQMs, namely RED, Fixed-Parameter Proportional Integral (PI), Model Predictive Control (MPC), and the Self-Tuning Regulator (STR). Their assessment is solely related to the queue length and drop probabilities at the network layer.

The study in [37] has considered three types of TCPs, namely TCP Illinois, TCP Westwood, and TCP Vegas, and two AQMs, that is, CoDel and DropTail. They have investigated the RTT, throughput, and fairness for all the TCPs by varying the target delays of the CoDel.

Table 1 summarises the aforementioned works on the bufferbloat problem using varieties of AQM schemes.

3 | CONTRIBUTION

To evaluate the performance of any AQM scheme it is important to test it over different TCPs and over different congestion levels. In previous studies the AQM algorithms performance is solely evaluated on transport layer [28, 29, 33]. Some authors, such as [30, 32, 39] and [35] have assessed the network layer metric, however they either used fewer metric or conducted their experiments on a single TCP which is not sufficient to explore the full capabilities of the investigated AQM schemes. In this paper we have conducted a detailed evaluation of popular AQMs such as CoDel and RED on network layer considering all the possible performance metric over different TCPs. This work will enable the researcher to see the behaviour of CoDel and RED in more depth from the network perspective over various TCP types and in different congestion settings. Our main contribution are summarised as follows.

- We use different traffic scenarios, such as (A) variable congestion and fixed payload VCFP, (B) variable payload and fixed congestion VPFC and (C) high congestion and high payload HCHP. We exploit such varying level of congestion and changing payload sizes to evaluate three performance metric such as, drop rate, link utilisation and queuing delay of the AQM algorithms in network layer.
- We show how the TCPs behaviour (congestion window) reflects on the AQMs performance (link utilisation, drop rate and queuing delay). Each AQM is assessed over six different types of TCPs.
- As an impact of both AQM schemes on drop rate, link utilisation and queuing delay, we show its influence on fairness among the TCP flows and the number of packets retransmitted by each flow.

4 | TOPOLOGY AND TRAFFIC

In this study, a standard dumbbell topology is used, as shown in Figure 3. It enables us to observe the impact of congestion produced by multiple flows on a bottleneck link router. The network traffic used is according to the File Transfer Protocol (FTP) [40]. There are *n* FTP pairs, where each connection

TABLE 1 Overview of recent work

Paper	AQM Variant Considered	TCP Variant Considered	Transport Layer Performance Metrics	Network Layer Performance Metrics
Augustu et al. [28]	DropTail, ARED, PIE, CoDel, GREEN and PINK	Yeah, Westwood, Cubic, Hybla, Illinois, Vegas, Newreno and HighSpeed	RTT and goodput	Queue occupancy
Khademi et al. [29]	CoDel, PIE and ARED	SACK	RTT and goodput	N/A
Jarvinen et al. [30]	CoDel, PIEupdate, PIEthresh, HRED, HRED (aggressive), SFQ CoDel	SACK1	N/A	Queuing delay and delay spike duration
Vyakaranal et al. [32]	CoDel, PIE and RED	Reno, Vegas and BIC	Packet drop, latency and throughput	N/A
Cardozo et al. [33]	CoDel, PIE, FQ-CoDel and BQL scheme	Reno	RTT and throughput	N/A
Grazia1 et al. [34]	DropTail, ARED, CoDel, PIE, GREEN, PINK	Yeah, Westwood, Cubic, Hybla, Compound, Vegas, Newreno and HighSpeed	RTT, goodput and fairness	N/A
Ye et al. [39]	ACoDel-IT, ACoDel-TIT, CoDel, PIE and ARED	NewReno	RTT	Queueing delay, link utilisation and packet drops
Palmei et al. [35]	CoDel and COBALT	Reno	RTT and goodput	Queue delay and queue occupancy
Kennedy et al. [36]	RED, PI, STR and MTP	Standard TCP	TCP window size	Queue length and drop probability
Dzivhani et al. [37]	CoDel and DropTail	Westwood, Illinois, Vegas	RTT, throughput and fairness	N/A

between the i^{th} FTP server (FS_i) and FTP client (FC_i) pair is established over the TCP. These connections use various TCPs, and are investigated over different levels of congestion, starting from lower to medium and high. Three, 10, and 14 FTP sources are used to represent the low, medium, and high levels of congestion, respectively. Furthermore, each TCP with different payload sizes is assumed to be at higher and fixed levels of congestion. We consider a case with high congestion and a high payload. The routers are configured with the CoDel (or RED) algorithm, and its performance is investigated in terms of packet sojourn time, link utilisation, average persistent queue, and drop rate over each TCP variant.

4.1 | Transmission control protocol (TCP)

The applications that require reliable transportation of data use the TCP as it offers peer connectivity at the transport layer and handles handshakes between connections. It encapsulates the incoming data bytes into packets and transmits them by assigning them sequence numbers. An acknowledgment is sent to the sender if the sequence number is received; otherwise, the packet is re-transmitted upon a pre-defined timeout [41]. The TCP is also responsible for balancing load in the congested network.

There are different types of TCPs, and each is designated for a specific purpose. Some TCPs function over wireless networks whereas others are intended for high-bandwidth-delay product (BDP) networks, congested networks, or fair flow rates. In addition to this, a TCP intended for high-speed performance never perform betters if it is used for a wireless network. In this study, we consider six variants of the TCP from different domains: TCP NewReno [42], TCP Vegas [43], Compound TCP [44], TCP SACK [45], TCP Cubic [46], and TCP Westwood [47]. These TCPs all fall into one of three categories: (A) congestion collapse, (B) lossy or wireless, and (C) high-speed TCPs. TCP NewReno (reactive/loss based)[3], TCP SACK (loss based) and TCP Vegas (proactive/delay based)[4] are congestion collapse-based TCPs, whereas Compound TCP (cTCP) (loss based and delay based) and TCP Cubic (loss based) are high-speed TCPs. TCP Westwood (loss based with bandwidth estimation) is a wireless TCP.

Whenever a packet is dropped by an AQM, it is taken as a congestion signal by the TCP, which instantly react to it by reducing its congestion window, which reduces the transmission rate. The congestion window is differently controlled by each variant of the TCP. Because CoDel senses congestion based on queue delay unlike RED, which detects it based on queue length, it is important to investigate CoDel's performance over different variants of TCP versus that of RED at different levels of congestion and packet sizes. This can give us a clear idea of whether a) queue delay-based algorithms are better than queue

[3] Reactive: The obtainable bandwidth of the connection is discovered based on network losses. Loss-based: Network congestion is detected by packet loss.
[4] Proactive: The incipient congestion is found by observing variations in the throughput. Delay based: The obtainable bandwidth in a network is estimated by packet delay.

TABLE 2 TCP variants along with their description

TCP Variant	Base	Predecessor	Intended Feature
TCP NewReno	Loss based	TCP Reno	Congestion collapse
TCP SACK	Loss based	TCP NewReno	Congestion collapse
TCP Vegas	Delay based	TCP Reno	Congestion collapse
Compound TCP	Delay-loss based	Vegas and HS-TCP	High speed
CUBIC-TCP	Loss based	BIC and H-TCP	High speed
TCP Westwood	Loss based with bandwidth estimation	TCP Reno	Wireless

length-based algorithms, and, b) if so, whether it is superior for any given TCP or only for specific ones. We have therefore chosen TCPs of different features that can adequately test the capabilities of CoDel. A brief description of the investigated TCP variants is provided in Table 2.

4.1.1 | TCP NewReno

The Reno Algorithm works better when the loss of a single packet occurs in a single window of data. However, when multiple packets are lost in the same window, it faces two prominent problems. First, Reno frequently takes a timeout as described in [48]. Second, multiple fast retransmission and window reductions occur as described in [49]. When an out-of-order segment arrives at the receiver, a duplicate acknowledgment (dup-ACK) is instantly sent to the sender. The sender can receive a dup-ACK for any of several reasons, such as network failure, reordering of the data segment by a network, replication of data segment, or replication of acknowledgment. Upon the receipt of three dup-ACKs, the sender assumes that a data segment has been lost. The TCP therefore retransmits the missing segment without waiting for the retransmission timer's expiry. This phenomenon is referred to as fast retransmit or fast recovery.

In TCP NewReno, the aforementioned problems associated with TCP Reno have been solved. Unlike Reno, the TCP NewReno is not pulled out of the fast recovery operation when partial ACKs[5] are received–where this is treated as an indication of multiple packet losses in a row from a single window–that can be retransmitted intelligently. TCP NewReno is sustained in the fast recovery mode until all outstanding data have been acknowledged.

4.1.2 | TCP SACK

The TCP SACK uses a nearly identical congestion control algorithm to that used by Reno. The SACK option follows the format discussed in [50] and does not affect the original congestion control algorithm when added to the TCP. Furthermore, the TCP SACK uses the same recovery method as Reno and is identically robust against out-of-order packets. The difference between the implementation of TCP SACK and that of TCP Reno is noticeable when multiple packets are dropped from a single window of data. The TCP SACK mechanism goes into the fast recovery operation when its sender receives dup-ACKs. In the same way as Reno, it reduces its congestion window to half its size and retransmits the packet. TCP SACK introduces a new variable `pipe` during fast recovery to estimate the number of outstanding packets in the network. The `pipe` variable is treated as follows: When either an old packet is retransmitted or a new one is sent, It is increased by one. On the contrary, a dup-ACK received by the sender (SACK option field) states that the out-of-order data have been delivered at the receiver, and `pipe` is reduced by one.

Any missing packet at the receiver forms a list, and as the sender get an opportunity to send, it retransmits the next packet from the list. If the list is empty (no missing packet), a new packet is sent. The TCP SACK leaves the fast recovery operation when it receives a recovery notification, which means that all outstanding packets have been acknowledged during fast recovery.

4.1.3 | TCP Vegas

TCP Vegas is a modified version of Reno that is more aggressive because it triggers its fast retransmission immediately upon receiving the first dup-ACK (rather than waiting for three dup-ACKs, as in case of RENO). For every segment sent, TCP Vegas measures its RTT alongside (and the duration of the timeout is computed). If a dup-ACK is received, TCP Vegas checks for it the expiry of its timeout period and retransmits the segment if it has expired. Furthermore, TCP Vegas integrates a bandwidth estimation feature that intelligently estimates the bandwidth by using the difference between the actual "`a`" and the expected "`e`" rates of flow. If the values of `e` and `a` are very close, the congestion window (*cwnd*) is increased in size because the network has the capacity to allow `a` to attain `e`. *cwnd* is reduced if `a` is significantly lower than `e`, and this condition is regarded as an indication of incipient congestion. TCP Vegas uses a window size based on the bandwidth estimation. The normalised difference 'Diff' of a particular segment in TCP Vegas is computed as follows [51].

$$\text{Diff} = (\mathtt{e} - \mathtt{a})\text{BaseRTT}, \tag{1}$$

[5] This identifies outstanding packets at the beginning of fast recovery duration

ALGORITHM 1 Simulation parameters

```
if Diff <Alpha then
    Increase the Congestion Window (CWND) linearly in the next RTT, that is, CWND ++
else if Diff >Beta then
    Reduce the CWND linearly in the next RTT, that is, CWND - -
else
    Do not change CWND;
end if
```

ALGORITHM 2 Pseudocode for CoDel

```
For each enqueuing packet (pkt)
S_t ← Now − E_t
if S_t ≥ T_d then
D_s ← 1;
end if
while D_s = 1 do
if first_above_time = 0 then
first_above_time ← 1;
D_i ← Now + Interval / √N;
else if Now > Drop_next then
N ← N + 1;
Drop(pkt);
else if S_t ≤ T_d then
first_above_time ← 0;
D_s ← 0;
Dequeue(pkt)
end if
end while
```

where BaseRTT[6] is the minimum round-trip time. During congestion avoidance, TCP Vegas controls its windows in a linear fashion. It defines two thresholds `Alpha` and `Beta` that are compared with the normalised difference Diff and consequently control the congestion window as follows:

4.1.4 | Compound TCP

Compound TCP (cTCP) is widely deployed in older operating systems of Microsoft, such as Windows XP, Windows Server 2003, Windows Vista, and Windows 7. Reno is a congestion collapse-based TCP model that suffers from the problem of underutilisation if used in high-BDP (bandwidth-delay-product) networks. Because Reno requires a long time to stretch its window to such a value (High BDP), the cTCP uses the synergic approach of combining both loss-based and delay-based congestion avoidance models (and thus it is referred to as compound TCP) to solve the problem faced by Reno. It has two special features: (1) When a network is sensed to have been underutilised, it aggressively increases its window size to achieve the desired throughput. (2) Once the link is full and a bottleneck queue has formed, further window increases can cause the problem of TCP unfairness. Therefore, a delay-based approach (like Vegas) is used to reduce the sending rate. The sending window *win* is therefore controlled by the loss-based and delay-based components and is computed as follows:

$$win = \min(cwnd - dwnd, awnd), \tag{2}$$

where *dwnd* is the delay-based component derived from TCP Vegas. It renders the cTCP more scalable in high-BDP networks. *awnd* is the advertised window from the receiver and *cwnd* is the loss-based component that is nearly identical to the convention congestion window. *cwnd* is increased on the arrival of ACK and is calculated as follows:

$$cwnd = cwnd + \frac{1}{cwnd + dwnd}. \tag{3}$$

When a new connection is stared up, the cTCP mirrors the slow behaviour of Reno. As discussed in [53], an exponential increase can work well even in a fast and long-distance network. The delay-based component *dwnd* is set to zero at the start of a connection but works effectively during the congestion phase.

4.1.5 | CUBIC transmission control protocol (CUBIC-TCP)

Compared with the conventional TCP [54], CUBIC-TCP leverages a Cubic function to substitute the linear window increase function. This not only improves scalability, but also increases stability in long-distance and fast networks.

Fast and long-distance networks usually suffer from the problem of low utilisation [55] when using a standard TCP (Reno). Congestion is followed by a slow increase in the size of the congestion window that leads to a large bandwidth delay product (BDP). This issue is associated with standard TCP and is applicable to Reno-style TCP standards [56], such as SACK [57], TFRC [58], and SCTP [59].

In CUBIC-TCP, the congestion control algorithm of the conventional TCP is modified to tackle the aforementioned problem. The CUBIC-TCP inherits its window-increase function from its predecessor, the BIC-TCP [60]. The window is increased aggressively until it is close to the saturation point. However because of its proximity to the saturation point, the increase in window size slows further. The CUBIC-TCP uses the following window growth function:

$$W(t)_{cubic} = c(t - t_p)^3 + W_{max}, \tag{4}$$

where W_{max} is the previously known size of the window (right before congestion occurs and the window reduces in size in fast

[6] BaseRTT is the RTT of a segment in an uncongested state as described in [52]. The expected flow rate is basically computed from BaseRTT.

recovery), c is a constant that expands the window aggressively in high-BDP networks, t is the time that has elapsed after the reduction in the size of the congestion window in response to Dup-ACK (or ECN-Echo ACKs), and t_p is the period taken by Equation (4) to grow the window size to W_{max} (if no loss event occurs), which can be computed as follows:

$$t_p = \sqrt[3]{\frac{W_{max}(1-\beta_{cubic})}{c}}, \tag{5}$$

where β_{cubic} is the decay factor for CUBIC multiplication, that is, as a packet loss is sensed, the given window size is reduced to

$$W(0)_{cubic} = W_{max}\beta_{cubic}. \tag{6}$$

Reliant on the given size of the window, Cubic operates in three modes. First, it operates in a TCP-friendly region if the size of the window is smaller than the *cwnd* that a standard TCP achieves in time t after the previous loss event has occurred. Second, Cubic operates in the concave[7] region if the given window size is smaller than W_{max}. Finally, Cubic runs in the convex[8] region if the given window size is larger than W_{max}. These feature render the Cubic stable and scalable, and provide RTT fairness (the same as in BIC-TCP). It is also friendly to the standard TCP.

CUBIC-TCP has been tested and deployed over the Internet, and has shown significant improvements compared with other types of TCP.

4.1.6 | TCP Westwood

In a wired network, packets are lost due to congestion at the router. However, in a wireless medium, channel impairments (fading radio or noisy channels) are expected to cause such losses. TCP Reno cannot discriminate wireless loss from congestion loss. Consequently, it severely reduces the size of its congestion window while reacting to wireless loss, and this reduces the transmission rate. TCP Westwood (TCPw) is the modified version of TCP Reno where the rate of acknowledgment of reception is monitored by the sender of TCPw and, from this, the data packet rate attained by the connection is estimated.

When the sender detects packet loss, the sender of TCPw sets appropriate thresholds for the slow start (*ssthresh*) and congestion window (*cwnd*) by using bandwidth estimation techniques, that is, the available bandwidth is probed during the congestion avoidance phase. If (n) dup-ACKs are received, the maximum capacity of the network is reached. Therefore, *ssthresh* is assigned with the available pipe size, *cwnd* is set to be equal to *ssthresh*, and the congestion avoidance phase starts probing once again for a new, attainable bandwidth. Thus, *cwnd* is reduced to the estimated bandwidth rather than its window size being halved (as in Reno). TCPw prevents excessively conservative reductions in *ssthresh* and *cwnd* and thus guarantees faster recovery.

4.2 | Applied AQM schemes

In this study, the RED and CoDel schemes were applied to the routers, and their performance metrics were compared over TCP variants in different network environments.

4.2.1 | Random early detection (RED)

This algorithm detects network congestion by measuring queue length. RED uses an exponential weighted-mean sliding model to calculate average queue length $AvgQ$;

$$AvgQ_{t+1} = (1-\omega)AvgQ_t + \omega Q_{inst}, \tag{7}$$

where ω is weight , $AvgQ_t$ is the average queue length at time t, and Q_{inst} is the instantaneous queue length. Bases on Equation (7), the probability of packet drop is computed as follows:

$$P_{drop} = \begin{cases} 0 & \text{if } AvgQ < T_{min} \\ \dfrac{AvgQ - T_{min}}{T_{max} - T_{min}} Max_P & \text{if } T_{min} < AvgQ < T_{max}, \\ 1 & \text{if } AvgQ > T_{max} \end{cases} \tag{8}$$

where T_{max} and T_{min} represent the maximum and the minimum thresholds, respectively, and Max_P denotes the maximum probability of dropping a packet. If $AvgQ$ is in between T_{max} and T_{min}, the probability of packet drop P_D (modified form of Equation 8) can be computed as

$$P_D = \frac{P_{drop}}{1 - CP_{drop}}, \tag{9}$$

where C is the packet counter that shows the number of incoming packet in the queue since the previous packet was dropped.

If the operations are globally synchronous, they can be handled efficiently by the RED algorithm while controlling transient congestion [62]. The main drawback of RED is the dependence of its performance on parameter tuning [63] [64]. $AvgQ$ is also highly sensitive to the level of congestion.

4.2.2 | CoDel

CoDel is a recently proposed approach to mitigate bufferbloat. Each packet's time is recorded at input to the enqueuing process, and at dequeuing, the time difference, referred to as sojourn time $\mathtt{S}_t$, is calculated to determine the queuing delay. Based on $\mathtt{S}_t$, certain actions are taken. First, $\mathtt{S}_t$ is compared with a threshold value (known as the target delay '$\mathtt{T}_d$'). If $\mathtt{S}_t$ is greater than $\mathtt{T}_d$, the algorithm goes into the dropping state '$\mathtt{D}_s$' and sets the next

[7] In congestion avoidance, when an ACK is received, *cwnd* is increased as $\frac{W(t+RTT)_{cubic} - cwnd}{cwnd}$ (for each ACK).

[8] The convex profile guarantees that the *cwnd* grows gradually at start and slowly increases its growth rate.

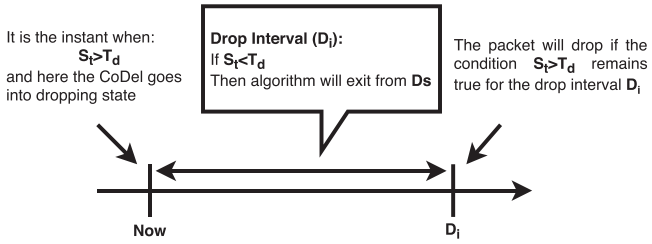

FIGURE 1 Dropping state

drop interval 'D_i' as shown in Figure 1. Before reaching D_i, if the sojourn time falls below T_d, it exits 'D_s' and the packet is simply forwarded.

The control law for the next drop interval D_i is defined as

$$D_i = Now + \frac{Interval}{\sqrt{N}}, \tag{10}$$

where N is the drop count. If N increase, D_i declines and, consequently, more packets are dropped, thereby emptying the buffer. The purpose of *Interval* is to allow enough time to the sender to react to a packet drop, that is, reduce its sending rate, to reduce S_t to below T_d. The interval considered for the sender to use should be at least the average RTT over the Internet. By default, *Interval* is set to 100 ms and target delay to be 5% of *Interval* [22]. The dropping state is cleared if the sojourn time is below the target delay value during the dropping state. The Pseudocode for CoDel [61] is given in the following table while the process is shown in the flowchart in Figure 2.

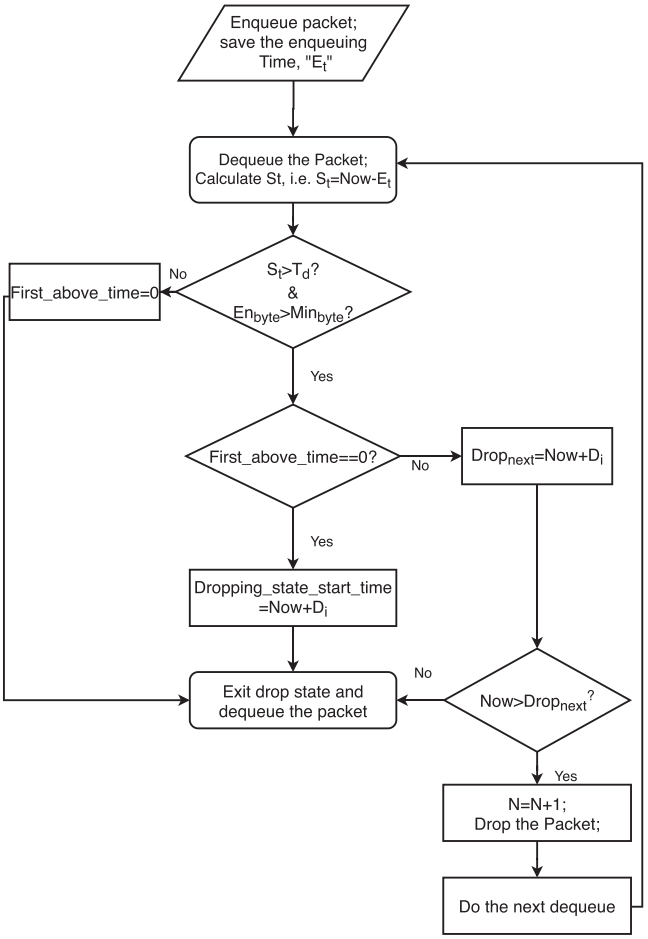

FIGURE 2 CoDel's flowchart

5 | EVALUATION

To evaluate the performance of CoDel and compare it with that of RED over different TCPs in different network environments, we configured the ns-2 simulator [65] with three settings. First, TCP variants were used at different levels of congestion with CoDel and RED as AQMs, Second, the AQMs are tested over different payloads while keeping the level of congestion fixed, and high congestion with large packet sizes were then used to assess the performance of both AQMs for extreme cases. We also considered six types of TCPs: TCP NewReno, Compound TCP, TCP Westwood, TCP SACK, TCP Vegas, and TCP Cubic.

Section 5.1 overviews the baseline model, factors, and parameters. Section 5.2 illustrates the performance metrics to assess the behaviour of CoDel over TCP variants and different levels of congestion. Section 5.3 compares the results of all TCPs under the VCFP, VPFC, and HCHP scenarios.

5.1 | Simulation environment

We installed Network Simulator 2—a.k.a., ns-2—(version 2.36) on Ubuntu 16.04 operating system. For writing the script for our topology we used OTcl programming language. We also used AWK language for the data extraction. The data acquired from

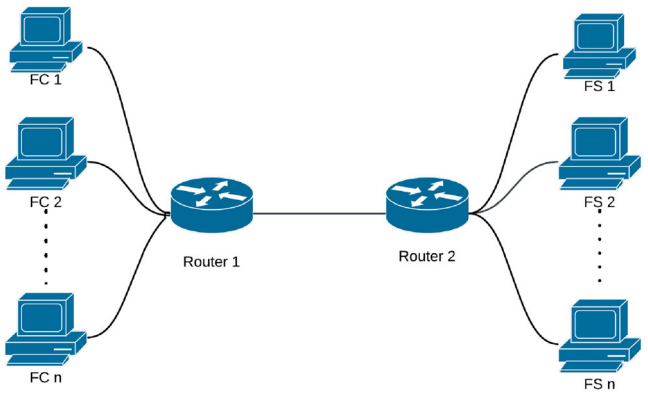

FIGURE 3 Network topology in the baseline simulation scenario

the ns-2 was further processed in MATLAB 2019b for analysis and plotting purposes. We used a computer with Intel(R) Core(TM) i7 CPU for the simulation.

Figure 3 shows the network model with a dumbbell topology. This scenario illustrates several FTP peers that act as network traffic generators. The FTP client *FC* sends data to the FTP server *FS* via two routers on the link. *FC* and *FS* are linked to the routers through a 15-Mbps bandwidth link with a 15-ms propagation delay. The routers form a bottleneck link at a speed of

TABLE 3 Simulation parameters

No.	Parameter	Value
1.	TCP variants	NewReno, SACK, Compound, Westwood, Vegas and Cubic
2.	Link capacity	15 Mbps for node–router, 3 Mbps for router–router (bottleneck link)
3.	Link delay	15 ms node to router, 0 ms router to router
4.	Buffer size	1000 Packets
5.	Traffic type	FTP
6.	Queuing algo	CoDel and RED
7.	Packet size	540 Bytes, 1040 bytes, and 1460 bytes
8.	Simulation time	100 s

3 Mbps with zero propagation delay. As it is clear in this topology, the router–router link does not feature a propagation delay, because our focus is to measure the sojourn time, adding extra jitters or delays can leads to erratic values. Therefore it is considered to be the best scenario to test the performance of the investigated AQMs under ideal conditions.

In the VCFP experiment, numerous levels of congestion were used, that is, low, medium, and high levels. In the low levels of congestion, there were as few as three FTP pairs, whereas in the medium and high levels of congestion, there were six and 14 FTP pairs, respectively. The connection between FC_n (subscript n denotes the number of nodes, that is, three, six, or fourteen) and FS_n is established by the TCP.

For the experimental setup of VPFC, the dumbbell topology was used with the maximum and fixed levels of congestion, and packets of different sizes were transmitted by FC_n. The rest of the setup for this experiment was same as that of the first one. In the final experiment, that is, HCHP, high congestion and a high payload size were considered. All experiments were repeated for each TCP variant. The buffer size was fixed to 1,000 packets, the default size for CoDel. The trials showed the impacts of different levels of congestion, payload sizes, and TCP variants on the performance of the CoDel algorithm. The TCPs used in these experiments were NewReno, cTCP, TCPw, SACK, Cubic, and Vegas. The variants were added to NS2 version 2.36. Table 3 illustrates the simulation parameters and setup for the experiment.

5.2 | Performance metrics

The following performance metrics were used to evaluate the CoDel algorithm over TCP variants. In the following equations, n represents the number of recurrences and t refers to the simulation time in seconds.

5.2.1 | Link utilisation

It is a ratio of the rate of packets dequeued by an AQM to the available link capacity, that is

$$L_U = \frac{\left(\sum_n \frac{B_n}{\Delta t}\right)\eta}{BW} \times 100\%, \tag{11}$$

where L_U denotes the link utilisation of the bottleneck link, $\sum_n B_n$ is the total number of bytes dequeued by the AQM algorithm, η is bits per byte, and BW is the capacity of the bandwidth at the bottleneck. For $L_U = 100\%$ the pipe is 100% full, and is empty when it is zero.

5.2.2 | Drop rate

The drop rate is the ratio of the number of packets dropped by an AQM to the total enqueued packets. This parameter tests the robustness of an AQM. In contrast, an ideal AQM does not drop (or drop very less amount of) packets even in high congestion state whilst maintaining a reasonable amount of packet's queue in the buffer. However, In practice, when the congestion increases, it violates the dropping state of the AQM which results in increasing the dropping of packets. The dropped packet is anticipating by the TCP as a congestion signal, which results in slowing down its sending rate. The role of an AQM is to retain the good queues in the buffer and avoid the excessive queuing delay (by draining the bad queues). Note that this drop rate is strictly related to the network layer, it does not take into account the amount of packets dropped by the link or channel at physical layer (which is collectively considered by the retranmission packets of TCP).

5.2.3 | Queue length

This shows the dynamic queue behaviour of CoDel. It is the average number of standing queues during the entire period of the simulation.

5.2.4 | Queuing delay

This metric shows the delay experienced by a packet in the queue from ingress to egress. In contrast, queuing delay is the time taken by a packet when it goes into a buffer, stays in the queue and then dequeues from it. The sojourn time (or queuing delay) is handled differently by different AQMs. For instance, RED algorithm cares for maintaining the queue length in the buffer, and do not hassle for the delay experienced by a

packet from enqueue to the dequeue. On the other hand, CoDel is concerned more about the queuing delay. It restricts the packets from passing the buffer within the interval of certain target delay. If a packet did not meet this requirement then it is dropped[9]. Note that the sojourn time is strictly related to the network layer and it does not takes into account the propagation delays of the link or channel. On the contrary, the round trip latency or delay can be obtained from the round-trip-time (RTT) of the TCP which is the summation of queuing delay and propagation delays. To narrow down our analysis (to network buffers) we use queuing delay to rigorously consider the packet's delay from ingress to egress. The queuing delay is expressed in milliseconds.

5.2.5 | Flow utilisation

It illustrates the utilisation of each TCP flow:

$$F_U = \frac{f^{(N)}\left(\frac{\sum_n B_n^{(f)}}{\Delta t}\right)\eta}{BW}, \tag{12}$$

where $\sum_n B_n^{(f)}$ is the total number of bytes in each flow– $f = \{1, \ldots, N\}$, during the transmission period, $f^{(N)}$ is the number of TCP flows (levels of congestion) used in the simulation, η is bits per byte, and BW is the link capacity of the bottleneck bandwidth. We characterised the difference among flows with the coefficient of variance, which is the ratio of the standard deviation to the mean of the flows (in any TCP variant).

5.2.6 | Retransmission packets

This metric is the ratio of total number of retransmitted packets to the total amount of sent packets for the entire simulation period. This metric is related to TCP connection. If any packet is dropped either in network layer, or physical layer, it is retransmitted accordingly.

5.3 | Results and discussion

In this section, we use the aforementioned three scenarios to describe the performance of CoDel as compared with RED using various TCP variants. We focus on link utilisation and the delay experienced by each packet from enqueuing to dequeuing (packet sojourn time), and the drop rate incurred by the AQM used.

[9] For dropping, CoDel gives a certain margin of interval, if the packet dequeues within that interval then it is not dropped, otherwise it is dropped.

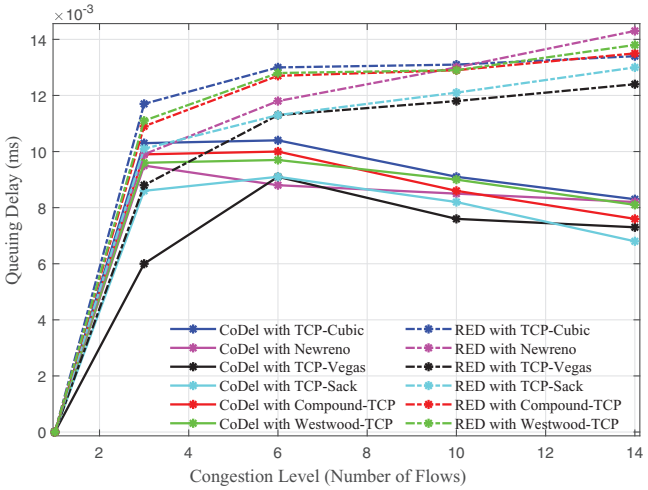

FIGURE 4 Comparison in terms of average queuing delay (VCFP) between CoDel and RED

5.3.1 | Variable congestion and fixed payload (VCFP) setting

Average queuing delay: Our first simulation evaluates the performance of CoDel versus RED at different levels of congestion with different types of TCPs. The average queuing delay is shown in Figure 4, which demonstrates that CoDel is all time experiencing less queuing delays than RED at different levels of congestion and with any variant of TCP. Moreover the gap between RED and CoDel is small at lower level of congestion. However, as the level of congestion grows in terms of number of flows, the RED's queuing delay increases.

On the contrary, CoDel's queuing delay decayed with an increase in the level of congestion. In the RED algorithm, the minimum queue length threshold T_{min} was fixed to five packets, which means that at any time, it held five packets and none was dropped irrespective of the delay. Because RED controls queuing occupancy in terms of packets rather than the delay experienced by them, it monitored the average queue length of the incoming traffic. As this exceeded the upper threshold, packets were dropped to dissipate the standing queues, and if the queue length remained between upper and lower thresholds, packets were dropped randomly based on Equation (9).

Conversely, the CoDel algorithms has no fixed packets, and directly controls packet delay rather than queue length. Moreover, CoDel's controller uses the stochastic gradient learning procedure as shown in Equation (2). The control loop of CoDel starts with the dropping of few packets if the level of congestion is small. However, the drop rate increase with the level of congestion to ensure that bad queues are drained and only good ones remain in the buffer.

Figure 5 illustrates packet dropping for different levels of congestion for TCP Cubic, which clearly shows that when congestion was high (14 flows) the drop rate was large and drained the persistent queue (as shown in Figure 6.) Consequently, the reduced queues experienced shorter delays. For instance, CoDel with Cubic experienced considerably longer delay with six flows,

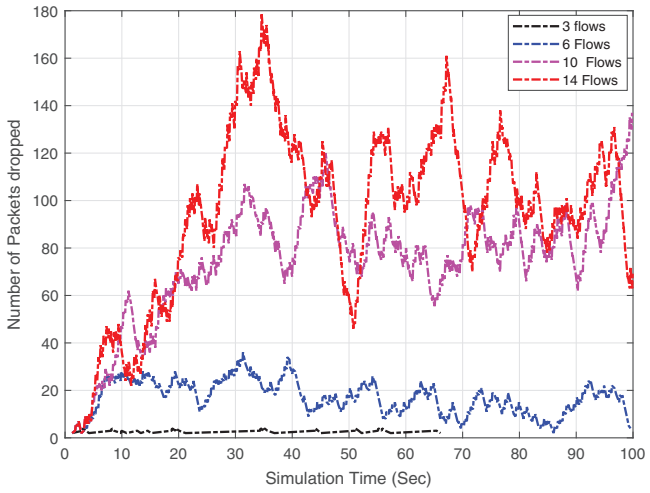

FIGURE 5 CoDel over TCP Cubic: Packets dropped at different levels of congestion

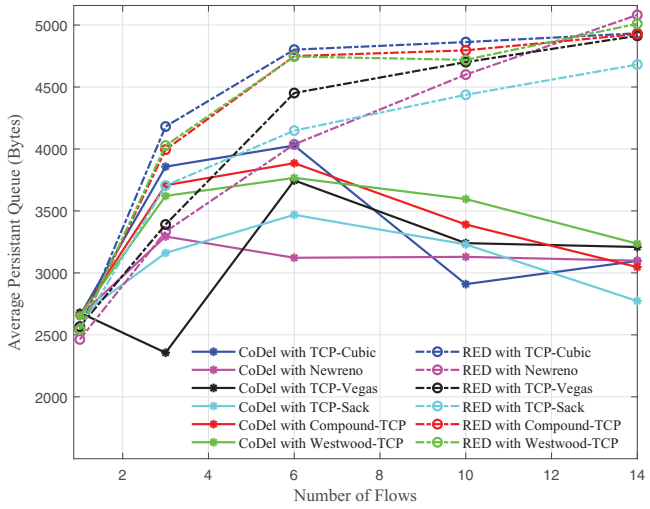

FIGURE 6 CoDel versus RED: Comparison of average queuing length (VCFP)

but as the number of flows increased to 10, the drop rate increased (and queuing delay decreased as shown in Figure 4). Because of the higher drop rate, a large number of queues were dissipated (this increase in drop rate had adverse effects on link utilisation that are discussed in the following section).

Figure 6 shows the queuing delay, at low congestion (one to three FTP sources), there was a small difference in queue size between RED and CoDel except in Vegas (which has a built-in bandwidth estimation feature). But as congestion increased, the number of standing queues of RED drastically increased whereas that of CoDel decreased (owing to the controller's stochastic gradient learning procedure). TCP Cubic did not perform well compared with the other variants. As we considered a congestion collapse network (with a 3 Mbps bottleneck link speed and 0 ms propagation delay) rather than a high-BDP network, its performance did not surpass that of any RENO-

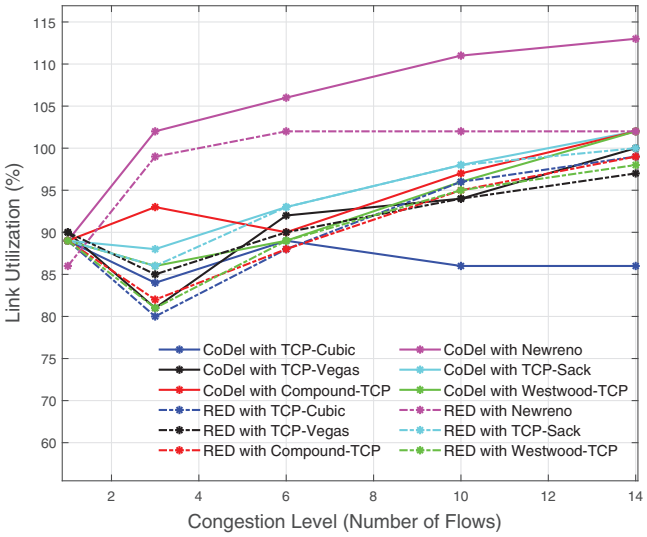

FIGURE 7 CoDel versus RED: link utilisation at VCFP

style TCP[10]. However, as the level of congestion increased, its delay experience became steady (with RED) or declines (with CoDel).

CoDel and RED performed better with TCP Vegas because it is based on congestion collapse TCP and possesses the bandwidth estimation feature, which enables it to learn about the available bandwidth. The AQMs (CoDel and RED) subsequently drained the bad queues. Likewise, TCP SACK also performed well because it is a Reno-style TCP and has a built-in SACK Option that intelligently reports the DUP-ACKs, and, as a result, the transmission rate is well controlled at higher congestion. TCP NewReno is a loss-based TCP that has no prior knowledge of bandwidth, and can respond to only dropped packets by an AQM scheme. Its implementation is considerably better with CoDel than with RED. cTCP is used for high-speed networks but uses the congestion avoidance algorithm of Reno, which makes it TCP friendly. It delivered average performance when used in congestion collapse networks. It performs much better if used in a high-speed network.

Link utilisation: Figure 7 shows the performance of CoDel versus RED in terms of link utilisation at different levels of congestion. The link utilisation was computed for all TCP variants using Equation (11). For each level, the average value was derived. It is clear that link utilisation depends on the choice of AQM as well as the TCP variant that establishes the connection. CoDel was better than RED over all TCP variants (except TCP Cubic) at different levels of congestion. As mentioned above, we considered a small BDP network with zero propagation delay in the bottleneck link. Therefore, TCP Cubic used the TCP-friendly region to achieve the same throughput as the standard TCP, which is why its utilisation curve oscillated around 75–85% of the total capacity available for all levels of congestion (in case of CoDel). However, in case of RED, TCP Cubic performed

[10] Cubic is intended for only high-speed networks. If it is used in congestion collapse networks, it delivers poorer performance than the RENO-style TCPs because they are designed for congestion collapse networks.

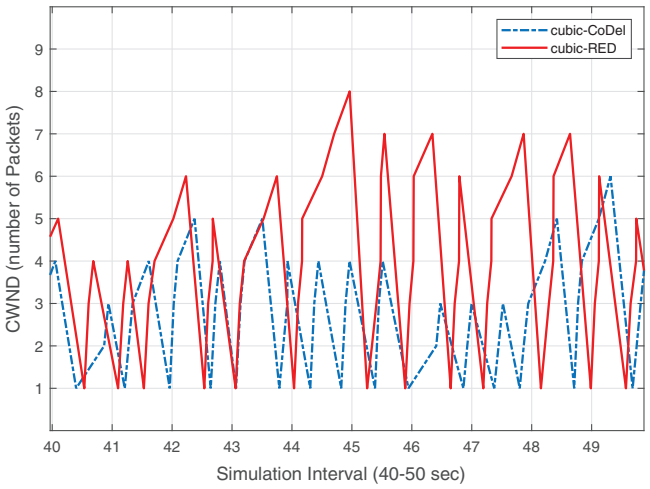

FIGURE 8 Cubic: Comparison of cwnd between CoDel and RED at VCFP

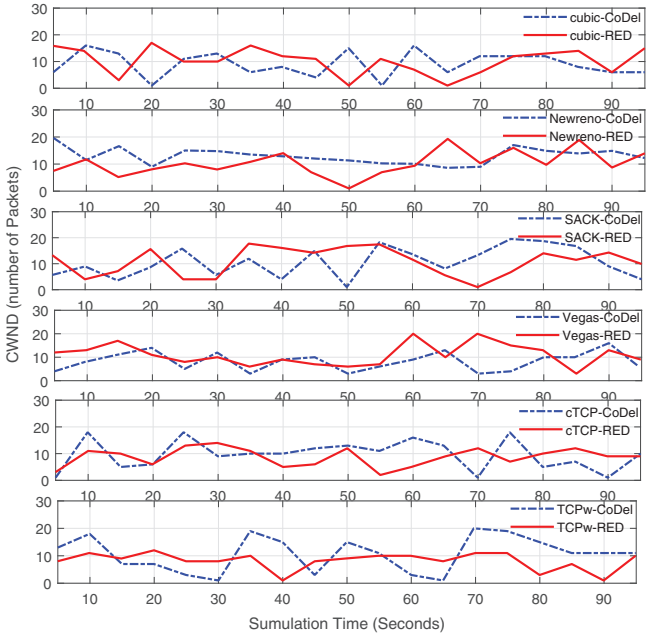

FIGURE 9 CoDel versus RED: Comparison over TCP variants (three flows) in terms of *cwnd*. T

well at high congestion levels (10 – 14 flows) and the utilisation curve is oscillating around 85–90%.

In Figure 11, the average drop rate of CoDel with Cubic is much higher than that of RED. If the packets were frequently dropped as a result of higher queues building up, W_{max} shown in Equation (4) could not have reached the maximum operating limit (concave or convex region), and the congestion window would have had to frequently be reduced as shown in Equation (6). On the contrary, RED algorithm dropped fewer packets on average and utilised more network links. Thus, Cubic's window increased to a greater extent than that of CoDel. This is shown in a time snapshot of Cubic's '*cwnd*' analysis for CoDel versus RED in Figure 8.

Figure 8 shows the congestion window of a single flow, considered randomly from among the 14 flows at higher levels of congestion. The behaviours of the flows were similar. For clarity, a snapshot was taken at the center (simulation time from 40 to 50 s). In all flows, Cubic's *cwnd* with CoDel had a smaller value than that of RED for the following reasons: (1) CoDel ensured that the sojourn time was shorter than the target delay, and as this was crossed, a packet was dropped. (2) The greater the number of packet drops occurred, the frequent was the congestion signal sent to the Cubic's sender, and, consequently, the smaller the values of *cwnd* was—that is, sending rate was reduced according to Equation (6). RED dropped packets based on a queue length, and was not as aggressive as CoDel, and thus dropped fewer packets. Thus, RED with Cubic operated in the concave region that resulted in sending more packets.

Because TCP NewReno and other Reno-style TCPs (such as SACK) are designed for congested networks, they fully utilised the link. The overall performance of CoDel with Vegas, cTCP, and TCPw was better than with RED. In Figure 7, most TCPs exhibit strange behaviour with the three FTP sources, where the utilisation plots declined for all TCP variants except NewReno (with both RED and CoDel) and cTCP (with CoDel). They then increased linearly (except cTCP with CoDel, which declined at six FTP sources).

This behaviour can be explained by analysing the values of *cwnd* for the three and six FTP sources shown in Figures 9 and 10, respectively. A single flow is considered (picked randomly) from both cases and the *cwnd* value is averaged for every 5 s for a simulation interval of 100 s. At a lower level of congestion (three to six FTP sources), the difference was not distinct from the analysis of the value of *cwnd* of Cubic with CoDel and RED. The link utilisations of Cubic with both AQMs (Figure 7) at three and six FTP sources had a minute difference (Cubic with CoDel had a slightly higher utilisation than RED). However, NewReno, which is more suitable for our topology, recorded higher network utilisation with both AQMs. This is also evident from the value of *cwnd*, where CoDel exhibited a more consistent behaviour than RED (i.e. transmitted more packets than RED). Our third TCP variant, SACK, performed better with CoDel at all levels of congestion. Its behaviour was prominent in terms of the size of the congestion window, where the value of *cwnd* of SACK with CoDel reached to its peak more frequently than with RED (with three FTPs).

However with six FTPs, we did not observe a prominent difference (because at six flows, SACK recorded similar utilisations for both AQMs). Vegas controlled its window based on the fluctuation in RTT. While there was no propagation delay in the bottleneck link, the overall end-to-end RTT delay was influenced by queuing delay at the routers. CoDel created far shorter queue delays than RED as shown in Figure 4. Vegas with CoDel at three FTP sources exhibited consistent performance but RED recorded peaks more often (which impacted utilisation, where Vegas with RED had slightly higher utilisation than CoDel at three FTPs). However, as the traffic increased to six FTPs the value of *cwnd* of Vegas with CoDel increased more than that of RED. cTCP controlled cwnd based on a

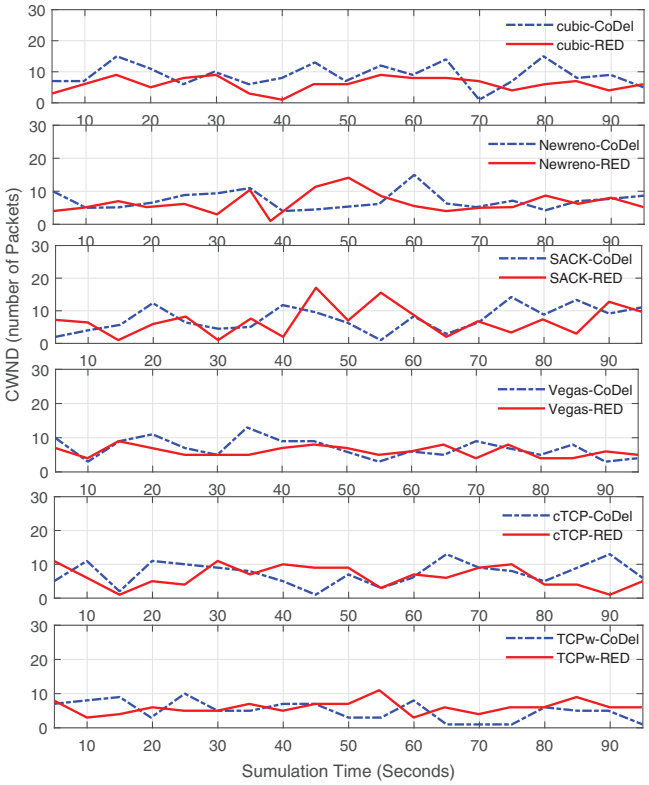

FIGURE 10 CoDel versus RED: Comparison over TCP variants (six flows) in terms of *cwnd*

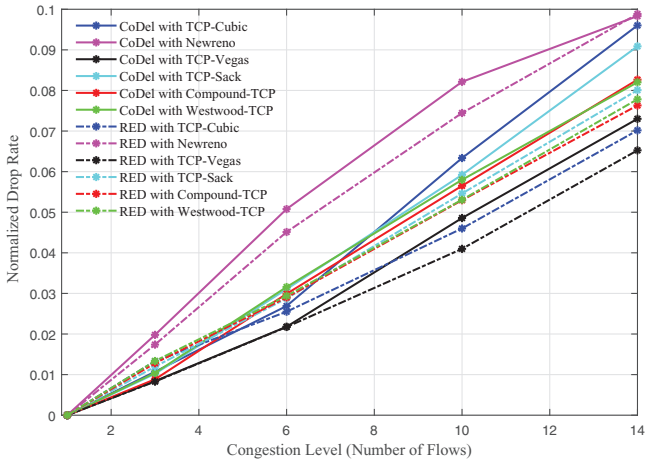

FIGURE 11 CoDel versus RED: Comparison in terms of drop rate (VCFP)

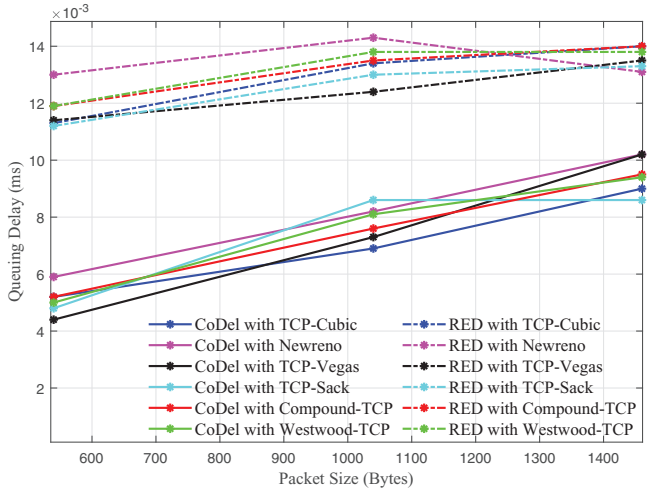

FIGURE 12 Comparison of average queuing delay of CoDel and RED at high congestion and different packet sizes.

delay-based component (as did Vegas), and had same function for *cwnd* as the standard TCP. Therefore, cTCP performed much better with CoDel than with RED because the former induced shorter delay (as shown in Figure 4). The function of Vegas benefited from this shorter delay and set its throughput accordingly (using the HTCP window increase function). Westwood is designed for wireless and lossy scenarios. Because, in the case considered here, a packet loss occurred only when there was congestion. TCPw increased its window size to a greater extent with CoDel than with RED at three FTPs. However, with six FTPs, we did not observe any prominent difference in its windows size.

Drop rate: Figure 11 presents the drop rate of CoDel versus RED at different levels of congestion. During the simulation, the average drop rate of RED was lower than that of CoDel over all TCP variants except NewReno at higher congestion, where RED and CoDel exhibited the same behaviour. In Figure 7, as NewReno attempts to utilise the maximum capacity of the available link with both RED and CoDel algorithms, and aggressively increases window size, its drop rate increases to greater that with other TCP types. Moreover, the TCP Cubic dropped more packets with CoDel than RED. The packet-dropping behaviour of TCP Cubic was clear from CoDel's law of control law (shown in Figure 5), which drained long queues. CoDel learned from the level of congestion and adapted its queue length. Figure 4 shows that if a packet sojourn time exceeded the target delay, it was dropped. This mechanism enabled CoDel to maintain a short queuing delay, but also led to an increased average drop rate.

5.3.2 | Variable payload and fixed congestion (VPFC) setting

Average queuing delay: Figure 12 shows the average queuing delay for a fixed level of congestion (higher) and different payloads over the TCP variants. CoDel delivered much better performance than RED when the packet size was small (i.e. 500 bytes). As packet size increased, CoDel's average queuing delay increased, and the gap in performance between CoDel and RED decreased at all TCP variants. For instance, CoDel with TCP NewReno at a payload of 500 bytes experienced a delay of 5.9 ms, and RED with the same connection and payload exhibited a delay of 13 ms. It therefore, experienced a delay 7.9 ms shorter (a difference of 75.13%) than that of RED. With

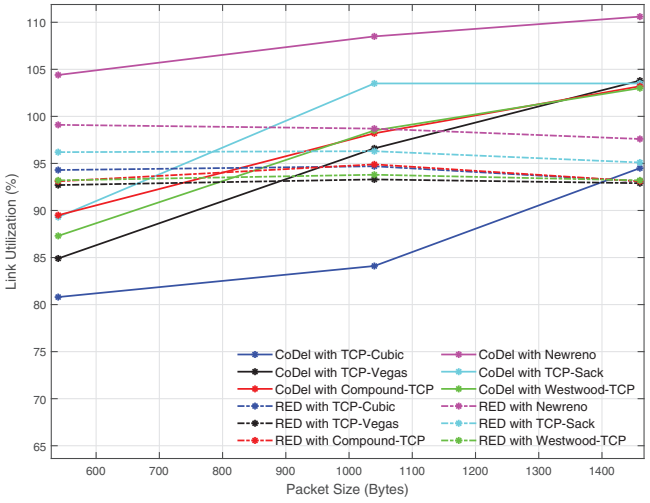

FIGURE 13 CoDel versus RED: Comparison in terms of link utilisation at different packet sizes

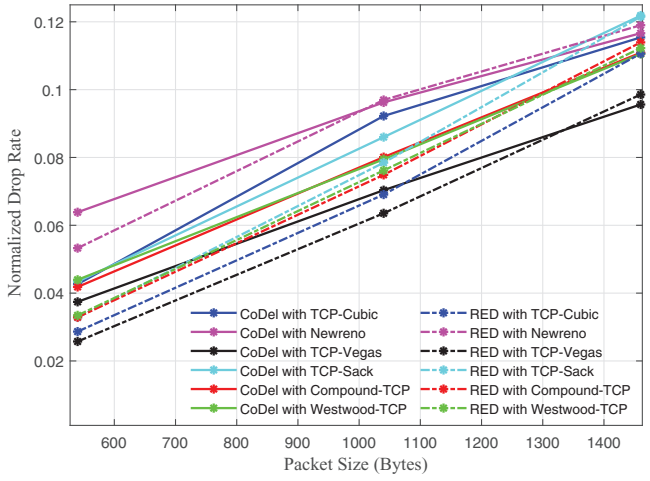

FIGURE 14 CoDel versus RED: Comparison in terms of drop rate with different packet sizes

a packet size of 1460 bytes, CoDel experienced a delay 2.9 ms shorter delay than that of RED (difference of 24.9%).

Link utilisation: Figure 13 shows the link utilisation for VPFC over the TCP variants. The behaviour of RED with all TCP variants was stable at different payloads, and it utilised between 86% and 93% of the link. However, the link utilisation of CoDel violently oscillating with changes in payload and TCP variant. This is because CoDel is delay based, and adapted to the increase in congestion. However, RED monitors queue length, and thus dropped packets when the queue length exceeded a defined threshold.

Drop rate: Figure 14 illustrates the behaviour of CoDel compared with RED over TCP variants with VPFC. It is evident that with packet sizes ranging from 500 bytes to 1000 bytes, RED's average packet drop rate was lower than that of CoDel. However, at 1460 bytes, CoDel performed better with NewReno, Vegas, cTCP, and TCPw. An increased in the byte size of a packet led to an increase in queue length. CoDel's performance was consistent and adaptive regardless of the payload size. However, RED increased its average drop rate when queue length became longer than the threshold. The drop rate of TCP Cubic was lower with RED for all packet sizes. CoDel and RED exhibited similar behaviours with TCP SACK at higher congestion and larger packet sizes.

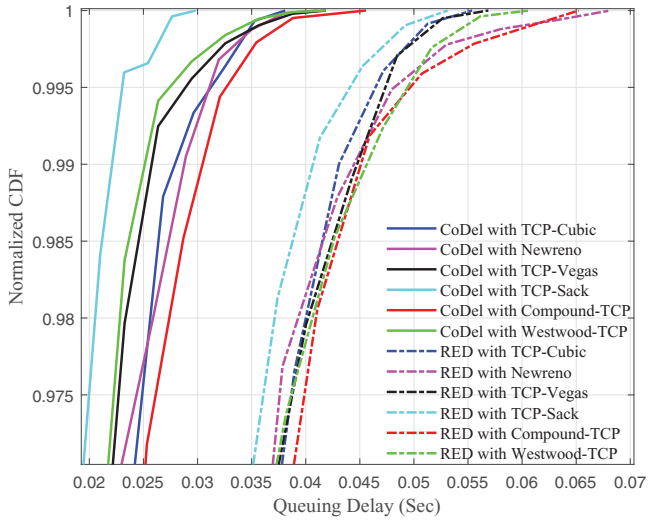

FIGURE 15 CDF of the queuing delays for CoDel and RED over TCP variants considered at HCHP

5.3.3 | High congestion and high payload (HCHP) setting

Cumulative distribution function (CDF) of the queuing delay: Figure 15 quantifies the long term queuing behaviours of CoDel and RED over TCP variants at HCHP as a cumulative distribution function of queuing delay. Each TCP variant underwent queue delays at different times. The queuing delays of both AQMs were very different. CoDel over TCP variants had a queuing delay of shorter than 25 ms for most packets, whereas for RED, this was approximately 40 ms. It is clear from Figure 15 that CoDel delivered exceptional performance in terms of queuing delay compared with RED over all TCP variants. As an example, CoDel with TCP SACK had 99% of its packets dequeued with a queuing delay of 22.11 ms, whereas this delay was 40.68 ms for RED.

Link utilisation: Figure 16 shows the behaviour of CoDel compared with RED in terms of link utilisation at HCHP over different TCP variants. It was averaged every 5 s for a simulation interval of 100 s. It is clear that CoDel with high congestion and large payload size tended to utilise more of the available link. However, RED remained in the underutilised region, that is, below 95% (except for TCP NewReno, on which it utilised approximately 97% of the link capacity).

Burst drop rate: We show the burst drop rates of the AQMs in Figure 17. The burst drop rate of CoDel was stable and fluctuating less than that of RED. CoDel delivered smooth performance for all TCP variants except NewReno, whereas RED

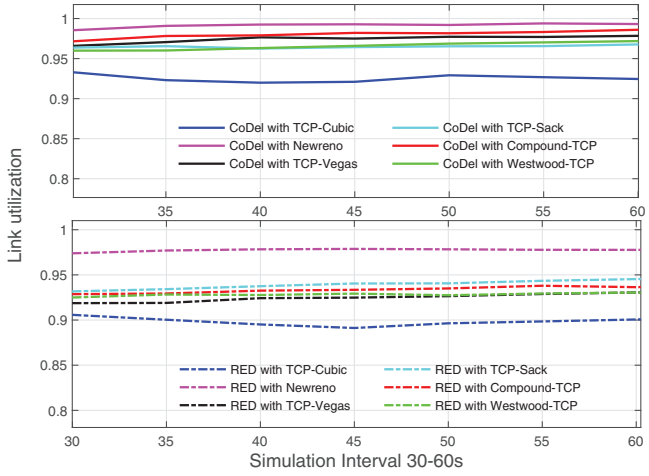

FIGURE 16 CoDel versus RED: Comparison in terms of link utilisation at HCHP

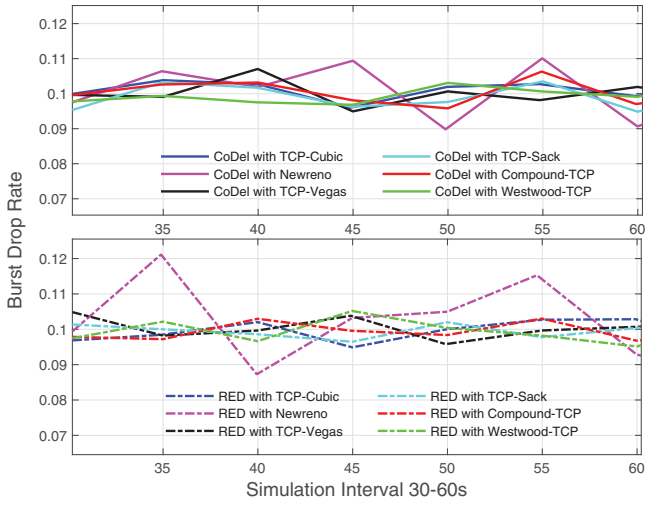

FIGURE 17 CoDel versus RED: Comparison in terms of burst drop rate at HCHP

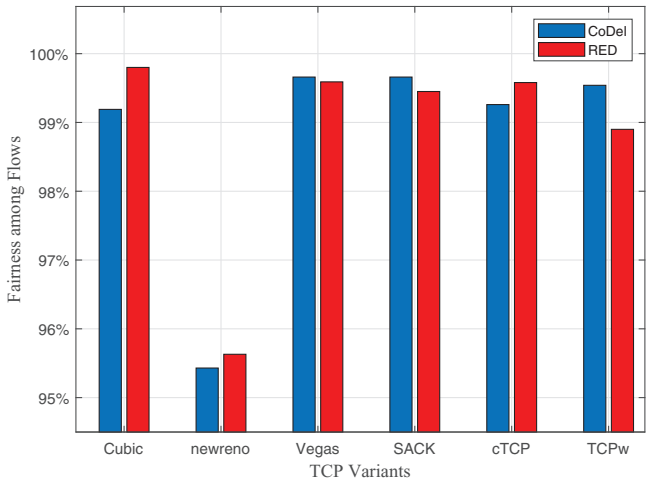

FIGURE 18 CoDel versus RED: Fairness in bandwidth sharing among multiple flows for all TCP variants (high congestion: 14 flows; large packet size: 1460 bytes)

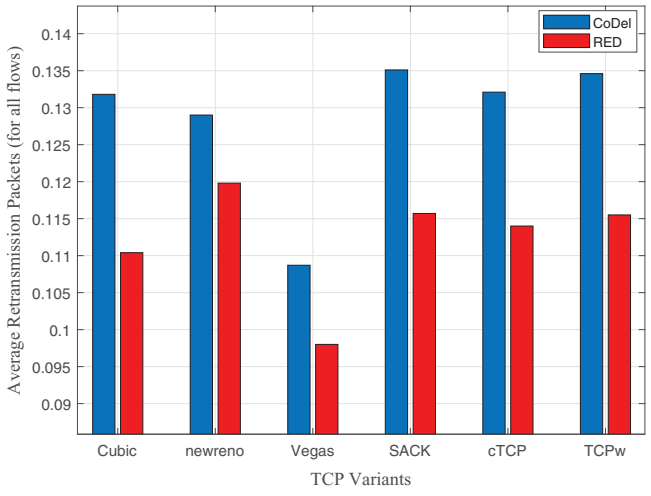

FIGURE 19 CoDel versus RED: Average retransmitted packets for all TCP variants (high congestion: 14 flows; large packet size: 1460 bytes)

fluctuated owing to frequent and simultaneous packet drops. The overall packet drops for CoDel were greater than those of RED. However, for most TCP variants, packet drops with CoDel did not occur frequently: An example is NewReno. The burst drop rates for the other TCP variants were similar for both AQMs. The burst drop rate in Figure 17 is the average number of packet drops every 5 s. The simulation was repeated with every TCP variant for CoDel and RED. Therefore each figure summarises 120 drop rate samples and several thousand per-packet drops in the enqueued samples.

Sensitivity analysis: Figure 20 shows the overall analysis of the investigated AQMs over TCP variants at HCHP. The three performance metrics (i.e. queuing delay, drop rate, and link utilisation) are plotted with a box plot representing the first (lower part) and third quantiles (upper part) for each metric.

The first and third quantiles for the normalised utilisation of CoDel were better than for RED (for all TCP variants). Moreover, the third quantile for the queuing delay in CoDel (almost 12–14 ms) was lower than that of RED (approximately 20 ms) over all TCP variants. However, the first quantiles of both AQMs were similar. A prominent difference was observed in their average drop rates. For instance, RED with Cubic and TCPw TCPs had a lower drop rate in its first quantile but higher drop rate in the third quantile than when CoDel was used. However, RED with cTCP and SACK TCPs had higher drop rates in the both first and the third quantiles. The drop rates in the first and third quantiles of RED with NewReno and Vegas were lower than those for CoDel. NewReno with both AQMs had high link utilisation at a higher drop rate, which is as expected from the previous analysis. In sum, with respect to utilisation[11] and queuing delay, CoDel outperformed RED, but was inferior to it in terms of drop rate.

[11] The link utilisation of CoDel was mostly above 95% (except with Cubic), whereas that for RED was in between 90% and 95% (except the first quantile of Cubic)

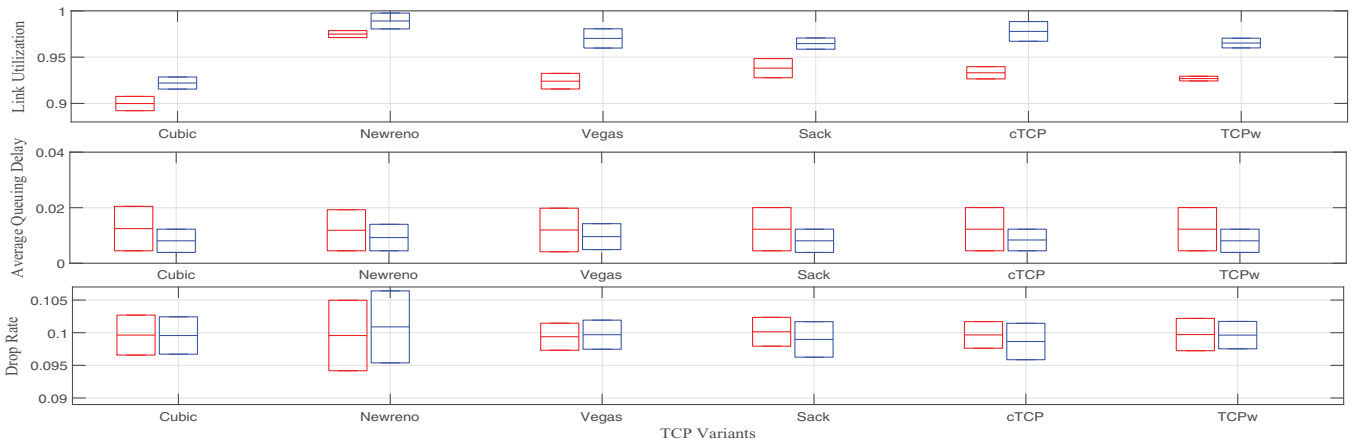

FIGURE 20 CoDel versus RED: Overall performance at high congestion (14 flows) and large packet size (1460 bytes). Link utilisation, drop rate and queuing delay were measured over TCP variants

TABLE 4 Flow utilisation of TCP variants over CoDel

TCP Variant	Flow 1	Flow 2	Flow 3	Flow 4	Flow 5	Flow 6	Flow 7	Flow 8	Flow 9	Flow 10	Flow 11	Flow 12	Flow 13	Flow 14	Jain's Fairness
TCP Cubic	0.993	1.003	1.138	0.946	0.875	1.003	0.923	0.884	0.763	1.043	0.946	0.963	0.958	0.873	0.9919
TCP NewReno	1.311	1.285	1.287	0.892	0.521	1.421	1.197	0.880	1.028	1.433	1.034	1.032	1.058	1.380	0.9543
TCP Vegas	1.192	1.016	1.101	1.015	1.088	1.019	1.027	1.078	0.950	1.075	0.977	0.987	0.991	1.004	0.9966
TCP SACK	1.042	1.109	0.981	1.105	0.986	1.069	1.129	0.994	1.102	1.098	1.033	0.925	1.071	0.977	0.9966
cTCP	1.161	1.063	1.117	1.057	1.116	1.005	0.916	1.101	1.034	1.131	1.044	1.004	0.964	0.818	0.9926
TCPw	1.086	1.022	0.854	1.131	1.044	1.076	1.013	1.021	1.143	1.031	1.006	0.948	1.075	1.068	0.9954

Flow utilisation and retransmission packets: This metric is related to the performance of the TCP, and reflects the utilisation of each flow and the fairness of bandwidth sharing among flows. Packet retransmission occurs when there is either a missing packet, or a particular packet has taken more than usual to acknowledge (timeout).

Figure 18 illustrates fairness among TCP flows using Jain's Fairness Index for high congestion (14 flows) and large payload sizes (1460 bytes) scenario. The Cubic, NewReno and cTCP with RED are more fairer than CoDel counterpart, however, the remaining TCP variants with CoDel has more fairness than that of RED. The detailed quantitative analysis is given in Tables 4 and 5, respectively, where each column shows flow utilisation (as discussed in Section 5.2.5).

Figure 19 shows the average packet retransmission for the simulation time—100 s—under high congestion (14 flows) with a large packet size (1460 bytes). CoDel dropped more packets than RED, which with any TCP variant retransmitted fewer packets. This is detailed in the Tables 6 and 7. Each column shows the number of retransmitted packets per TCP flow.

TABLE 5 Flow utilisation of TCP variants over RED

TCP Variant	Flow 1	Flow 2	Flow 3	Flow 4	Flow 5	Flow 6	Flow 7	Flow 8	Flow 9	Flow 10	Flow 11	Flow 12	Flow 13	Flow 14	Jain's Fairness
TCP Cubic	1.078	1.015	1.062	1.051	0.968	1.072	1.024	1.100	1.061	0.947	1.019	0.993	1.114	1.064	0.9980
TCP NewReno	1.360	1.449	1.196	1.104	0.888	1.189	1.023	1.205	1.424	1.158	0.856	0.543	1.208	0.927	0.9563
TCP Vegas	1.188	0.954	0.993	1.014	0.989	1.023	1.062	0.933	1.064	1.061	0.977	0.978	1.096	0.959	0.9959
TCP SACK	1.198	1.118	1.066	0.967	1.070	1.226	1.157	1.036	1.059	1.085	0.975	0.947	1.030	1.094	0.9945
cTCP	1.010	1.066	1.132	0.994	1.130	1.085	0.905	1.104	1.052	0.952	1.012	1.144	1.029	1.025	0.9958
TCPw	1.275	1.032	0.952	1.052	1.059	0.787	1.140	0.988	1.035	1.063	1.176	1.081	0.951	1.013	0.9890

TABLE 6 Retransmission packets of TCP variants over CoDel

TCP Variant	Rtx 1	Rtx 2	Rtx 3	Rtx 4	Rtx 5	Rtx 6	Rtx 7	Rtx 8	Rtx 9	Rtx 10	Rtx 11	Rtx 12	Rtx 13	Rtx 14	*Mean*(*Rtx*)
TCP Cubic	0.1223	0.1208	0.1267	0.1321	0.1218	0.1431	0.1409	0.1412	0.123	0.1389	0.1424	0.1332	0.1424	0.1168	0.1318
TCP NewReno	0.1181	0.1117	0.1207	0.1281	0.1303	0.1195	0.1433	0.1278	0.1406	0.1437	0.1154	0.14	0.1397	0.1273	0.1290
TCP Vegas	0.0974	0.0953	0.1024	0.1139	0.1088	0.0944	0.1144	0.1149	0.1228	0.1136	0.1053	0.1066	0.1121	0.1194	0.1087
TCP SACK	0.1331	0.1321	0.1301	0.1222	0.1305	0.1366	0.1456	0.132	0.1354	0.1352	0.1519	0.1292	0.1345	0.143	0.1351
cTCP	0.126	0.1282	0.1243	0.1401	0.1323	0.1302	0.1221	0.1458	0.1432	0.1437	0.1289	0.1288	0.1259	0.1293	0.1321
TCPw	0.1251	0.1315	0.1378	0.1264	0.1262	0.1283	0.145	0.1464	0.1374	0.1342	0.1442	0.127	0.1294	0.1449	0.1346

TABLE 7 Retransmission packets of TCP variants over RED

TCP Variant	Rtx 1	Rtx 2	Rtx 3	Rtx 4	Rtx 5	Rtx 6	Rtx 7	Rtx 8	Rtx 9	Rtx 10	Rtx 11	Rtx 12	Rtx 13	Rtx 14	*Mean*(*Rtx*)
TCP Cubic	0.0954	0.0997	0.1259	0.1133	0.1135	0.1154	0.1039	0.1099	0.105	0.1073	0.1155	0.1176	0.1117	0.1121	0.1104
TCP NewReno	0.1161	0.1132	0.1135	0.1187	0.1295	0.1031	0.1263	0.1158	0.1047	0.135	0.1261	0.1224	0.14	0.1128	0.1198
TCP Vegas	0.0893	0.108	0.1035	0.0885	0.0856	0.0991	0.1065	0.0995	0.097	0.0964	0.0949	0.1019	0.0953	0.1065	0.0980
TCP SACK	0.0984	0.1192	0.1072	0.1139	0.121	0.1121	0.1229	0.1177	0.1199	0.1165	0.1198	0.1125	0.1151	0.1233	0.1157
cTCP	0.113	0.1073	0.1175	0.1356	0.1073	0.1203	0.1142	0.1064	0.1159	0.1079	0.1118	0.1099	0.1154	0.113	0.1140
TCPw	0.1135	0.1118	0.1153	0.1168	0.1087	0.1014	0.1251	0.1203	0.1143	0.1223	0.1161	0.1292	0.1154	0.1068	0.1155

6 | CONCLUSION

In this paper, we explored queuing delay, link utilisation, and drop rate in two AQM schemes, CoDel and RED, over six types of TCP variants (i.e. TCP Cubic, TCP NewReno, TCP SACK, TCP Vegas, Compound TCP, and TCP Westwood). We chose a variety of TCPs intended for different applications. We used three scenarios to evaluate the performance of CoDel as compared with RED: variable congestion and fixed payload (VCFP), variable payload and fixed congestion (VPFC), and high congestion and high payload (HCHP). We first examined the AQMs in terms of the performance metrics, where CoDel outperformed RED in terms of average queuing delay and normalised link utilisation (except in VCFP, where RED over cubic TCP had better utilisation at a high level of congestion). However, in terms of drop rate, RED dominated the CoDel scheme in several cases. RED in VCFP had a lower drop rate than CoDel; similarly, in VPFC, RED's drop rate was smaller than that of CoDel at low and medium congestion; moreover, In HCHP, its drop rate was lower for some TCPs, such as NewReno and Vegas. We also evaluated the performance of the TCPs using CoDel and RED in terms of fairness among flows and retransmission of packets per flow. We used Jain's Fairness Index for per-flow utilisation to show fairness in bandwidth sharing among multiple flows. NewReno, Cubic, and cTCP showed better fairness with RED while the other TCPs showed this with CoDel. We also determined the average retransmitted packets of all flows over the TCP variants with HCHP, which provided an important comparison at each flow. The retransmission of packets in all considered TCPs was lower in the RED scheme than in CoDel.

7 | DISCUSSION AND FUTURE WORKS

Given that AQM can mitigate the bufferbloat problem, We are also interested to include some other AQM schemes such as PIE and variant of CoDel and PIE, that is, FQ-CoDel [67] and FQ-PIE [66] to conduct a more detailed analysis. The PIE scheme is a counterpart of CoDel, which is a light weight scheme and can control the queuing delay very effectively. We have given some of its examples in our GitHub[12] that the readers can further examine on TCP variants to see its behaviour over various congestion.

In the future we intend to use diverse traffic types such as web traffic (PackMime [68]) and video traffic like DASH [69] etc. to further assess the impact of bufferbload and AQM scheme's performance.

In addition the TCP-BBR is a good candidate for mitigating the bufferbloat problem, that can be used with the aforementioned AQM schemes. This can be interesting to tackle the bufferbloat problem both at layer three and four and a lot of performance improvement is expected to be achieved. We have put our code in the GitHub for the readers to use it for their future work.

ACKNOWLEDGEMENT

This work was supported by a research grant from Inha University.

ORCID

Salman Muhammad https://orcid.org/0000-0003-4754-805X
Touseef Javed Chaudhery https://orcid.org/0000-0003-1443-3571

[12] https://github.com/salmanpolito/Bufferbloat-AQM-performance-over-TCP-variants-

REFERENCES

1. Lan, K.C., Heidemann, J.: A measurement study of correlations of internet flow characteristics. Comp. Netw. 1(16), 46–62 (2006)
2. Alfredsson, S., et al.: Impact of TCP congestion control on bufferbloat in cellular networks. In: 2013 IEEE 14th International Symposium on A World of Wireless, Mobile and Multimedia Networks (WoWMoM), vol. 6, pp. 1–7. IEEE, Piscataway (2013)
3. Gettys, J., Nichols, K.: Bufferbloat: Dark buffers in the internet. Queue 11(29), 40–54 (2011)
4. Chirichella, C., Rossi, D. To the Moon and back: Are internet bufferbloat delays really that large? In: 2013 Proceedings IEEE INFOCOM, vol. 4, pp. 3297–3302. IEEE, Piscataway (2013)
5. Dischinger, M., et al.: Characterizing residential broadband networks. In: Proceedings of the 7th ACM SIGCOMM Conference on Internet Measurement, vol. 10, pp. 43–56. ACM, New York (2007)
6. Sundaresan, S., et al.: Broadband internet performance: A view from the gateway. ACM SIGCOMM Comp. Commun. Rev. 8(15), 134–45 (2011)
7. Kreibich, C., et al.: Illuminating the edge network. In: Proceedings of the 10th ACM SIGCOMM Conference on Internet Measurement, vol. 11, pp. 246–259. ACM, New York (2010)
8. Jiang, H., et al.: Tackling bufferbloat in 3G/4G networks. In: Proceedings of the 2012 Internet Measurement Conference, vol. 11, pp. 329–342. ACM, New York (2012)
9. Dong, P., et al.: Receiver-side TCP countermeasure in cellular networks. Sensors 19(12), 2791 (2019)
10. Jude, M., et al.: Throughput stability and flow fairness enhancement of TCP traffic in multi-hop wireless networks. Wirel. Netw. 26, 4689–4704 (2020)
11. Showail, A., et al.: An empirical evaluation of bufferbloat in IEEE 802.11 n wireless networks. In: 2014 IEEE Wireless Communications and Networking Conference (WCNC), vol. 4, pp. 3088–3093. IEEE, Piscataway (2014)
12. Ferlin-Oliveira, S., et al.: Tackling the challenge of bufferbloat in multi-path transport over heterogeneous wireless networks. In: 2014 IEEE 22nd International Symposium of Quality of Service (IWQoS), vol. 5, pp. 123–128. IEEE, Piscataway (2014)
13. Hien, D.T., et al.: A software defined networking approach for guaranteeing delay in Wi-Fi networks. In: Proceedings of the Tenth International Symposium on Information and Communication Technology, vol. 12, pp. 191–196. ACM, New York (2019)
14. Cardwell, N., et al.: BBR: Congestion-based congestion control. Queue 10(1), 20–53 (2016)
15. Pan, R., et al.: PIE: A lightweight control scheme to address the bufferbloat problem. In: 2013 IEEE 14th International Conference on High Performance Switching and Routing (HPSR), vol. 7, pp. 148–155. IEEE, Piscataway (2013)
16. May, M., et al.: Reasons not to deploy RED. In: 1999 Seventh International Workshop on Quality of Service, vol. 5, pp. 260–262. IEEE, Piscataway (1999)
17. Misra, V., et al.: Fluid-based analysis of a network of AQM routers supporting TCP flows with an application to RED. In: Proceedings of the Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication, vol. 8, pp. 151–160. ACM, New York (2000)
18. Ott, T.J., et al.: Sred: Stabilized red. In: Proceedings of IEEE INFOCOM'99: Conference on Computer Communications—Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies, vol. 3, pp. 1346–1355. IEEE, Piscataway (1999)
19. Nagle, J.: On packet switches with infinite storage. IEEE Trans. Commun. 4, 435–438 (1987)
20. Abbasloo, S., et al.: C2TCP: A flexible cellular TCP to meet stringent delay requirements. IEEE J. Sel. Areas Commun. 37(4), 918–932 (2019)
21. Floyd, S., Jacobson, V.: Random early detection gateways for congestion avoidance. IEEE/ACM Trans. Netw. 8, 397–413 (1993)
22. Nichols, K., et al.: Controlled delay active queue management. RFC 8289 1, 1–25 (2018)
23. Misra, S., et al.: Random early detection for congestion avoidance in wired networks: A discretized pursuit learning-automata-like solution. IEEE Trans. Syst. Man Cybern. B (Cybernetics) 40(1), 66–76 (2010), 2
24. Abdel-Jaber, H.: An exponential active queue management method based on random early detection. J. Comp. Netw. Commun. 2020, 1–11 (2020)
25. Zhou, K., Li, K.L.V.: Nonlinear RED: A simple yet efficient active queue management scheme. Comp. Netw. 50(18), 3784–3794 (2006)
26. Abbasov, B., Korukoglu, S.: Effective RED: An algorithm to improve RED's performance by reducing packet loss rate. J. Netw. Comp. Appl. 32(3), 703–709 (2009)
27. Nichols, K., Jacobson, V.: Controlling queue delay. Commun. ACM 7(1), 42–50 (2012)
28. Grazia, C.A., et al.: A cross-comparison between TCP and AQM algorithms: Which is the best couple for congestion control? In: 2017 14th International Conference on Telecommunications (ConTEL), vol. 6, pp. 75–82. IEEE, Piscataway (2017)
29. Khademi, N., et al.: The new AQM kids on the block: An experimental evaluation of CoDel and PIE. In: 2014 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS), vol. 4, pp. 85–90. IEEE, Piscataway (2014)
30. Järvinen, I., Kojo, M.: Evaluating CoDel, PIE, and HRED AQM techniques with load transients. In: 39th Annual IEEE Conference on Local Computer Networks, vol. 9, pp. 159–167. IEEE, Piscataway (2014)
31. Hamadneh, N., et al.: HRED, an active queue management algorithm for TCP congestion control. Recent Pat. Comp. Sci. 12(3), 212–217 (2019)
32. Vyakaranal, S.B., Jayalaxmi, G.N.: Performance evaluation of TCP using AQM schemes for congestion control. In: 2018 Second International Conference on Advances in Electronics, Computers and Communications (ICAECC), pp. 1–6. IEEE, Piscataway (2018)
33. Vyakaranal, S.B., Naragund, J.G.: Performance evaluation of TCP using AQM schemes for congestion control. In: 2018 Second International Conference on Advances in Electronics, Computers and Communications (ICAECC), vol. 2, pp. 1–6. IEEE, Piscataway (2018)
34. Grazia, C., et al.: Transmission control protocol and active queue management together against congestion: Cross-comparison through simulations. SIMULATION 95(10), 979–993 (2019)
35. Palmei, J., et al.: Design and evaluation of COBALT queue discipline. In: 2019 IEEE International Symposium on Local and Metropolitan Area Networks (LANMAN), vol. 7, pp. 1–6. IEEE, Piscatawa (2019)
36. Okokpujie, K., et al.: Comparative analysis of the performance of various active queue management techniques to varying wireless network conditions. Int. J. Electr. Comp. Eng. 9(1), 359–368 (2019)
37. Dzivhani, M., Khmaies, O.: Performance evaluation of TCP congestion control algorithms for wired networks using NS-3 simulator. In: IEEE AFRICON, pp. 1–7. IEEE, Piscataway (2019)
38. Feng, W.C., et al.: The BLUE active queue management algorithms. IEEE/ACM Trans. Netw. 11(7), 513–528 (2002)
39. Ye, J., Leung, K.C.: Adaptive and stable delay control for combating bufferbloat: Theory and algorithms. IEEE Syst. J. 7(31), 1285–1296 (2019)
40. Postel, J., Reynolds, J.: File transfer protocol. RFC 765 10, 1–69 (1985)
41. Postel, J.: Transmission control protocol. RFC 793 9, 1–89 (1981)
42. Parvez, N., et al.: An analytic throughput model for TCP NewReno. IEEE/ACM Trans. Netw. 11(3), 448–461 (2009)
43. Brakmo, L.S., Peterson, L.L.: TCP Vegas: End to end congestion avoidance on a global internet. IEEE J. Sel. Areas Commun. 10, 1465–1480 (1995)
44. Song, K.T., et al.: Compound TCP: A scalable and TCP-friendly congestion control for high-speed networks. PFLDnet 2006 2, 1–8 (2006)
45. Mathis, M., et al. TCP selective acknowledgment options. RFC 2018 10(1), 1–12 (1996)
46. Xu, L., et al.: Cubic for fast long-distance networks. RFC 8312 2, 1–18 (2018)
47. Mascolo, S., et al.: TCP westwood: Bandwidth estimation for enhanced transport over wireless links. In: Proceedings of the 7th Annual International Conference on Mobile Computing and Networking, vol. 7, pp. 287–297. ACM, New York (2001)
48. Hoe, J.C.: Improving the start-up behavior of a congestion control scheme for TCP. ACM SIGCOMM Comp. Commun. Rev. 8(28), 270–280 (1996)
49. Floyd, S.: TCP and successive fast retransmits. Technical report, 5(22), 1–4 (1995)

50. Mathis, M., et al.: TCP selective acknowledgment options. RFC 2018 10(1), 1–12 (1996)
51. Mo, J., et al.: Analysis and comparison of TCP Reno and Vegas. In: Proceedings of IEEE INFOCOM'99: Conference on Computer Communications—Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies, vol. 3, pp. 1556–1563. IEEE, Piscataway (1999)
52. Brakmo, L.S., et al.: TCP Vegas: New techniques for congestion detection and avoidance. In: Proceedings of the Conference on Communications Architectures, Protocols and Applications, vol. 10, pp. 24–35. ACM, New York (1994)
53. Floyd, S.: HighSpeed TCP for large congestion windows. RFC 3649 12, 1–34 (2003)
54. Postei, J.: Transmission control protocol-DARPA internet program protocol specification. RFC 793 9, 1–85 (1981)
55. Kelly, T.: Scalable TCP: Improving performance in highspeed wide area networks'. ACM SIGCOMM Comp. Commun. Rev. 4(1), 83–91 (2003)
56. Floyd, S., et al.: The NewReno modification to TCP's fast recovery algorithm. RFC 3782 4, 1–19 (2004)
57. Blanton, E., et al.: A conservative selective acknowledgment (SACK)-based loss recovery algorithm for TCP. RFC 3517 4(1), 1–13 (2003)
58. Handley, M., et al.: TCP friendly rate control (TFRC): Protocol specification. RFC 3448 1(1), 1–24 (2003)
59. Stewart, R., et al.: Stream control transmission protocol. RFC 4960 9, 1–152 (2007).
60. Xu, L., et al.: Binary increase congestion control (BIC) for fast long-distance networks. In: IEEE INFOCOM 2004, vol. 3, pp. 2514–2524. IEEE, Piscataway (2004)
61. Chaudhery, T.J.: Performance evaluation of CoDel queue mechanism and TFRC transport protocol when using VoIP flows. In: 2017 International Conference on Frontiers of Information Technology (FIT) 2017, vol. 12, pp. 1–6. IEEE, Piscataway (2018)
62. Hashem, E.S.: Analysis of random drop for gateway congestion control. Technical Report, pp. 1–108. Lab for Computer Science, Massachusetts Institute of Technology, Cambridge (1989)
63. Feng, W.C., et al.: A self-configuring RED gateway. In: Proceedings of IEEE INFOCOM'99: Conference on Computer Communications—Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies, vol. 3, pp. 1320–1328. IEEE, Piscataway (1999)
64. Floyd, S., et al.: Adaptive RED: An algorithm for increasing the robustness of RED's active queue management, pp. 1–12. International Computer Science Institute, Berkeley (2001)
65. Issariyakul, T., Hossain, E.: Introduction to Network Simulator 2 (NS2), pp. 1–18. Springer, Boston (2009)
66. Ramakrishnan, G., et al.: FQ-PIE queue discipline in the Linux kernel: Design, implementation and challenges. In: 2019 IEEE 44th LCN Symposium on Emerging Topics in Networking (LCN Symposium), vol. 10, pp. 117–124. IEEE, Piscataway (2019)
67. Rao, V.P., et al.: Analysis of sfqCoDel for active queue management. In: The Fifth International Conference on the Applications of Digital Information and Web Technologies (ICADIWT 2014), vol. 2, pp. 262–267. IEEE, Piscataway (2014)
68. Cao, J., et al.: PackMime: An internet traffic generator. In: National Institute of Statistical Sciences Affiliates Workshop on Modeling and Analysis of Network Data, vol. 3. National Institute of Statistical Sciences, Bath (2001)
69. Khan, K., Goodridge, W.: B-DASH: Broadcast-based dynamic adaptive streaming over HTTP. Int. J. Auton. Adapt. Commun. Syst. 12, 50–74 (2019)

How to cite this article: Muhammad S, TJ Chaudhery, Y Noh. Study on performance of AQM schemes over TCP variants in different network environments. *IET Commun.* 2021;15:93–111.
https://doi.org/10.1049/cmu2.12061