Journal of Advanced Research in Applied Sciences and Engineering Technology

Journal homepage:
https://semarakilmu.com.my/journals/index.php/applied_sciences_eng_tech/index
ISSN: 2462-1943

Analysing TCP Traffic Congestion Algorithms for Wired Links Based on NS3

Zhou Weichen[1], Azana Hafizah Mohd Aman[1], Zainab Senan Attarbashi[2,*], Wan Muhammad Hazwan Azamuddin[1], Aymen Dheyaa Khaleel[3]

[1] Center of Cyber Security, Faculty of Information Science and Technology, University Kebangsaan Malaysia, 43600 Bangi, Selangor, Malaysia
[2] Faculty of Information & Communication Technology, International Islamic University Malaysia, 53100 Gombak, Selangor, Malaysia
[3] Department of Computer Engineering, College of Engineering, Al-Iraqia University, Baghdad Governorate, Iraq

Keywords:
Transmission control protocol; congestion control algorithms; Reno; NewReno; Cubic; NS3

ABSTRACT

An essential component of the Internet Protocol Suite is Transmission Control Protocol (TCP). In this article, the effectiveness of three distinct TCP congestion control algorithms, namely Reno, NewReno, and Cubic, in the context of wired lines has been examined. In a series of simulations, we changed the congestion control technique while maintaining the other parameters constantly using Network Simulator (NS3). With its distinctive fast recovery mechanism, TCP Reno has shown a strong ability to recover from packet losses quickly, cutting down on the time needed to go back to the highest attainable throughput. The TCP congestion control technique selected can have a big impact on a network's performance, notably in terms of throughput and stability. While our findings demonstrate major differences between the examined algorithms.

1. Introduction

The Transmission Control Protocol (TCP) [1], a vital part of the Internet Protocol Suite, functions at the transport layer. It ensures the accurate, sequenced, and error-verified exchange of data streams between applications operating on hosts that interact via an IP network [2-4]. TCP ensures that data sent from one end of a connection reaches the other end without errors and in the correct order [5]. TCP establishes a connection using a three-way handshake mechanism [6] before data transmission begins, and it employs an acknowledgment mechanism to confirm the receipt of packets. The protocol also implements flow control to match the sending rate to the receiving capacity of the receiver and congestion control to adapt the sending rate to network conditions [7]. Congestion in a TCP network occurs when the total demand for network resources exceeds the available capacity [8]. It is a state of excessive buffer occupation and network load, causing a degradation of network performance. The major symptoms of congestion include an increase in packet loss rate and round-trip time (RTT) [9]. If not well controlled, congestion can lead to a scenario

[*] *Corresponding author.*
E-mail address: zainab_senan@iium.edu.my

https://doi.org/10.37934/araset.58.1.242251

known as congestion collapse, where the network transmits many packets but delivers few useful packets, causing a decline in network performance.

To avert network congestion, TCP introduces a collection of control measures. The fundamental congestion control for TCP was suggested by V. Jacobson in his 1988 paper [10], comprising slow start and congestion avoidance mechanisms. Subsequently, the TCP Reno version incorporated fast retransmit and fast recovery. The TCP NewReno modification further enhanced the fast recovery approach [11]. The essential concept behind TCP congestion control is the use of a congestion window (cwnd) as a regulatory tool. TCP also utilizes a receiver-announced window (Receive Window, rwnd) for flow control. The size of the window value represents the maximum amount of data segments that can be sent but have not yet received an acknowledgment (ACK). Clearly, the larger the window, the faster the data can be sent, but this also increases the likelihood of network congestion. If the window value is set to 1, it simplifies to a stop-and-wait protocol. For each piece of data sent, the sender must wait for an acknowledgment from the receiver before sending the next data packet, which obviously results in low data transmission efficiency. The TCP congestion control algorithm is crafted to maintain equilibrium between these two elements, choosing the ideal cwnd value that boosts network throughput without bringing congestion. This research aims to analyse and compare the performance of TCP Reno, NewReno, and Cubic [12] on wired links using NS3 [13]. In the rest of this paper, we will first discuss the technology that used to implement congestion control. Following this, we will present our experimental setup, explaining the network topology, traffic patterns, types of links and the results of our simulations. Finally, the conclusion part will summarize our simulation works to understanding of TCP congestion control algorithms.

2. Congestion Control Mechanism

2.1 Slow Start

Slow Start is the initial state of a TCP connection [14,15]. It is used to prevent a connection from sending more data than the network is capable of handling. As shown in Figure 1, when initial a connection, the sender sets the congestion window size to a small value, typically a maximum segment size (MSS). For every Acknowledgment (ACK) received, the congestion window is increased by one MSS. This leads to an exponential growth in the congestion window size. To keep the cwnd from expanding too much and leading to network congestion, a state variable named slow start threshold (ssthresh) is also established. When cwnd is less than ssthresh, the state stays on slow start. When cwnd is more than ssthresh, the congestion avoidance algorithm is implemented instead. When cwnd is equal to ssthresh, these two algorithms can be used. Until it reaches a threshold, it will be called the slow start threshold (SSThresh), or until packet loss is detected.

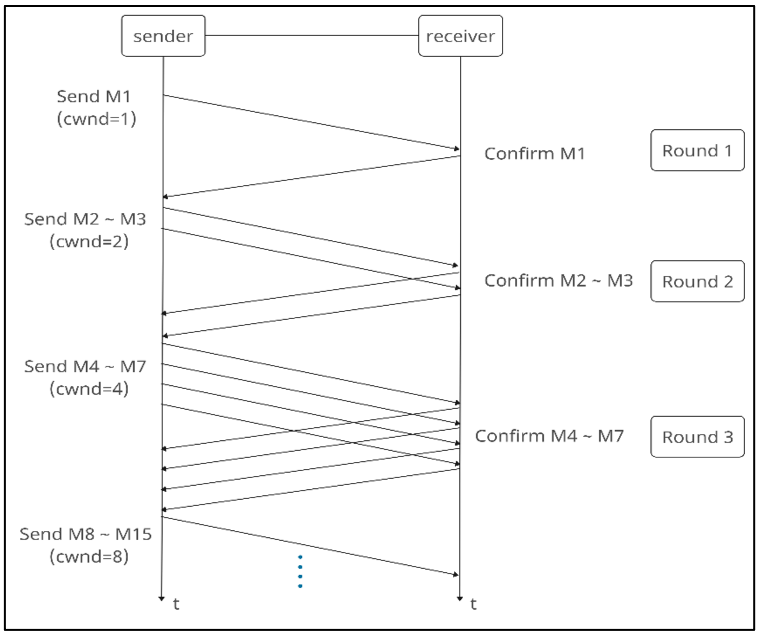

Fig. 1. Process of slow start

2.2 Congestion Avoidance

Once the congestion window size has reached the SSThresh, the TCP sender enters the Congestion Avoidance phase. In this phase, the congestion window is increased more conservatively to probe for additional network capacity. Specifically, for every round-trip time (RTT), the congestion window is increased by approximately 1 MSS. This results in a linear increase in the congestion window size, thereby avoiding a rapid surge in network traffic that could lead to congestion. If packet loss is detected during this phase, it's an indication of network congestion, and the TCP sender reduces the SSThresh and the congestion window size and re-enters the Slow Start phase.

2.3 Fast Retransmit

Fast Retransmit is a mechanism in TCP's congestion control suite used for accelerating the retransmission of lost packets [16]. At first, TCP uses timers to detect lost packets. If an acknowledgment (ACK) for a packet is not received within a specified timeout period, the packet is considered lost and retransmitted. However, this method may often lead to unnecessary delays in loss detection. To mitigate this delay, TCP uses the Fast Retransmit algorithm, which leverages duplicate acknowledgments (dupACKs) to quickly detect packet losses. As shown in Figure 2, when a packet is lost, out-of-order packets prompt the receiver to send duplicate ACKs. The sender receives three duplicate ACKs, infers that a packet has been lost then immediately retransmits the lost packet without waiting for the retransmission timer to expire. This is a more responsive method than timer-based retransmission.

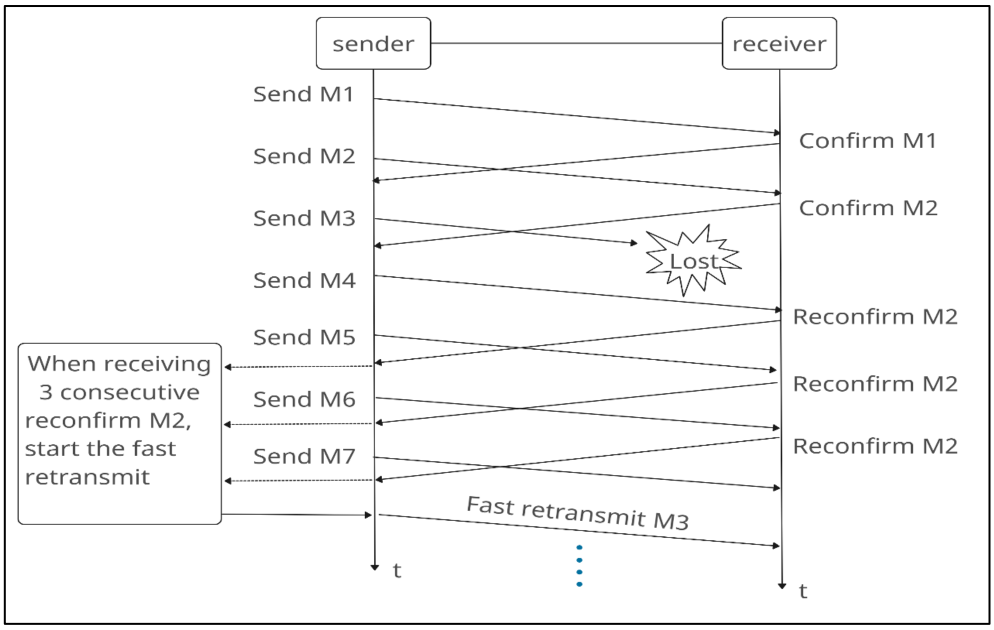

Fig. 2. Fast retransmit

2.4 Fast Recovery

Fast Recovery is a mechanism works with fast retransmit to address packet loss swiftly and maintain high throughput rates [17]. During Fast Recovery, the TCP sender does not reset the cwnd to the initial size. Instead, it deflates the cwnd by a certain amount but keeps it relatively high. This mechanism allows TCP to continue transmitting new packets at a higher rate instead of throttling down to the minimum cwnd. Once entering fast recovery, the cwnd is typically cut in half, and the slow start threshold (ssthresh) is adjusted to this new value. However, TCP continues to transmit new packets, maintaining a level of throughput more consistent with the estimated network capacity. Once the sender receives an ACK for the retransmitted packet, it ends the fast recovery phase. The cwnd is then set to the ssthresh value, and the sender re-enters the Congestion Avoidance phase, where the cwnd is gradually increased until a new loss event occurs or until the cwnd reaches the receiver's advertised window size.

3. Simulation and Results
3.1 Experiment Setup

This simulation is set up to investigate the congestion control algorithm in TCP protocol. The network topology consists of 2 routers and 2 nodes (user A and user B) connected in a linear fashion as shown in Figure 3.

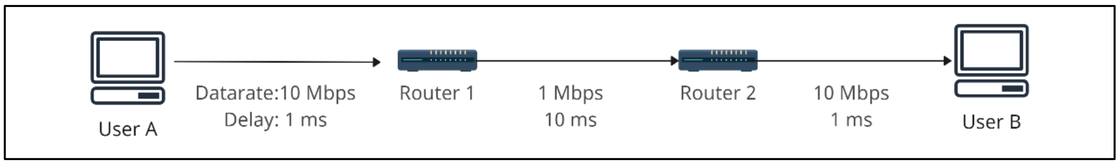

Fig. 3. Network topology

Each node is connected to a router through a point-to-point link with a data rate of 10 Mbps and delay of 1 ms. The two routers are also connected with a point-to-point link, but with a data rate of 1 Mbps and a delay of 10 ms. A bulk send application is installed at the source node (user A), and a packet sink is installed at the destination node (user B). In this simulation, TCP congestion control algorithm is the only variable to change. Other configurations are listed in the Table 1.

Table 1
Configuration

Items	Details
Operating system	Ubuntu 22.04
NS3 version	3.36.1
Network model	Wired
Bottleneck of topology	1 Mbps bottleneck
Buffer discipline	First in first out
Initial cwnd	10

3.2 TCP Reno

Transmission Control Protocol (TCP) Reno was designed to improve upon its predecessor, TCP Tahoe, with the introduction of more refined congestion control mechanisms [18-20]. TCP Reno implements the same slow-start, congestion avoidance, and fast retransmit algorithms as TCP Tahoe. However, where it stands out is its introduction of a new mechanism called "fast recovery." In contrast to TCP Tahoe, which responds to packet loss events by dropping its congestion window (cwnd) size to one segment and entering slow-start mode, TCP Reno's fast recovery algorithm allows it to keep the cwnd size at half of its current value this is shown in Figure 4. This allows TCP Reno to recover more quickly from packet losses, reducing the time taken to return to the maximum achievable throughput.

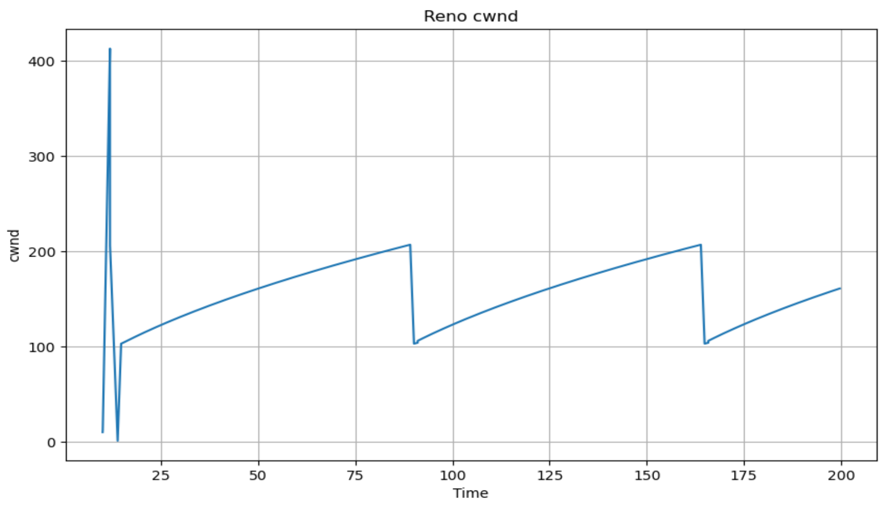

Fig. 4. Reno cwnd

In this simulation result presented in Figure 5, when the connection starts, the TCP Reno will enter slow start phase, where it doubles the cwnd every round trip time (RTT) until a packet loss is detected or reach the ssthresh. At this point, it will enter the congestion avoidance phase, increasing it one segment every RTT (Eq. (1)). When a packet loss is detected during the congestion avoidance phase, the cwnd is halved (Eq. (2)) and enter into the fast retransmit and fast recovery status.

$$cwnd \mathrel{+}= 1/cwnd \tag{1}$$

$$cwnd \mathrel{-}= \frac{cwnd}{2} \tag{2}$$

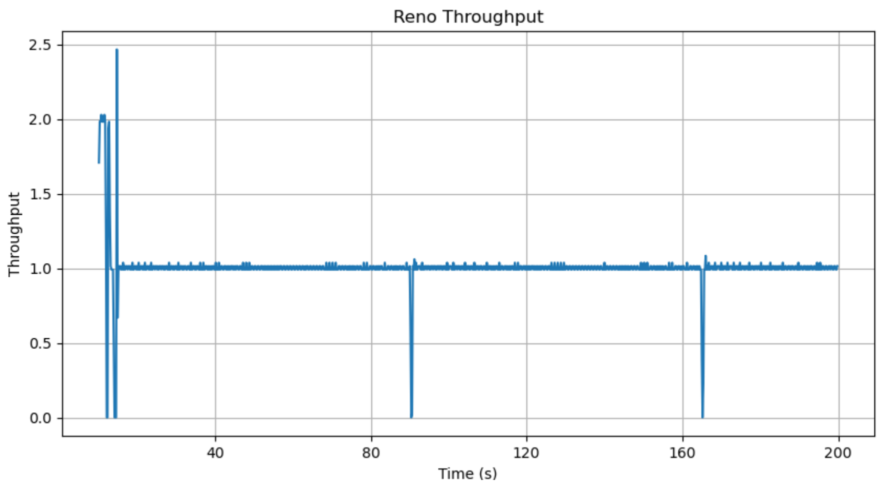

Fig. 5. Reno throughput

This simulation shows that the throughput of the TCP Reno connection shown in Figure 6 fluctuates. It increases during the slow start and drops whenever a packet loss is detected. When in the congestion avoidance process, the throughput stays stable around 1 Mbps.

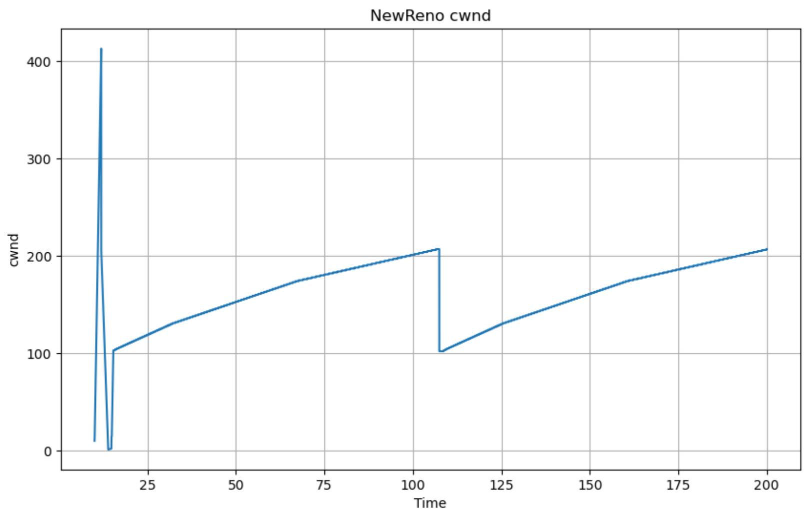

Fig. 6. NewReno cwnd

3.3 TCP NewReno

TCP NewReno is an enhancement of the TCP Reno congestion control algorithm [21-23]. In TCP Reno, after a packet loss, the protocol reduces the congestion window size and starts the fast recovery phase. However, it exits this fast recovery phase as soon as it receives an acknowledgement, which may not account for multiple lost packets. This can lead to additional round-trip times to detect and recover from the loss of these packets. TCP NewReno improves upon this by remaining in the fast recovery phase until all lost packets within a window of data are acknowledged. This way, it can recover from multiple losses within a window without needing additional round-trip times, thereby improving efficiency and throughput. In this simulation, result shows that NewReno has better performance on stability of cwnd as shown in Figure 6 and throughput as shown in Figure 7.

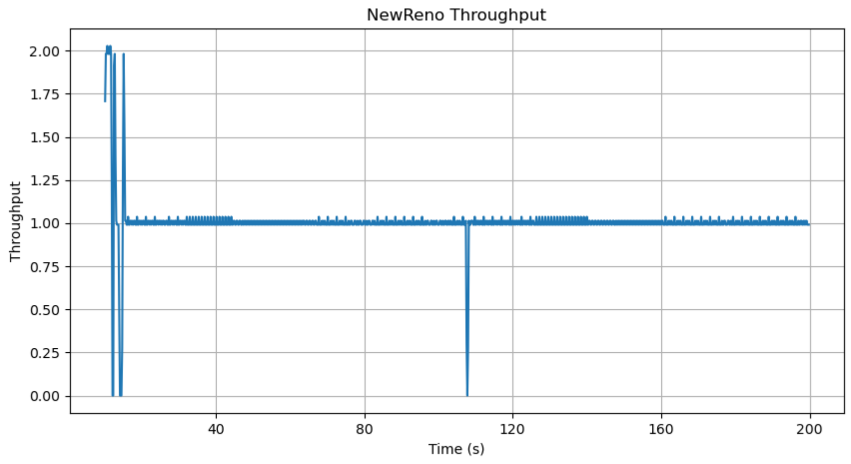

Fig. 7. NewReno throughput

3.4 TCP Cubic

TCP Cubic diverges from the traditional TCP protocols, designed to make more effective use of available bandwidth, particularly on high-speed, high-latency networks [24-26]. It is the next generation version of BIC-TCP, smooth the congestion window growth curve via a cubic function (Eq. (3) and Eq. (4)), enabling it to maintain for a longer period when approaching the previous cwnd maximum. Furthermore, it decouples the growth of cwnd from the Round-Trip Time (RTT) duration, meaning it does not increase cwnd with each Acknowledgement (ACK) received. Instead. This results in a fairer network, where connections with shorter RTTs cannot monopolize the resources of those with longer RTTs. Figure 8 and Figure 9 shows the result of congestion windows and throughput respectively for TCP Cubic. Compared with previous traditional protocols, TCP Cubic performs a higher congestion window.

$$W(t) = C(t-K)^3 + W_{max} \tag{3}$$

$$K = \sqrt[3]{\frac{W_{max} * \beta}{C}} \tag{4}$$

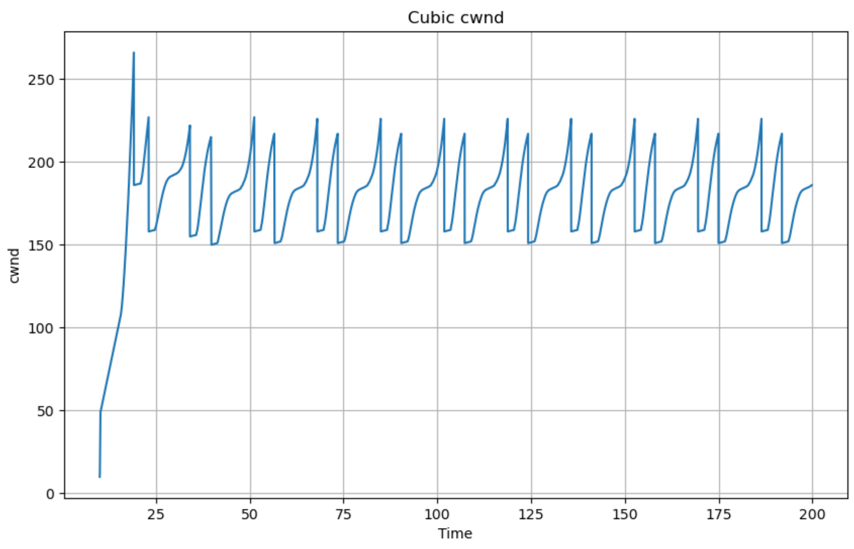

Fig. 8. Cubic cwnd

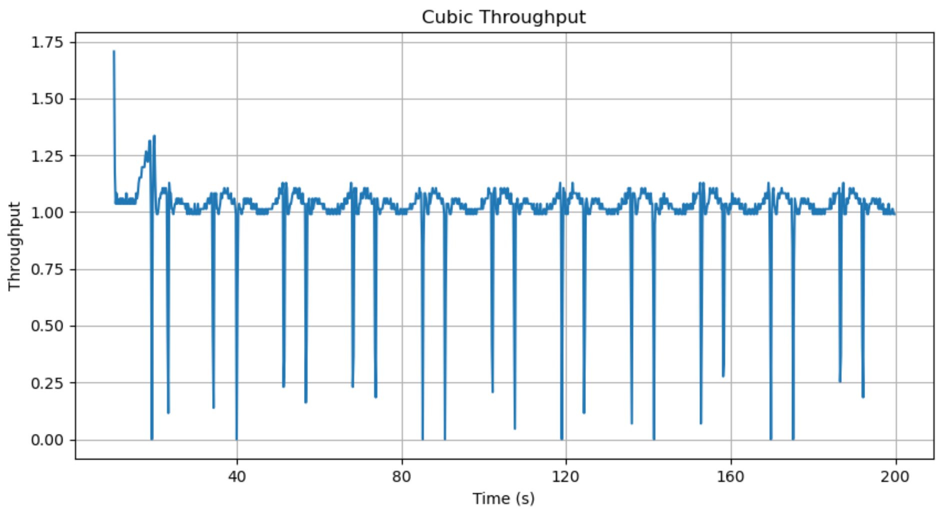

Fig. 9. Cubic throughput

4. Conclusions

In this article, we have analysed the performance of three different TCP congestion control algorithms, namely Reno, NewReno, and Cubic, in the context of wired links. We used NS3 to conduct a series of simulations, altering the congestion control algorithm while keeping other parameters constant. TCP Reno, with its unique fast recovery mechanism, demonstrated a strong ability to quickly recover from packet losses, thereby reducing the time taken to return to maximum achievable throughput. However, it is easily impacted by the buffer of link. When the buffer is small, packet loss may occur on the link before the data reaches the Bandwidth Delay Product (BDP), causing Reno to immediately halve its transmission rate and fail to efficiently utilize the network bandwidth. If the buffer is large, exceeding the BDP, it may enter a "Buffer Bloat" state, characterized by excessively high latency. Each time Reno reduces its speed due to packet loss, it will retransmit data, resulting in previously transmitted data possibly still queued on the link, occupying resources without being effective and eventually being discarded. TCP Cubic showcased a higher congestion window among these three congestion control algorithms. It is good at effectively utilizing available bandwidth. The algorithm's distinctive feature of decoupling congestion window growth from Round-Trip Time, led to a fairer network environment.

In conclusion, the choice of a TCP congestion control algorithm can significantly impact the performance of a network, especially in terms of throughput and stability. While our results show clear distinctions between the tested algorithms. In other words, the best TCP congestion control algorithm is highly dependent on the specific use case and network environment.

Acknowledgement
We would like to thank and acknowledge IIUM Research Management Centre (RMC) for funding the publication fees.

References

[1] Kühlewind, M., and R. Scheffenegger. *TCP Modifications for Congestion Exposure (ConEx)*. No. rfc7786. 2016. https://doi.org/10.17487/RFC7786
[2] Lorincz, Josip, Zvonimir Klarin, and Julije Ožegović. "A comprehensive overview of TCP congestion control in 5G networks: Research challenges and future perspectives." *Sensors* 21, no. 13 (2021): 4510. https://doi.org/10.3390/s21134510
[3] Gomez, Jose, Elie F. Kfoury, Jorge Crichigno, and Gautam Srivastava. "A survey on TCP enhancements using P4-programmable devices." *Computer Networks* 212 (2022): 109030. https://doi.org/10.1016/j.comnet.2022.109030
[4] Thangarasu, Gunasekar, and Kesava Rao Alla. "Compressive Sensing Path for Optimal Data Transmission in Underwater Acoustic Sensor Network." *Journal of Advanced Research in Applied Mechanics* 108, no. 1 (2023): 47-55.
[5] Bonaventure, Olivier, Christoph Paasch, and Gregory Detal. *Use cases and operational experience with multipath TCP*. No. rfc8041. 2017. https://doi.org/10.17487/RFC8041
[6] Katz, D., R. Saluja, and D. Eastlake 3rd. *Three-Way Handshake for IS-IS Point-to-Point Adjacencies*. No. rfc5303. 2008. https://doi.org/10.17487/rfc5303
[7] Fairhurst, Godred, Brian Trammell, and Mirja Kühlewind. *Services provided by IETF transport protocols and congestion control mechanisms*. No. rfc8095. 2017. https://doi.org/10.17487/RFC8095
[8] Lorincz, Josip, Zvonimir Klarin, and Julije Ožegović. "A comprehensive overview of TCP congestion control in 5G networks: Research challenges and future perspectives." *Sensors* 21, no. 13 (2021): 4510. https://doi.org/10.3390/s21134510
[9] Martinsen, P., T. Reddy, D. Wing, and V. Singh. *Measurement of Round-Trip Time and Fractional Loss Using Session Traversal Utilities for NAT (STUN)*. No. rfc7982. 2016. https://doi.org/10.17487/RFC7982
[10] Jacobson, Van. "Congestion avoidance and control." *ACM SIGCOMM computer communication review* 18, no. 4 (1988): 314-329. https://doi.org/10.1145/52325.52356
[11] Henderson, Tom, Sally Floyd, Andrei Gurtov, and Yoshifumi Nishida. *The NewReno modification to TCP's fast recovery algorithm*. No. rfc6582. 2012. https://doi.org/10.17487/rfc6582
[12] Bruhn, Philipp, Mirja Kuehlewind, and Maciej Muehleisen. "Performance and improvements of TCP CUBIC in low-delay cellular networks." *Computer Networks* 224 (2023): 109609. https://doi.org/10.1016/j.comnet.2023.109609
[13] Aldalbahi, Adel, Michael Rahaim, Abdallah Khreishah, Moussa Ayyash, and Thomas DC Little. "Visible light communication module: An open source extension to the ns3 network simulator with real system validation." *IEEE Access* 5 (2017): 22144-22158. https://doi.org/10.1109/ACCESS.2017.2759779
[14] Tlaiss, Ziad, Isabelle Hamchaoui, Isabel Amigo, Alexandre Ferrieux, and Sandrine Vaton. "Troubleshooting enhancement with automated slow-start detection." In *2023 26th Conference on Innovation in Clouds, Internet and Networks and Workshops (ICIN)*, pp. 129-136. IEEE, 2023. https://doi.org/10.1109/ICIN56760.2023.10073485
[15] Lübben, Ralf. "Forecasting TCP's Rate to Speed up Slow Start." *IEEE Open Journal of the Computer Society* 3 (2022): 185-194. https://doi.org/10.1109/OJCS.2022.3208701
[16] Noman, Haeeder M., Ali A. Abdulrazzaq, Marwah M. Kareem, and Adnan Hussein Ali. "Improvement Ivestigation of the TCP Algorithms With Avoiding Network Congestion Based on OPNET." In *IOP Conference Series: Materials Science and Engineering*, vol. 518, no. 5, p. 052025. IOP Publishing, 2019. https://doi.org/10.1088/1757-899X/518/5/052025
[17] Gambhava, Bhavika, and C. K. Bhensdadia. "Mathematical modelling of packet transmission during reclamation period in NewReno TCP and CTCP." *International Journal of Internet Protocol Technology* 16, no. 2 (2023): 110-118. https://doi.org/10.1504/IJIPT.2023.10056778

[18] Abadleh, Ahmad, Aya Tareef, Alaa Btoush, Alaa Mahadeen, Maram M. Al-Mjali, Saqer S. Alja'Afreh, and Anas Ali Alkasasbeh. "Comparative analysis of tcp congestion control methods." In *2022 13th International Conference on Information and Communication Systems (ICICS)*, pp. 474-478. IEEE, 2022. https://doi.org/10.1109/ICICS55353.2022.9811217

[19] Saedi, Taha, and Hosam El-Ocla. "TCP CERL+: Revisiting TCP congestion control in wireless networks with random loss." *Wireless Networks* 27 (2021): 423-440. https://doi.org/10.1007/s11276-020-02459-0

[20] Song, Yeong-Jun, Geon-Hwan Kim, Imtiaz Mahmud, Won-Kyeong Seo, and You-Ze Cho. "Understanding of bbrv2: Evaluation and comparison with bbrv1 congestion control algorithm." *IEEE Access* 9 (2021): 37131-37145. https://doi.org/10.1109/ACCESS.2021.3061696

[21] Abdullah, Saleh M., Mohamed S. Farag, Hatem Abdul-Kader, and Shaban E. Abo Youssef. "Improving the TCP Newreno Congestion Avoidance Algorithm on 5G Networks." *J. Commun.* 18, no. 4 (2023): 228-235. https://doi.org/10.12720/jcm.18.4.228-235

[22] Bennouri, Hajar, and Amine Berqia. "U-NewReno transmission control protocol to improve TCP performance in Underwater Wireless Sensors Networks." *Journal of King Saud University-Computer and Information Sciences* 34, no. 8 (2022): 5746-5758. https://doi.org/10.1016/j.jksuci.2021.08.006

[23] Gambhava, Bhavika, and C. K. Bhensdadia. "Mathematical modelling of packet transmission during reclamation period in NewReno TCP and CTCP." *International Journal of Internet Protocol Technology* 16, no. 2 (2023): 110-118. https://doi.org/10.1504/IJIPT.2023.131296

[24] Bruhn, Philipp, Mirja Kuehlewind, and Maciej Muehleisen. "Performance and Improvements of TCP CUBIC in Low-Delay Cellular Networks." In *2022 IFIP Networking Conference (IFIP Networking)*, pp. 1-9. IEEE, 2022. https://doi.org/10.23919/IFIPNetworking55013.2022.9829781

[25] Mahmud, Imtiaz, Tabassum Lubna, Geon-Hwan Kim, and You-Ze Cho. "BA-MPCUBIC: Bottleneck-aware multipath CUBIC for multipath-TCP." *Sensors* 21, no. 18 (2021): 6289. https://doi.org/10.3390/s21186289

[26] Bruhn, Philipp, Mirja Kuehlewind, and Maciej Muehleisen. "Performance and improvements of TCP CUBIC in low-delay cellular networks." *Computer Networks* 224 (2023): 109609. https://doi.org/10.1016/j.comnet.2023.109609