ASEAN Engineering Journal

Full Paper

END-TO-END DELAY REDUCTION IN SD-WAN BY USING BUCKET SORT ALGORITHM

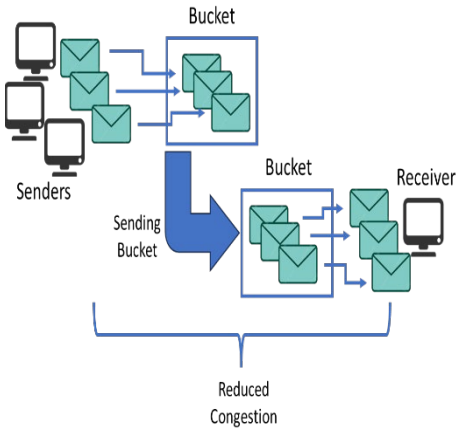
Muhammad Haqeem Mohd Nasir[a], Fairuz Abdullah[b*]

[a]College of Engineering, Universiti Tenaga Nasional, Jalan IKRAM-UNITEN, 43000 Kajang, Selangor Darul Ehsan, Malaysia
[b]Institute of Power Engineering, Universiti Tenaga Nasional, Jalan IKRAM-UNITEN, 43000 Kajang, Selangor Darul Ehsan, Malaysia

Article history
Received
21 August 2025
Received in revised form
16 October 2025
Accepted
22 January 2026
Published online
31 May 2026

*Corresponding author
fairuz@uniten.edu.my

Graphical abstract



Abstract

Software-Defined Networking (SDN) has emerged to address the complexities of traditional Wide Area Networks (WANs) by decoupling the control plane from the data plane, thereby enabling programmability at network nodes and improving their flexibility and manageability. Integrating SDN with WAN infrastructure produces Software-Defined WAN (SD-WAN) which can deliver enhanced Quality of Service (QoS) at reduced cost. However, as communication services evolves, more devices are able to connect to the internet, hence, generate millions of data. This causes higher chance of links to be congested especially at Customer Premise Edge (CPE) that has become the bottleneck of the network. This worsens the congestion and increase the end-to-end delay. Therefore, this paper proposes a proactive congestion control algorithm based on a bucket-sort approach. The technique groups packets that share the same destination network Internet Protocol (IP) address into a single bucket and transmits them as one packet. Then, the proposed algorithm is compared with the traditional loss-based congestion control algorithms which are Reno, New Reno, and Cubic, in terms of end-to-end delay between two CPEs. The results show that the proposed algorithm produces better and more stable end-to-end delay than the traditional loss-based congestion control algorithms.

Keywords: SDN, SD-WAN, Transmission Control Protocol, congestion control, bucket sort

1.0 INTRODUCTION

A Wide Area Network (WAN) is a communication framework designed to interconnect multiple Local Area Networks (LANs) across geographically dispersed locations. To establish these inter-site links, organizations have traditionally relied on leased-capacity services provided by public network operators, utilizing technologies such as Multiprotocol Label Switching (MPLS) has become the dominant choice because it can efficiently provide Quality of Service (QoS) guarantees [1]. However, MPLS imposes higher costs on enterprise users since it inserts an additional forwarding layer between Open System Interconnection (OSI) Layers 2 and 3 [2, 3]. In contrast, VPNs can be deployed over almost any underlying transport, enabling organizations to interconnect sites using inexpensive Internet links, but they lack inherent QoS assurances. To reconcile this cost–performance trade-off, Software-Defined Wide Area Networking (SD-WAN) has been introduced, delivering improved QoS at a lower operational expense. Consequently, SD-WAN is increasingly viewed as a cost-effective alternative for enterprises seeking both performance and cost efficient [1].

SD-WAN is based on the principles of Software-Defined Networking (SDN) that separates the data plane with control plane. This architectural separation streamlines management, improves administrative control, and enables programmability that fosters innovation. SD-WAN's monitoring and programmability capabilities, including traffic-engineering and congestion-management routines, reside in the application plane. Relative to MPLS, SD-WAN reduces operational costs, simplifies WAN configuration, and eases integration with cloud

services [1]. In contrast to VPN, SD-WAN's centralized control logic provides enhanced QoS and greater reliability while preserving comparable access costs. These benefits are realized through centralized controllers in the control plane that continuously monitor and steer the network [1, 4]. SD-WAN functionality is deployed at the network edge via Customer Premises Equipment (CPE) operating in the data plane, as shown in Figure 1. The CPEs execute switching and routing decisions dictated by the control plane [1]. Controller-to-switch interactions are typically implemented over southbound interfaces using OpenFlow derivatives. On the other hand, northbound interface mediate communication between the control and application planes by using Representation State Transfer (REST) Application Programming Interface (API) [5].

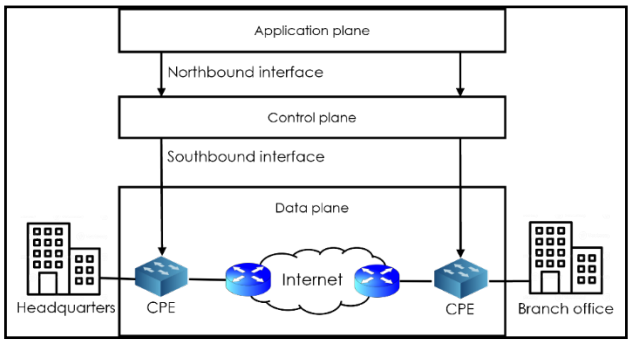


Figure 1 SD-WAN architecture

In addition, SD-WAN is capable of utilizing up-to-date networking technologies such as fifth-generation (5G) networks to enhance overall network throughput [6]. Currently, 5G is a primary transport enabler for a wide set of use cases under the "everything is connected" paradigm. One notable 5G capability is Ultra-Reliable Low-Latency Communication (URLLC), which targets almost instant responsiveness and specifies a 1 milliseconds (ms) round requirement for both downlink and uplink [7, 8]. Examples of the primary services that 5G supports are data-intensive traffics such as virtual reality, cloud gaming, modern online banking, and Internet of Things (IoT) [8]. To maintain reliable end-to-end connections in a high density network such as 5G, the congestion control mechanism of Transmission Control Protocol (TCP) is widely used to prevent network congestion caused by sudden spike of data [7]. One of the congestion control algorithms that is widely used in our Internet today is loss-based congestion control algorithms [9].

Throughout the years, many researchers propose state-of-the-art loss-based congestion control algorithms. For example, Abdullah *et al.*, (2023) [10] propose an enhanced TCP NewReno congestion avoidance for 5G networks, targeting loss-triggered rate changes. They adjust the Congestion Window (CWND) increase or decrease strategy in both slow-start and congestion-avoidance phases to better match the high-bandwidth, variable-RTT 5G channel. Majeed *et al.*, (2024) propose Buffer Occupancy-based Congestion Control (BOCC) which is a optimization-driven congestion control for wireless network [11]. The proposed work is to throttle sources before buffers overflow. Which means, the algorithm tries to slow down traffic early, when the buffers are nearly full, so that no packets are lost and the network remains smooth. If the buffer is full, new incoming packet is dropped. However, when the traffic slows down, the end-to-end delay increased and throughput is decreased. Furthermore, the complexity of the algorithm may also introduce additional delays [12]. Another relevant study is by Zhou *et al.*, (2024) [13] who used NS3 simulations to compare TCP Reno, New Reno and Cubic over wired links. They provide valuable insights into the behavior of TCP Reno, New Reno, and Cubic in simulated wired networks such as the throughputs. However, they highlighted that these algorithms do not respond to early signs of congestion such as queue buildup, and often increase delay due to reactive retransmission mechanisms and aggressive window reductions [14].

Based on the presented works [10, 11, 13], loss-based congestion control are reactive algorithms. Which means, they react only after congestion occurs. This shortcoming may not be suitable for fast high-density network. Furthermore, CPE that serves as the on-premises device that connects local networks to broader SD-WAN infrastructure can become overwhelmed by high data loads. Many legacy CPEs possess limited processing capability and restricted bandwidth, making them prone to congestion during traffic surges [15]. When CPEs fail to handle ingress and egress traffic efficiently, they form choke points within the network that result in higher latency, packet loss, and overall performance degradation. This bottleneck therefore undermines not only throughput but also the quality and reliability of critical services.

To mitigate this problem, this paper proposes a proactive congestion control algorithm in SD-WAN by using bucket sort algorithm, which aims to reduce end-to-end delay between CPE Since most network traffic is concentrated at these CPEs, which handle data from multiple sources. The proposed algorithm is developed using Python in PyCharm. The packets from multiple sources that have the same destination network Internet Protocol (IP) addresses are put into the same bucket at CPE. By doing this, the routers only have to check their respective forwarding table once. This will reduce the lookup time and ultimately, reduces the end-to-end delay. The details of this algorithm are discussed in the methodology section.

The remainder of this paper is structured as follows; Section 2, we discuss on the methodology of the proposed algorithm. Then in Section 3, the performance of the proposed algorithm in terms of end-to-end delay between two CPEs are presented and analyzed. Furthermore, it is compared with traditional loss-based algorithms. Finally, this paper is concluded with Section 4.

2.0 METHODOLOGY

In this section, we introduce our proposed congestion control algorithm designed to minimize end-to-end delay between two CPEs in an SD-WAN environment. The proposed algorithm is implemented at the CPEs because these are the programmable nodes in SD-WAN. We employ a point-to-point transmission topology, where three clients transmit data simultaneously into one CPE as illustrated in Figure 2. The topology is simulated by using Python programming language in PyCharm. As a proof of concept, our topology comprises two CPE nodes, each equipped with an OpenFlow switch and three routers [16-18], and a centralized SDN controller. Unlike prior work in [19], which utilizes multiple channels, we opt for a single-channel setup to simulate a bottleneck environment where congestion is more

likely to occur. This approach ensures a controlled testing scenario where congestion control techniques can be thoroughly evaluated under high-traffic conditions. Furthermore, maintaining a single channel minimizes the frequency of channel switching which can negatively impact network performance [20]. Therefore, to maintain stability and focus on congestion mitigation, our approach adheres to a single-channel transmission only.

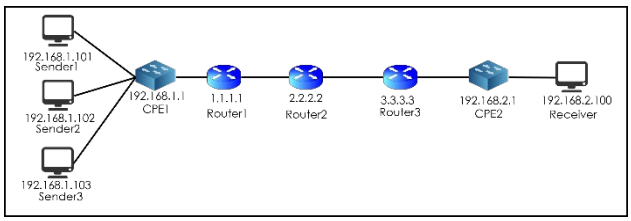


Figure 2 Simulation topology

For the proposed algorithm we adopt three core assumptions. First, the bucket capacity is chosen to approximate the Maximum Transmission Unit (MTU) so that each bucket can accommodate as many packets as possible. Although the standard MTU is 1500 bytes [21], our hardware constraints limit the bucket size to 1448 bytes. Second, each client transmits a 25 bytes payload 30 times to obtain meaningful end-to-end delay averages [22, 23]. Hence, allowing a larger number of packets to be aggregated per bucket. Each packet contains a 20 bytes transport header, a 20 bytes network header, and a 26 bytes data link header [24]. The third assumption is that the link capacity is configured at 1 Gigabit per second (Gbps), while transmission and processing delays are considered negligible, as they exceed 10 Mbps in typical WAN environments [25, 26]. Table 1 shows the summary of our assumptions. After establishing these parameters, we outline the detailed flow of the algorithm.

Table 1 Algorithm default assumptions

Items	Values
Bucket size	1448 bytes
Packet payload	25 bytes
Transport header	20 bytes
Network header	20 bytes
Datalink header	26 bytes
Bandwidth	1 Gbps

In Figure 2, The single centralized controller registers the IP addresses of each CPE. While a single controller simplifies management, it also constitutes a single point of failure under congestion conditions. To mitigate this risk, we introduce a congestion control mechanism inspired by the bucket-sort algorithm. The proposed method groups packets that share the same destination network IP addresses into a common bucket.

Figure 3 illustrates the algorithm implemented in CPE1. Upon receiving packets from all senders, the switch in CPE1 checks if the bucket still has available space. Then, the algorithm checks the source and destination network IP addresses of each packet by looking at the first three octets of the IP address. For example, the IP address of Sender 1 is 192.168.1.101. Therefore, its source network IP address is 192.168.1.1. Similarly, the destination IP address for Sender 1 is 192.168.2.100. Therefore, its destination network IP address is 192.168.2.1. The bucket uses these source and destination network IP addresses as its Network header. Then, the algorithm appends each packet to the current bucket until the bucket's remaining capacity is smaller than the incoming packet size. After placing a packet into the bucket, the switch deducts that packet's size from the bucket's available space and repeats this procedure for subsequent arrivals. If an incoming packet exceeds the bucket's remaining capacity, the switch transmits the filled bucket immediately. Then, the individual packets that could not be appended into the bucket is also forwarded separately. Similarly, when the source and destination network IP addresses of the individual packet do not belong to the bucket, it will be sent individually as well. However, in this simulation, we send the packets that are already belong to the bucket as a proof-of-concept.

When switch in CPE1 transmits the bucket, the routers treat it as a single packet. As a result, the router only performs one packet processing task for the entire bucket instead of processing each packet individually, which is the standard procedure in typical packet processing [27]. This method allows the router to handle other packets, hence, reducing packet queuing and minimizing the need for retransmissions.

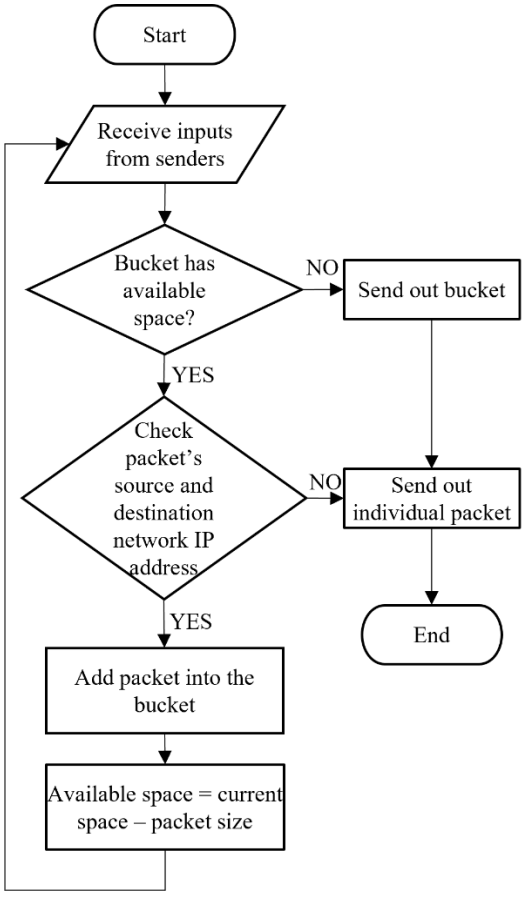


Figure 3 Algorithm in CPE1

On the other hand, Figure 4 shows the algorithm when the bucket arrives at CPE2. Upon arrival, the switch unpacks the

bucket by sorting all packets into an array. This allows the switch to forward each individual packet to its respective destination. Figure 5 illustrates the flow of the bucket transmission. This process is expected to reduce the end-to-end delay.

This aggregation strategy is inefficient for a single client because filling a bucket would require an extended duration, which is why the proof-of-concept uses three concurrent clients to accelerate bucket population. In operational networks, a large number of devices typically transmit concurrently, facilitating rapid bucket filling.

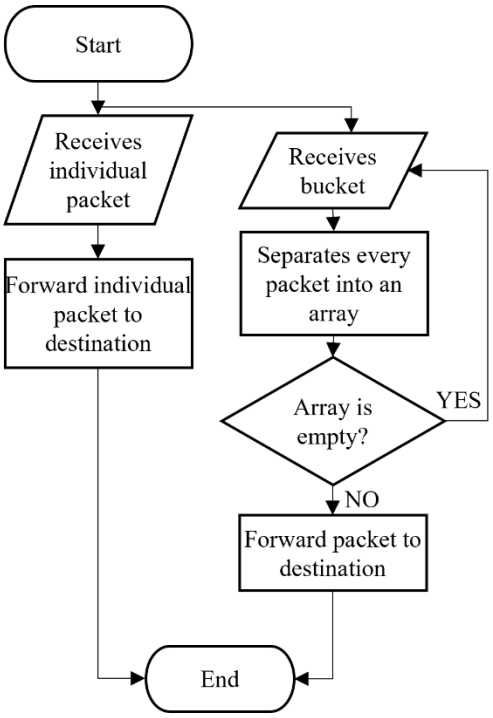


Figure 4 Algorithm in CPE2

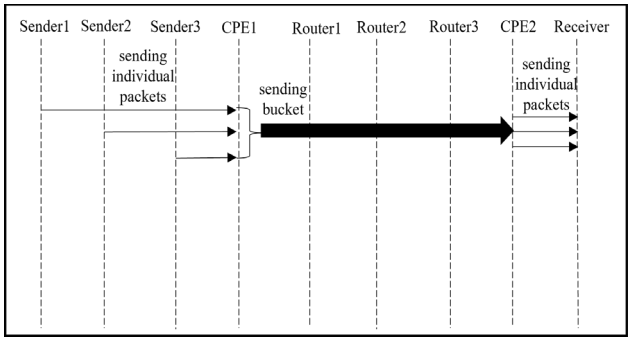


Figure 5 Bucket transmission flow

In traditional networking, the end-to-end delay, Z_p for each packet is calculated by using Equation 1 and Equation 2 [28]. *Z* is the summation of processing delay, *W* with the packet size, V_p over link capacity, *c*. *V* includes the payload, *pl* and the headers, *ph* [24]. In this equation, x_j is the link that the packet is current using. Whereas, x_{j+1} is the next link.

$$Z_p = \sum_{j=1}^{h}\left(W(x_j, x_{j+1}) + \frac{V_p}{c(x_j, x_{j+1})}\right); j = 1,2,3,\dots,h \quad (1)$$

where

$$V_p = ph + pl \quad (2)$$

In our proposed work, we use the size of the bucket, V_b is the summation of individual packets, V_p *with the bucket header, bh* as shown in Equation 3. In Equation 3, the limit of the bucket's payload is 1445 bytes as mentioned before. α is the iteration of packets until β^{th} packet.

$$V_b = \left(\lim_{pl \to 1445} \sum_{\alpha=1}^{\beta} V_{p_\alpha}\right) + bh$$

$$= \left(\lim_{pl \to 1445} \sum_{\alpha=1}^{\beta} (ph_\alpha + pl_\alpha)\right) + bh; \alpha = 1,2,3,\dots,\beta \quad (3)$$

Hence, the end-to-end delay of a bucket, Z_b can be defined in Equation 4. However, since our result also includes the end-to-end delay of individual packets that unable to fit in the bucket, the overall end-to-end delay, Z_t is the summation of Z_b *and* Z_p as shown in Equation 5.

$$Z_b = \sum_{j=1}^{h}\left(W(x_j, x_{j+1}) + \frac{V_b}{c(x_j, x_{j+1})}\right); j = 1,2,3,\dots,h \quad (4)$$

$$Z_t = Z_b + Z_p \quad (5)$$

After running the algorithm in PyCharm, its end-to-end performance is compared to loss-based congestion control algorithms. The default configurations and topology used are the same as in Figure 2. A detailed analysis of the resulting performance comparison is provided in the next section.

3.0 RESULTS AND DISCUSSION

In this section we compare our proposed algorithm and loss-based algorithms, which are Reno, New Reno and Cubic in terms of end-to-end delay in seconds (s) between two CPEs. The end-to-end delay is evaluated between the two CPE nodes, as they serve as the primary aggregation points for network traffic and therefore represent the most congestion-prone locations in the network. The objective of this comparison is to prove that our proposed algorithm works better in congested network. As mentioned in methodology section, each transmission is repeated for 30 times.

Table 2 shows the results of the simulation in terms of average end-to-end delay. Figure 6 illustrates the comparisons of end-to-end delay graphs that reflects the values in Table 2. The end-to-end delay values produced by our proposed algorithm are range from 33.81s to 35.05s with the maximum standard deviation of ±1s. Reno produces end-to-end delay range from 34.5s to 41.25(±1)s. New Reno produces end-to-end delay from 33.6s to

38.4(±2)s. Whereas, Cubic produces end-to-end delay from 33.57s to 40.74(±1)s. It shows that the proposed algorithm produces lower end-to-end delay with comparable variation to all loss-based congestion control algorithms except when transmitting 25 bytes. This is because of two reasons. The first reason is it is the first time clients send the data. The controller needs to install the flow table into the switch [29]. The second reason is it requires more time to fill up the bucket compared due to its small size. When transmitting bigger data size, our algorithm produces better end-to-end delays. This is because each router in the network has to process one bucket that has all the packets rather than processing them one by one. This also reduces the packet queuing and congestion. The individual packets that are unable to fit in the bucket, able to reach the nodes faster because there is now more space in the link since all the packets are in bucket.

Based on Table 2, it shows that the proposed algorithm produces more stable end-to-end delays compared to others as their values varies only in between 33s to about 35s. Whereas the end-to end delay produced by loss-based algorithms steadily continue to increase as the data size increases. This shows that loss-based algorithms are not suitable for high speed and high-density network. Furthermore, it proves that the proposed algorithm able to maintain the stability of the network despite increasing number of traffics.

Table 2 End-to-end delay comparisons

Transmitted data (bytes)	End-to-end delay (s)			
	Proposed algorithm	Reno	New Reno	Cubic
25	35.05	34.50	33.6	33.57
50	33.81	35.47	34.29	34.71
75	34.33	36.43	34.97	35.55
100	34.04	37.39	35.66	36.68
125	34.36	38.36	36.34	37.53
150	33.88	39.32	37.03	38.82
175	34.46	40.28	37.71	39.59
200	34.58	41.25	38.4	40.74

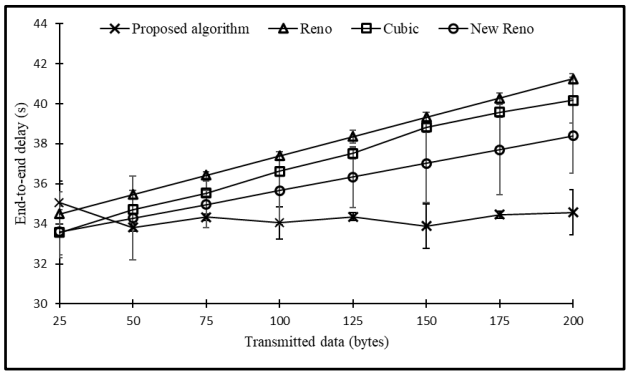


Figure 6 End-to-end delay comparisons between proposed algorithm with Reno, New Reno and CUBIC

Table 3 shows the number of buckets and individual packets that are transmitted during the simulation of the proposed algorithm. It shows that the algorithm works more effectively when larger data sizes are transmitted because less individual packets are transmitted. Hence, less Acknowledgement (ACK) messages were generated.

Table 3 Numbers of transmitted buckets and individual packets

Transmitted data (bytes)	Number of buckets transmitted	Number of individual packets transmitted
25	1	32
50	3	3
75	4	12
100	6	3
125	7	9
150	9	3
175	10	7
200	12	3

In conclusion, the proposed algorithm able to produce better end-to-end delays compared to traditional loss-based congestion control algorithms. This is because the proposed algorithm collects multiple packets and put them into one bucket. Then, it transmits the bucket as one packet. Hence, the nodes perceive this bucket as one packet. This reduces queueing packets and congestions. On the other hand, for loss-based algorithms, the routers need to inspect and process each packet one by one which consequently, produce higher delay. While our simulations involved only three clients, the advantages of the proposed approach is expected to scale well with increasing numbers of simultaneous clients. In larger deployments typical of SD-WAN environments, the aggregation of packets into buckets would occur more rapidly, further improving the system's bandwidth utilization and reducing per-packet processing delays. Future work will involve scaling up the simulation to multiple controllers and heterogenous network.

4.0 CONCLUSION

WAN have become fundamental for enabling communication across diverse platforms and end devices. However, the high operational cost of traditional WANs motivated the innovation of SDN, which decouples the control plane from the data plane and introduces programmability at network nodes, increasing flexibility and manageability. The integration of WAN and SDN that produces SD-WAN offers improved QoS at reduced cost. As network usage grows, traffic volumes increase and congestion becomes more likely, while low-latency delivery remains essential. Consequently, congestion control algorithms have been developed to address these challenges. Although numerous advanced congestion-control schemes have been proposed, most operate reactively and engage only after congestion has occurred. Furthermore, the CPE in SD-WAN frequently constitutes a performance bottleneck because of limited processing resources. To address this limitation, this paper presents a proactive congestion control algorithm based on a bucket-sort approach. The packets that share the same destination network IP address are aggregated into a single bucket and transmitted together. We evaluate the proposed algorithm against the traditional loss-based congestion control algorithms as they are widely used in today's Internet by comparing end-to-end delay between two CPEs. The

experimental results indicate that the bucket sort method achieves lower end-to-end delay than the loss-based.

Acknowledgement

The authors would like to thank Universiti Tenaga Nasional (UNITEN) for the financial support through the BOLD Grant 2024 (Project Code: J510051040).

Conflicts of Interest

The author(s) declare(s) that there is no conflict of interest regarding the publication of this paper

References

[1] S. Troia, F. Sapienza, L. Varé, and G. Maier, 2020. On Deep Reinforcement Learning for Traffic Engineering in SD-WAN. *IEEE Journal on Selected Areas in Communications*. 39(7): 2198-2212. DOI: https://doi.org/10.1109/JSAC.2020.3041385

[2] E. Obiodu, and N. Sastry, 2020. From ATM to MPLS and QCI: The Evolution of Differentiated QoS Standards and Implications for 5G Network Slicing. *IEEE Communications Standards Magazine*. 4(2): 14-21. DOI: https://doi.org/10.1109/MCOMSTD.001.1800041

[3] M. A. Ridwan, N. A. M. Radzi, W. S. H. M. Wan Ahmad, F. Abdullah, M. Z. Jamaludin, and M. N. Zakaria, 2020. Recent trends in MPLS networks: technologies, applications and challenges. *IET Communications*. 14(2): 177-185. DOI: https://doi.org/10.1049/iet-com.2018.6129

[4] S. D. A. Shah, M. A. Gregory, and S. Li, 2021. Cloud-Native Network Slicing Using Software Defined Networking Based Multi-Access Edge Computing: A Survey. *IEEE Access*. 9: 10903-10924. DOI: https://doi.org/10.1109/ACCESS.2021.3050155

[5] S. Ahmad, and A. H. Mir, 2022. SDN Interfaces: Protocols, Taxonomy and Challenges. *International Journal of Wireless and Microwave Technologies*. 12(2): 11-32. DOI: https://doi.org/10.5815/ijwmt.2022.02.02

[6] A. S. George, A. H. George, and T. Baskar, 2023. SD-WAN Security Threats, Bandwidth Issues, SLA, and Flaws: An In-Depth Analysis of FTTH, 4G, 5G, and Broadband Technologies. *Partners Universal International Innovation Journal*. 1(3): 1-37. DOI: https://doi.org/10.5281/zenodo.8057014

[7] J. Lorincz, Z. Klarin, and J. Ožegović, 2021. A Comprehensive Overview of TCP Congestion Control in 5G Networks: Research Challenges and Future Perspectives. *Sensors*. 21(13): 1-41. DOI: https://doi.org/10.3390/s21134510

[8] R. Dangi, P. Lalwani, G. Choudhary, I. You, and G. Pau, 2021. Study and Investigation on 5G Technology: A Systematic Review. *Sensors*. 22(1): 1-32. DOI: https://doi.org/10.3390/s22010026

[9] Y. Korbi, M. F. Zhani, and J. Kaippallimalil, 2024. Congestion Control in Wi-Fi Networks-State of the Art, Performance Evaluation, and Key Research Directions. *IEEE Access*. 12: 94972 - 94989. DOI: https://doi.org/10.1109/ACCESS.2024.3425271

[10] S. M. Abdullah, M. S. Farag, H. Abdul-Kader, and S. E. A. Youssef, 2023. Improving the TCP Newreno Congestion Avoidance Algorithm on 5G Networks. *Journal of Communications*. 18(4): 228-235. DOI: https://doi.org/10.12720/jcm.18.4.228-235

[11] U. Majeed, A. N. Malik, N. Abbas, A. S. Alfakeeh, M. A. Javed, and W. Abbass, 2024. Buffer Occupancy-Based Congestion Control Protocol for Wireless Multimedia Sensor Networks. *Electronics*. 13(22): 1-26. DOI: https://doi.org/10.3390/electronics13224454

[12] A. Fausto, G. Gaggero, F. Patrone, and M. Marchese, 2022. Reduction of The Delays Within an Intrusion Detection System (IDS) Based On Software Defined Networking (SDN). *IEEE Access*. 10: 109850-109862. DOI: https://doi.org/10.1109/ACCESS.2022.3214974

[13] Z. Weichen, A. H. M. Aman, Z. S. Attarbashi, W. Muhammad, H. Azamuddin, and A. D. Khaleel, 2026. Analysing TCP Traffic Congestion Algorithms for Wired Links Based on NS3. *Journal of Advanced Research in Applied Sciences and Engineering Technology* 58(1): 242-251. DOI: https://doi.org/10.37934/araset.58.1.242251

[14] W. Wei, H. Gu, and B. Li, 2021. Congestion Control: A Renaissance With Machine Learning. *IEEE Network*. 35(4): 262-269. DOI: https://doi.org/10.1109/MNET.011.2000603

[15] M. A. Ouamri, T. Alharbi, D. Singh, and Z. Sylia, 2025. A comprehensive survey on software-defined wide area network (SD-WAN): principles, opportunities and future challenges. *The Journal of Supercomputing*. 81(1): 1-47. DOI: https://doi.org/10.1007/s11227-024-06718-1

[16] A. Abane, M. Cubeddu, V. S. Mai, and A. Battou, 2025. Entanglement Routing in Quantum Networks: A Comprehensive Survey. *IEEE Transactions on Quantum Engineering*. 6: 1-39. DOI: https://doi.org/10.1109/TQE.2025.3541123

[17] H. S. Alotaibi, M. A. Gregory, and S. Li, 2022. Multidomain SDN-Based Gateways and Border Gateway Protocol. *Journal of Computer Networks and Communications*. 2022(1): 1-23. DOI: https://doi.org/10.1155/2022/3955800

[18] M. Nguyen, and S. Debroy, 2022. Moving Target Defense-Based Denial-of-Service Mitigation in Cloud Environments: A Survey. *Security and Communication Networks*. 2022(1): 1-24. DOI: https://doi.org/10.1155/2022/2223050

[19] S. Troia, M. Mazzara, M. Savi, L. M. M. Zorello, and G. Maier, 2022. Resilience of Delay-sensitive Services with Transport-layer Monitoring in SD-WAN. *IEEE Transactions on Network and Service Management*. 19(3): 2652-2663. DOI: https://doi.org/10.1109/TNSM.2022.3191943

[20] Y. Zhang, J. Tourrilhes, Z.-L. Zhang, and P. Sharma, 2021. Improving SD-WAN Resilience: From Vertical Handoff to WAN-Aware MPTCP. *IEEE Transactions on Network and Service Management*. 18(1): 347-361. DOI: https://doi.org/10.1109/TNSM.2021.3052471

[21] I. Hussain, and J. Bashir, 2022. Dynamic MTU: A Smaller Path MTU Size Technique to Reduce Packet Drops in IPv6. *Journal of King Saud University-Computer and Information Sciences*. 34(9): 7070-7088. DOI: https://doi.org/10.1016/j.jksuci.2021.06.011

[22] A. Sahu, V. Venkatraman, and R. Macwan, 2023. Reinforcement Learning Environment for Cyber-Resilient Power Distribution System. *IEEE Access*. 11: 127216-127228. DOI: https://doi.org/10.1109/ACCESS.2023.3282182

[23] F. J. Andújar, M. Sánchez de la Rosa, J. Escudero-Sahuquillo, and J. L. Sánchez, 2023. Extending the VEF traces framework to model data center network workloads. *The Journal of Supercomputing*. 79(1): 814-831. DOI: https://doi.org/10.1007/s11227-022-04692-0

[24] K. Alwasel, D. N. Jha, E. Hernandez, D. Puthal, M. Barika, B. Varghese, S. K. Garg, P. James, A. Zomaya, and G. Morgan, 2020. IOTSIM-SDWAN: A Simulation Framework for Interconnecting Distributed Datacenters Over Software-Defined Wide Area Network (SD-WAN). *Journal of Parallel and Distributed Computing*. 143: 17-35. DOI: https://doi.org/10.1016/j.jpdc.2020.04.006

[25] D. Chefrour, 2021. One-way delay measurement from traditional networks to sdn: A survey. *ACM Computing Surveys (CSUR)*. 54(7): 1-35. DOI: https://doi.org/10.1145/3466167

[26] Y. Tan, M. Veeraraghavan, H. Lee, S. Emmerson, and J. W. Davidson, 2022. High-performance reliable network-multicast over a trial deployment. *Cluster Computing*. 25(4): 2931-2952. DOI: https://doi.org/10.1007/s10586-021-03519-6

[27] T. Korikawa, and E. Oki, 2022. Memory Network Architecture for Packet Processing in Functions Virtualization. *IEEE Transactions on Network and Service Management*. 19(3): 3304-3322. DOI: https://doi.org/10.1109/TNSM.2022.3159091

[28] Y. Chen, 2020. End-to-End Delay Approximation in Packet-Switched Networks. *arXiv preprint arXiv:2003.08780*. 1-14. DOI: https://doi.org/10.48550/arXiv.2003.08780

[29] B. Isyaku, M. S. Mohd Zahid, M. Bte Kamat, K. Abu Bakar, and F. A. Ghaleb, 2020. Software Defined Networking Flow Table Management of OpenFlow Switches Performance and Security Challenges: A Survey. *Future Internet*. 12(9): 1-30. DOI: https://doi.org/10.3390/fi12090147