Performance Analysis of TCP Variants

Nan Chen
chen.nan2@northeastern.edu

Intro

The Transmission Control Protocol (TCP) is an essential component of the Internet, providing reliable data delivery over IP networks. While the original TCP design worked well for small networks, it was not designed to handle the large and congested networks that exist today. As a result, several TCP variants have been developed, each with different congestion control mechanisms. In this project, we will study the performance of TCP variants under various network conditions using the NS-2 network simulator.

Our first set of experiments will focus on the performance of these four TCP variants (Tahoe, Reno, NewReno, and Vegas) in the presence of a Constant Bit Rate (CBR) flow. We will vary the bandwidth of the CBR flow and analyze the throughput, packet drop rate, and latency of the TCP stream. Our second set of experiments will investigate the fairness between different TCP variants when competing for bandwidth with a CBR flow. We will measure the performance of two TCP flows (one from N1 to N4 and the other from N5 to N6) and a CBR flow (from N2 to N3) using different combinations of TCP variants. Finally, in our third set of experiments, we will study the performance of TCP Reno and SACK under two queuing algorithms: DropTail and RED. We will analyze the fairness between the TCP and CBR flows and the impact of the queuing algorithm on end-to-end latency and TCP behavior.

The importance of studying TCP variants lies in their impact on network performance, particularly in large and congested networks. Understanding the behavior of these TCP variants and their interaction with other network components such as queuing algorithms is crucial for the development of efficient and reliable network protocols. Our experiments will provide insight into the relative strengths and weaknesses of different TCP variants and contribute to the ongoing efforts to improve network performance and reliability. Our key results will include the identification of TCP variants that provide higher average throughput, lower latency, and fewer drops, as well as the evaluation of fairness between different TCP variants and the impact of queuing algorithms on TCP performance.

Methodology

In this study, we conducted three experiments to evaluate the performance of different TCP variants and queue types in a simulated network environment. We used the NS-2 network simulator, which is a widely used tool for network simulation and provides a realistic simulation environment for evaluating network protocols and algorithms. We also used Python, which is a widely used language for data analysis and visualization, to analyze and visualize the results of the experiments. To study the behavior of TCP Variants, we set up a network topology as shown in Figure 0.1.

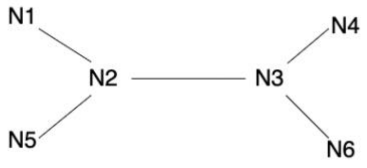

Figure 0.1: Network Topology

Experiment 1: In the first experiment, we evaluated the performance of four TCP variants (Tahoe, Reno, NewReno, and Vegas) under different CBR flow rates using the NS-2 network simulator. We chose CBR flow rates from 1 to 10 Mbps, with increments of 0.1 Mbps, 0.5 Mbps or 1 Mbps, to cover a wide range of flow rates and evaluate the performance of TCP variants under different traffic loads. We set up a simple network topology with six nodes and five links and created a TCP connection from node 1 to node 4 and an FTP over TCP connection to generate traffic. We also set up a CBR over UDP connection from node 2 to node 3 to generate a continuous bit rate flow. We set the queue limit to 10 between nodes 2 and 3 to simulate a congested network. We set up a network topology as shown in Figure 0.2.

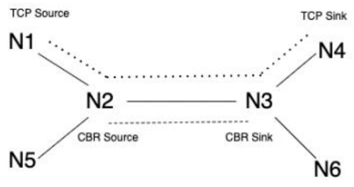

Figure 0.2: Simulation Scenario for Experiment 1

To analyze the results, we used the analyzeData1.py script to parse the trace file generated by the simulation and calculate three performance metrics: throughput, average latency, and packet drop rate. We also used the generateGraph1.py script to generate graphs for each TCP variant and CBR flow rate combination.

Experiment 2: In the second experiment, we evaluated the performance of four TCP variants (Reno/Reno, NewReno/Reno, Vegas/Vegas, and NewReno/Vegas) under different CBR flow rates using the NS-2 network simulator. We set up a simple network topology with six nodes and five links and created two TCP connections from N1 to N4 and N5 to N6 and two FTPs over TCP connections to generate traffic. We also set up a CBR over UDP connection from node 2 to node 3 to generate a continuous bit rate flow. We chose the same CBR flow rates as in Experiment 1 and set the link bandwidth and delay to 10 Mbps and 10 ms, respectively. We also set the queue limit to 10 packets for the link between nodes 2 and 3. We set up a network topology as shown in Figure 0.3.

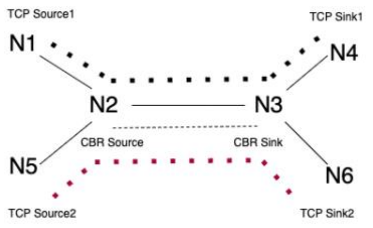

Figure 0.3: Simulation Scenario for Experiment 2

To analyze the results, we used the analyzeData2.py script to calculate the performance metrics for each TCP variant and generateGraph2.py script to visualize the results. Then we got three graphs for each pair of TCP variants, which represent the average throughput, packet loss rate, and latency of two TCP flows respectively according to the different CBR flow's rate.

Experiment 3: In the third experiment, we evaluated the performance of two TCP variants (Reno and Sack1) and two queue types (RED and DropTail) using the NS-2 network simulator. We set up a network topology with six nodes and used FTP over TCP and CBR over UDP traffic to simulate traffic flows. FTP traffic was sent from node n1 to node n4, while CBR traffic was sent from node n5 to node n6. The CBR traffic rate was set to 5Mb/s. The window size for TCP agents was set to 80 packets. After setting up the network, the script ran the simulation for a period of 10 seconds. We set up a network topology as shown in Figure 0.4.

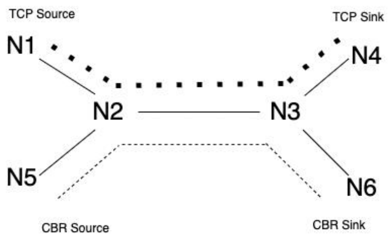

Figure 0.4: Simulation Scenario for Experiment 3

To analyze the results, we used the analyzeData3.py script to calculate the throughput and latency for each time interval (0.5s). The script read the trace file and calculated the throughput and latency for each time interval.

The throughput was calculated as the total number of bits received by the TCP agent and CBR traffic over a time interval, divided by the time interval and converted to Mbps. The latency was calculated as the time taken for a packet to travel from the source to the destination, averaged over all packets that arrived at the destination within a time interval. A Python script called generateGraph3.py was used to generate graphs from the data collected by analyzeData3.py. The script read the data from a file and plotted the throughput and latency for each TCP variant and queue type combination.

1 Experiment 1

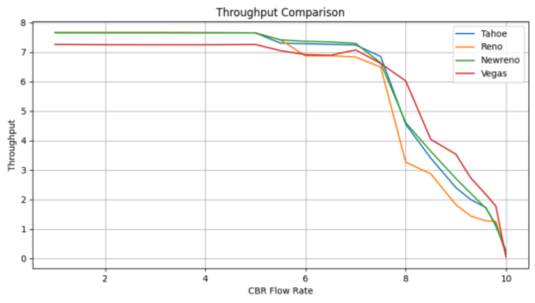

Figure 1.1: Throughput Among Four TCP variants

From the results, it is clear that Vegas and NewReno are the TCP variants that achieved the highest average throughput. Vegas has the highest throughput among all the TCP variants for all bandwidth levels, while NewReno has the second-highest throughput, except at very low bandwidth levels where Tahoe has a slightly higher throughput. This can be attributed to the proactive congestion control mechanism employed by TCP Vegas, which estimates the available bandwidth and adjusts the sending rate accordingly. This results in better utilization of network resources and fewer packet drops.

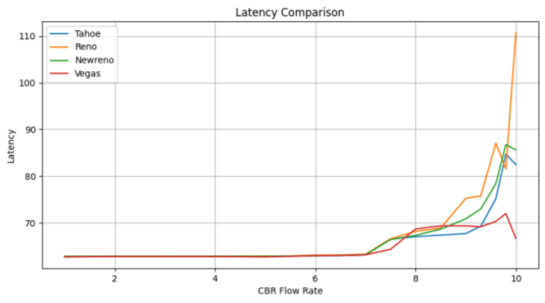

Figure 1.2: Latency Comparison Among Four TCP variants

Among the four TCP variants, Vegas has the lowest average latency for all bandwidth levels. The latency of Vegas remains relatively constant throughout the experiment. This is due to its proactive congestion control mechanism, which allows it to react more quickly to network changes and maintain a smaller backlog of packets in the queues, resulting in lower latency. On the other hand, the latency of other TCP variants increases as the bandwidth of the CBR flow increases. Tahoe has the highest average latency among all the TCP variants.

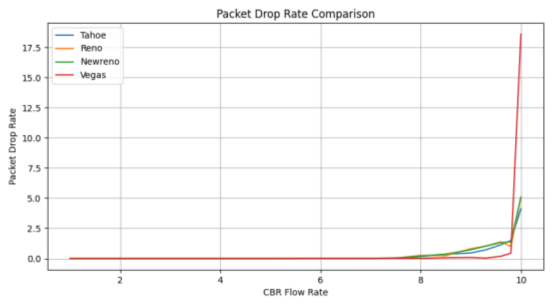

Figure 1.3: Packet Drop Rate Among Four TCP variants

Among the four TCP variants, Vegas has the fewest drops for all bandwidth levels. This is because Vegas uses a different congestion control algorithm than the other TCP variants, which allows it to avoid packet drops more effectively.

The behavior of different TCP variants is influenced by their congestion control mechanisms.

TCP Tahoe and Reno rely on reactive congestion control, where they reduce the sending rate only after detecting packet loss (Tahoe) or receiving duplicate acknowledgments (Reno). This can lead to a more conservative sending rate and a higher likelihood of packet drops due to congestion.

TCP NewReno improves upon Reno by modifying its congestion control mechanism to recover faster from multiple packet losses. However, it still employs a reactive approach and can experience higher packet drops than Vegas.

TCP Vegas employs a proactive congestion control mechanism. It estimates the available bandwidth and adjusts the sending rate accordingly, avoiding congestion and minimizing packet drops.

In conclusion, TCP Vegas outperforms the other three TCP variants in this experiment due to its proactive congestion control mechanism, resulting in higher throughput, lower latency, and fewer packet drops. However, the best TCP variant might change depending on specific network conditions and application requirements.

2 Experiment 2

It seems that the assumption of fairness between different TCP variants is not always true. In particular, the results from the experiment suggest that certain combinations of TCP variants are more prone to unfairness than others.

Looking at the results from the tests, we can see that in most cases, the average throughput, packet loss rate, and latency of the TCP flows decrease as the bandwidth used by the CBR flow increases. This is not surprising, as the CBR flow is essentially competing with the TCP flows for available bandwidth, and the more bandwidth the CBR flow uses, the less is available for the TCP flows.

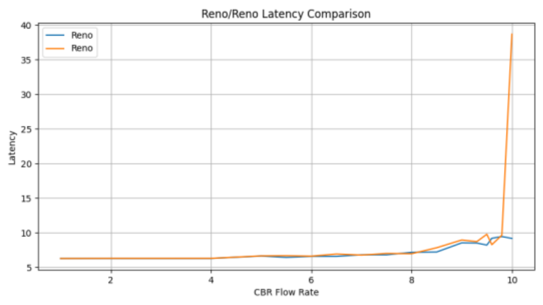

Figure 2.1: Reno/Reno Latency Comparison

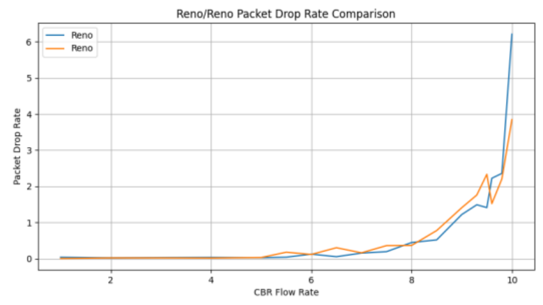

Figure 2.2: Reno/Reno Packet Drop Rate Comparison

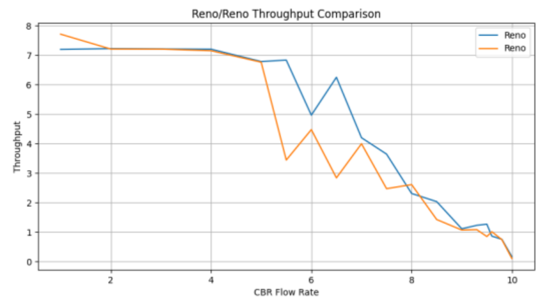

Figure 2.3: Reno/Reno Throughput Comparison

As shown in Figure 2.1/ Figure 2.2/ Figure 2.3, in the Reno/Reno case, the throughput of both TCP flows is relatively stable until the CBR flow reaches a bandwidth of around 8.5 Mbps, at which point it starts to drop sharply.

As the bandwidth used by the CBR flow increases, the average throughput of both Reno flows decreases. Latency increases for both flows, and packet loss rate increases as well.

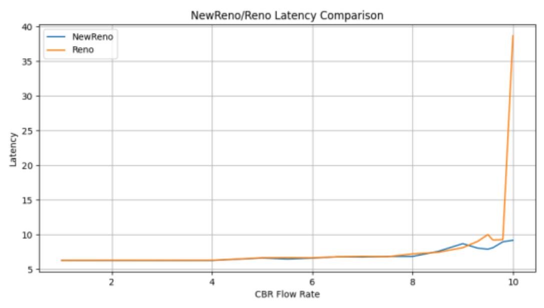

Figure 2.4: NewReno/Reno Latency Comparison

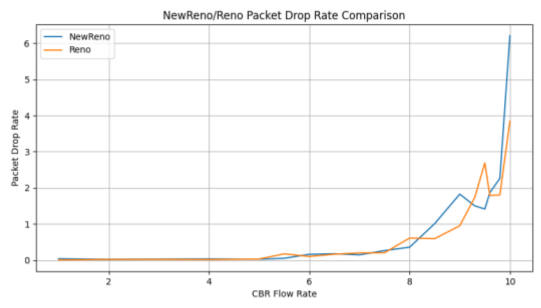

Figure 2.5: NewReno/Reno Packet Drop Rate Comparison

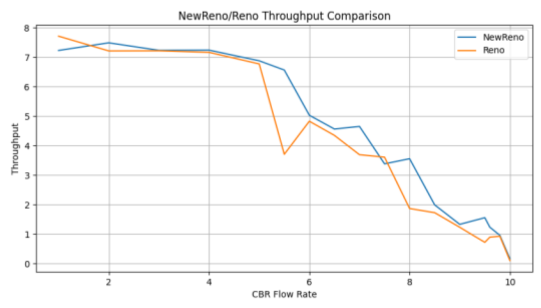

Figure 2.6: NewReno/Reno Throughput Comparison

As shown in Figure 2.4/ Figure 2.5/ Figure 2.6, in the NewReno/Reno case, the throughput of the first TCP flow increases steadily up to a CBR bandwidth of around 7 Mbps, before starting to level off, while the throughput of the second TCP flow is more sensitive to changes in the CBR bandwidth, dropping sharply as soon as the CBR flow starts using more than around 5 Mbps.

The average throughput of both flows decreases as the bandwidth used by the CBR flow increases, with the NewReno flow generally having a slightly higher throughput. Latency increases for both flows, and packet loss rate increases as well.

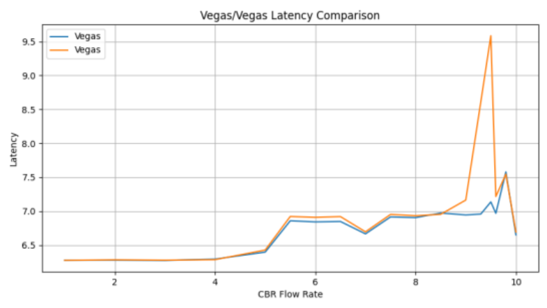

Figure 2.7: Vegas/Vegas Latency Comparison

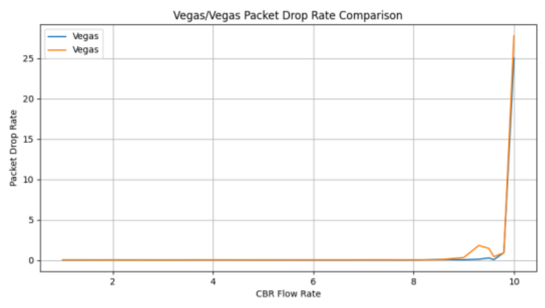

Figure 2.8: Vegas/Vegas Packet Drop Rate Comparison

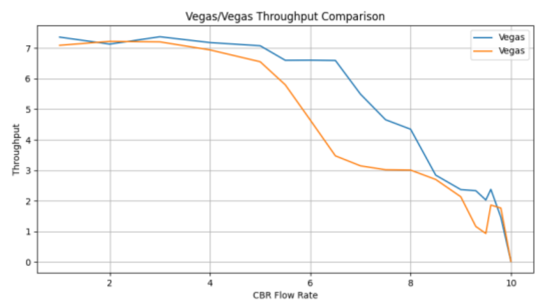

Figure 2.9: Vegas/Vegas Throughput Comparison

As shown in Figure 2.7/ Figure 2.8/ Figure 2.9, in the Vegas/Vegas case, the first TCP flow is relatively insensitive to changes in the CBR bandwidth, while the second TCP flow is much more affected, with its throughput dropping sharply as soon as the CBR flow starts using more than around 7.5 Mbps.

As the bandwidth used by the CBR flow increases, the average throughput of both Vegas flows decreases. Latency remains relatively stable for both flows but eventually increases as the CBR flow bandwidth becomes higher. Packet loss rate remains low initially but increases significantly for high CBR flow bandwidth values.

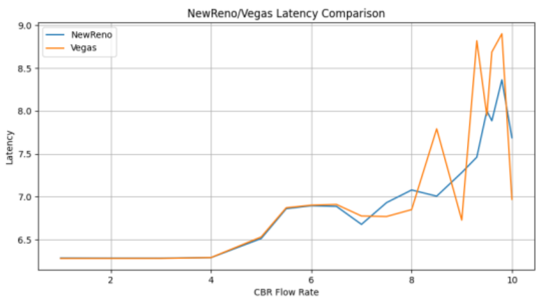

Figure 2.10: NewVegas/Vegas Latency Comparison

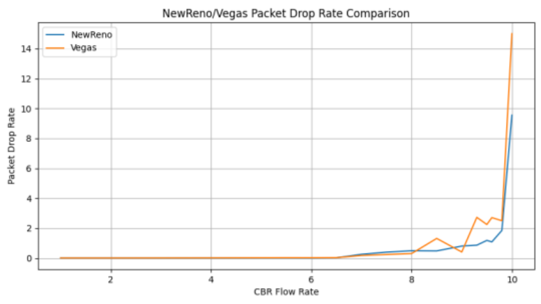

Figure 2.11: NewVegas/Vegas Packet Drop Rate Comparison

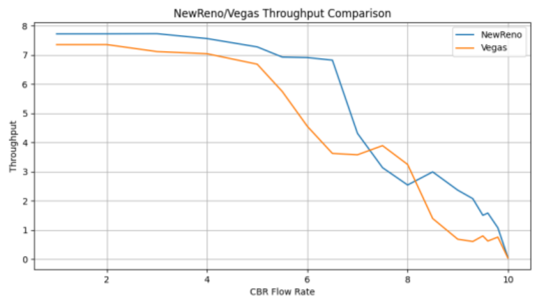

Figure 2.12: NewVegas/Vegas Throughput Comparison

As shown in Figure 2.10/ Figure 2.11/ Figure 2.12, in the NewReno/Vegas case, the first TCP flow is again relatively insensitive to changes in the CBR bandwidth, while the second TCP flow is more sensitive, with its throughput dropping sharply as soon as the CBR flow starts using more than around 6.5 Mbps.

The average throughput of both flows decreases as the bandwidth used by the CBR flow increases, with the NewReno flow generally.

Based on these results, we can infer that the ideal assumption of fairness between different TCP variants is not entirely true. The performance of each TCP variant depends on the specific variant and the network conditions. In general, as the CBR flow bandwidth increases, the TCP flows' throughput decreases, and their latency and packet loss rate increase.

It is also observed that the performance of the Reno and NewReno variants is more sensitive to CBR flow bandwidth changes compared to the Vegas variant. The Vegas variant exhibits more stable latency and packet loss rate values across a wider range of CBR flow bandwidths.

In summary, the fairness between different TCP variants is not perfect, and the performance of each variant depends on the specific variant and the network conditions. However, it is important to note that the experiments were conducted in a controlled environment and may not fully represent the behavior of TCP variants in real-world networks.

3 Experiment 3

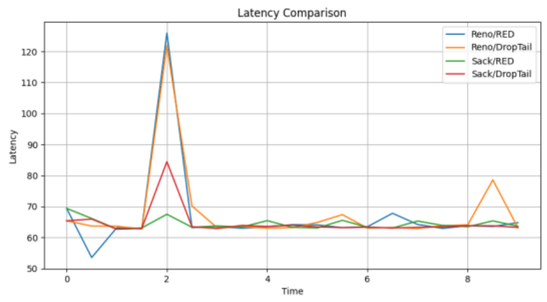

Figure 3.1: Latency Comparison between Queuing disciplines

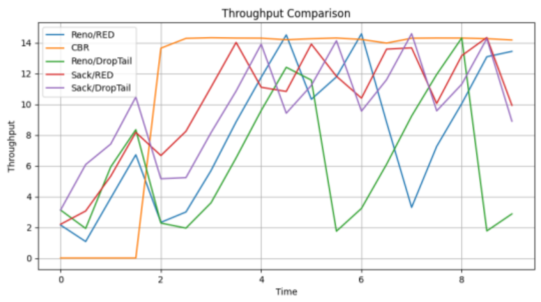

Figure 3.2: Throughput Comparison between Queuing disciplines

In general, RED seems to provide a fairer bandwidth distribution between the TCP and CBR flows when compared to DropTail. This can be observed in the throughput values for both flows under different TCP variants.

For Reno: Reno/RED: The TCP flow has an average throughput of around 8.5 Mbps, while the CBR flow maintains around 14 Mbps. Reno/DropTail: The TCP flow has a much lower average throughput of around 6.5 Mbps, while the CBR flow maintains around 14 Mbps.

For SACK: Sack/RED: The TCP flow has an average throughput of around 10.5 Mbps, while the CBR flow maintains around 14 Mbps. Sack/DropTail: The TCP flow has an average throughput of around 9 Mbps, while the CBR flow maintains around 14 Mbps.

The latency values show that RED typically results in lower end-to-end latency compared to DropTail. This is because RED detects congestion earlier and drops packets to avoid a full queue, whereas DropTail waits until the queue is full, leading to higher latency.

When the CBR flow starts, the TCP flow generally experiences a decrease in throughput, as the two flows now compete for the same resources. This effect is more pronounced in the DropTail scenario, where the TCP flow suffers more significant throughput reduction compared to the RED scenario.

RED seems to work well with both Reno and SACK TCP variants, as it helps maintain lower latency and provides a fairer bandwidth distribution compared to DropTail. In the SACK scenario, RED performs better in terms of latency and throughput distribution.

SACK is designed to recover more quickly from packet losses and handle multiple packet losses in a single round-trip time, while Reno relies on a more traditional retransmission mechanism. Thus, SACK can benefit from the early congestion detection and proactive packet dropping provided by RED, leading to improved overall performance.

Each TCP variant reacts differently to queue types based on their congestion control and loss recovery mechanisms. Reno is more sensitive to packet losses, so it is more affected by the aggressive packet dropping in DropTail. SACK, on the other hand, can handle multiple losses more effectively, making it better suited to deal with RED's early congestion detection.

Based on the data provided, the combinations of RED with both Reno and SACK seem to work well together. DropTail with Reno shows a more significant impact on the TCP flow, so it may be less desirable. However, it's important to consider that network conditions and specific use cases could change these results, so it's essential to test and evaluate the performance of each combination in different scenarios.

Conclusion

This study conducted experiments to analyze the performance of TCP variants (Tahoe, Reno, NewReno, and Vegas) under congestion, fairness between TCP variants, and the influence of queuing disciplines on overall throughput. These experiments are significant as they provide insights into the behavior of different TCP variants under various network conditions, which can help optimize network performance and ensure fairness among users.

The results from the first experiment shed light on the strengths and weaknesses of each TCP variant under congestion. Although there may not be a universally "best" TCP variant, the findings can be used to optimize the choice of TCP variant based on specific network requirements and conditions. Understanding how different TCP variants perform in the presence of unresponsive flows, such as the CBR/UDP flow, is crucial for enhancing network efficiency and user experience.

In the second experiment, we investigated the fairness between different TCP variants. The results revealed that some combinations of TCP variants were more unfair than others, and that the fairness could be affected by the design and implementation of the TCP protocols. This understanding of TCP variant interactions is important for network administrators and engineers when designing and configuring networks to avoid unintentional biases and ensure equal opportunities for all users.

The third experiment focused on the influence of queuing disciplines, such as DropTail and RED, on the overall throughput of TCP and CBR flows. By comparing the performance of the flows under different queuing disciplines, we gained insights into the suitability of each discipline for specific TCP variants and scenarios. This knowledge can be useful for network operators when selecting queuing algorithms to balance the needs of diverse applications and users, while maintaining overall network performance.

The real-world significance of our results is evident in the potential impact on deployed systems. By selecting the most appropriate TCP variant and queuing discipline for a given network environment, operators can ensure optimal performance, user satisfaction, and fair resource allocation. Additionally, our findings may inform the development of new TCP protocols or improvements to existing ones.

Despite the valuable insights gained from this study, there are still open questions and areas for future research. For example, exploring the interactions between other TCP variants or studying the impact of more complex network topologies and scenarios could reveal additional nuances in TCP behavior. Moreover, investigations into alternative queuing disciplines and congestion control mechanisms could further refine our understanding of network performance and fairness.

REFERENCES

[1] Jain, R. (1990). The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling. John Wiley & Sons.

[2] Padhye, J., Firoiu, V., Towsley, D., & Kurose, J. (2000). Modeling TCP Throughput: A Simple Model and its Empirical Validation. ACM SIGCOMM Computer Communication Review, 28(4), 303-314. https://doi.org/10.1145/964723.383071

[3] Allman, M., Paxson, V., & Stevens, W. (1999). TCP Congestion Control. IETF RFC 2581. Internet Engineering Task Force. https://tools.ietf.org/html/rfc2581

[4] Brakmo, L. S., & Peterson, L. L. (1995). TCP Vegas: End to End Congestion Avoidance on a Global Internet. IEEE Journal on Selected Areas in Communications, 13(8), 1465-1480. https://doi.org/10.1109/49.464716

[5] Floyd, S., & Jacobson, V. (1993). Random Early Detection Gateways for Congestion Avoidance. IEEE/ACM Transactions on Networking, 1(4), 397-413. https://doi.org/10.1109/90.251892