Enhancing Network Efficiency Through TCP E-newreno Congestion Control in High-Density Network

Karthik K P[1], Chinmay S Avaradi[2], Chandana Gotur[3], Sinchan H[4], Dr. Jayalaxmi G Naragund [5]

[1]School of Computer Science and Engineering

[1] KLE Technological University, Hubli, Karnataka, India

{ 01fe22bcs183, 01fe22bcs306, 01fe22bcs104, 01fe22bcs055, jaya_gn}@kletech.ac.in,

***Abstract*—This paper presents a comprehensive performance evaluation of a novel TCP variant, TCP ENewReno, designed to enhance congestion control efficiency in high-density and delay-sensitive network environments. TCP ENewReno introduces an adaptive congestion avoidance mechanism that leverages real-time bandwidth estimation through acknowledgment (ACK) intervals to dynamically adjust the congestion window size (CWND). Using NS-2 simulations, we compared TCP ENewReno against traditional TCP Reno and TCP NewReno variants across four critical metrics: Throughput, End-to-End Delay, Congestion Window Size, and Jain's Fairness Index (JFI). The simulations were conducted over a high-density, mixed wired-wireless topology involving 300 User Equipment (UE) nodes, each initiating FTP sessions under uniform load conditions. Results show that TCP ENewReno consistently outperforms its counterparts, achieving up to 34% higher throughput, more stable and lower delay, better congestion window utilization, and the highest fairness index. These findings highlight TCP ENewReno's capability to maintain efficient and fair resource utilization under dynamic network conditions, making it a promising solution for emerging 5G and next-generation networking scenarios.**

Keywords: TCP ENewReno, Congestion Control, NS-2 Simulation, Throughput, End-to-End Delay, CWND, Jain's Fairness Index, 5G Networks

I. Introduction

The Transmission Control Protocol (TCP)[1] is a cornerstone of the Internet architecture, responsible for ensuring reliable, ordered, and congestion-controlled[2] delivery of data across a wide range of applications. As the volume and diversity of Internet traffic continue to grow, so too does the demand for TCP to perform efficiently in increasingly complex network environments[3]. Over the years, a variety of TCP variants have been developed to address issues such as congestion collapse[4], packet loss, and inefficient bandwidth utilization. Among the most widely adopted are TCP Reno, TCP NewReno, and emerging adaptive variants such as TCP ENewReno[5].

TCP Reno, introduced in the early 1990s, implemented important enhancements like Fast Retransmit and Fast Recovery[6], which allowed it to respond to packet loss without waiting for a timeout. These mechanisms significantly improved TCP's performance over earlier versions, particularly on wired networks. However, Reno's response to multiple packet losses within a single congestion window remained limited[7].

TCP NewReno built upon Reno by improving the retransmission behavior during the Fast Recovery phase[8]. It is capable of recovering from multiple losses without reverting to slow start, which makes it more robust in lossy environments. Despite these improvements, both Reno and NewReno use fixed congestion control heuristics that may not generalize well to dynamic or high-density scenarios, where network conditions fluctuate rapidly, and multiple flows compete for limited resources[9].

To overcome these limitations, researchers have proposed more adaptive TCP variants, such as TCP ENewReno, which introduces a bandwidth estimation-based congestion control mechanism[10]. This mechanism dynamically adjusts the Congestion Window (CWND) in real time based on perceived network bandwidth, enabling faster reaction to changing conditions such as transient congestion or sudden link capacity drops. This is especially beneficial in networks with a high number of concurrent connections, where traditional TCP algorithms can lead to unfair bandwidth sharing, inefficient congestion handling, and increased latency[11].

In this study, we conduct a comprehensive performance evaluation of TCP Reno, TCP NewReno, and TCP ENewReno in a high-density wired network environment. Using the NS-2[12] network simulator, we model a scenario where 100 user nodes for each variant initiate FTP[13] sessions to a common remote server through a shared base station and core network node. This simulation setup emulates a dense access environment where congestion and competition for bandwidth are prominent[14].

To objectively assess the effectiveness of each TCP variant, we evaluate four key performance metrics. These include throughput, which represents the amount of data successfully delivered over a period of time; end-to-end delay, referring to

the total time taken for a packet to travel from the sender to the receiver; congestion window size (CWND), which indicates the sender's control over the number of packets in transit and reflects its aggressiveness in utilizing network bandwidth; and Jain's Fairness Index (JFI), a widely used metric that quantifies the fairness of bandwidth distribution among multiple concurrent flows[11, 15].

The primary goal of this evaluation is to investigate how well each TCP variant performs in handling congestion, achieving high throughput, maintaining low latency, and distributing bandwidth fairly under high-load conditions. This kind of analysis is vital for network designers and protocol developers who aim to select or design TCP algorithms suited for modern, traffic-intensive environments such as data centers, academic networks, and enterprise backbones[16].

The rest of this paper is organized as follows: Section II provides an overview of related work on TCP variants and congestion control. Section III describes the simulation setup and methodology in detail. Section IV presents the results and analysis of the performance metrics. Finally, Section V concludes the study and suggests directions for future research.

II. Background Study

Transmission Control Protocol (TCP) plays a critical role in ensuring reliable communication over wired networks. Over the years, various TCP congestion control variants such as Reno, NewReno, and ENewReno [8, 10] have been proposed to enhance performance in terms of throughput, packet loss, and fairness. Despite these improvements, challenges remain in achieving optimal throughput while ensuring fairness across concurrent flows, especially under varying congestion levels.

The paper by J. Padhye et al. [7] provides a model to estimate TCP Reno's throughput using loss rate and RTT. It is useful for our work as it sets a baseline for comparing improved variants. However, it focuses only on Reno and ignores fairness among multiple flows. The paper also lacks evaluation of modern enhancements like ENewReno and does not use fairness metrics such as Jain's Fairness Index [11]. This limits its applicability in scenarios with diverse traffic conditions.

The paper by Saleh M. Abdullah et al. [17] proposes enhancements to TCP NewReno to optimize its performance in 5G networks, addressing issues like increased latency and packet loss in high-speed environments. It is relevant to our work as it shows how traditional TCP variants can be adapted for next-generation networks. However, the focus is limited to wireless 5G scenarios and does not evaluate performance in wired networks. Moreover, it lacks fairness analysis using metrics like Jain's Fairness Index and does not include a comparative study with ENewReno, leaving a gap that our work addresses by evaluating multiple TCP variants in a wired environment with both throughput and fairness as key metrics.

The paper by A. Afanasyev et al. [9] provides an overview of key congestion control mechanisms used in TCP, highlighting their strengths and weaknesses through comparative analysis. It emphasizes the growing challenges of congestion in modern high-traffic networks and the limitations of existing TCP variants in fully addressing these issues. The contribution lies in summarizing a wide range of algorithms and identifying gaps in their performance under different network conditions. However, the paper remains theoretical, with no simulation-based evaluation or fairness analysis using standardized metrics like Jain's Fairness Index. Unlike this survey, our work focuses on practical performance evaluation through NS2 simulations [18], comparing variants such as Reno, NewReno, and ENewReno using both throughput and fairness as core criteria.

The paper by T. Bonato et al. [19] introduces a novel congestion control algorithm tailored for datacenter environments, where traditional TCP variants struggle due to bursty and synchronized ML workloads. FASTFLOW integrates delay signals, ECN, and packet trimming with a new mechanism called QuickAdapt to enable accurate bandwidth estimation and rapid congestion response. While it achieves notable performance and fairness improvements over protocols like BBR and Swift [20, 21], its design is specialized for datacenters and ML-heavy traffic. In contrast, our work focuses on evaluating classical TCP variants like Reno, NewReno, and ENewReno in general-purpose wired networks, emphasizing fairness (via Jain's Index) and throughput under standard NS2 simulations—providing insights more applicable to traditional and heterogeneous network settings.

After analyzing the related works, it is evident that while previous studies have focused on individual TCP variants, their throughput behavior, or fairness aspects, most lack a comprehensive evaluation combining both throughput and fairness—especially for enhanced variants like TCP ENewReno. Some works focus only on theoretical modeling, while others perform limited simulations without using standardized fairness metrics like Jain's Fairness Index[11]. In contrast, our work provides a simulation-based comparative analysis of TCP Reno, NewReno, and ENewReno in wired networks, integrating both performance and fairness evaluation. By using Jain's Fairness Index alongside throughput measurements, we offer a more balanced and practical assessment, contributing a more holistic understanding of congestion control efficiency in multi-flow environments.

III. Simulation Setup and Methodology

To evaluate the performance of TCP Reno, TCP NewReno, and the proposed TCP ENewReno, a series of simulations were conducted using the Network Simulator 2 (NS-2) framework. The simulation environment was designed to emulate a dense wired infrastructure network, focusing on how each TCP variant responds to congestion and resource contention in a multi-user scenario. The goal was to understand each protocol's throughput behavior, responsiveness, and fairness under identical traffic and topology conditions.

A. Integration with NS-2

The proposed *TCP-ENewReno* was implemented as a custom congestion control agent in NS-2 by extending the existing `TcpNewReno` class. A new agent, `TcpENewReno`, was registered using C++ bindings and integrated into the NS-2 build system by modifying `Makefile.in`, followed by standard compilation steps. The key algorithmic enhancement involved modifying the `recv_ack()` method to incorporate Delay-to-Throughput Ratio (DTR) measurements, where changes in DTR (ΔDTR) guided CWND adjustments—decreasing CWND when $\Delta\text{DTR} > \beta$, increasing it when $\Delta\text{DTR} < -\beta$, and maintaining it otherwise for stability. Tracing hooks were added to monitor CWND dynamics, and the agent was instantiated in TCL scripts using `Agent/TCP/ENewreno`.

B. Simulation Topology

The simulation topology comprised a User Equipment (UE) connected to a Base Station (BS), which links to a Core Network (CN) node and a Remote Server (RS), all connected via full-duplex wired links at 10 Mbps. Latencies were set to 2 ms for UE–BS and BS–CN links, and 100 ms for the CN–RS segment to emulate edge and backbone delays. The simulation involved 100 UEs per TCP variant (Reno, NewReno, ENewReno), each initiating an FTP session for 60 seconds, staggered by 0.01 seconds to prevent burst synchronization. Packet size was fixed at 1500 bytes, and a uniform random loss model was used to emulate real-world noise. The `TCP ENewReno` variant dynamically adjusted CWND based on DTR, enhancing responsiveness across both low and high latency paths, and was evaluated against standard Reno and NewReno under identical conditions.

C. System Design

The internal architecture of the *TCP-ENewReno* agent is illustrated in Figure 1, which depicts its modular composition and integration points within the NS-2 simulation framework. The design emphasizes adaptive congestion responsiveness while maintaining compliance with the core principles of TCP operation. It introduces a dynamic bandwidth estimation mechanism based on the Delay-to-Throughput Ratio (DTR), embedded directly into the agent's congestion control logic.

1) Congestion Avoidance Logic

This module forms the core of the TCP-ENewReno algorithm. It enhances the traditional additive-increase behavior using real-time DTR measurements to adjust the congestion window (CWND). The logic functions through DTR computation, where at each ACK reception, the agent calculates DTR using the formula:

$$\text{DTR} = \frac{\text{Estimated RTT}}{\text{Current Throughput}} \tag{1}$$

This is followed by ΔDTR evaluation where the difference between the current and previous DTR values is computed as:

$$\Delta\text{DTR} = \text{DTR}_{\text{current}} - \text{DTR}_{\text{previous}} \tag{2}$$

The CWND adaptation follows specific rules: if $\Delta\text{DTR} > \beta$, CWND is multiplicatively decreased indicating congestion is likely; if $\Delta\text{DTR} < -\beta$, CWND is additively increased showing bandwidth availability; and if $|\Delta\text{DTR}| \leq \beta$, CWND remains unchanged to reduce oscillations.

This adaptive congestion control enables early detection of congestion while efficiently utilizing available capacity.

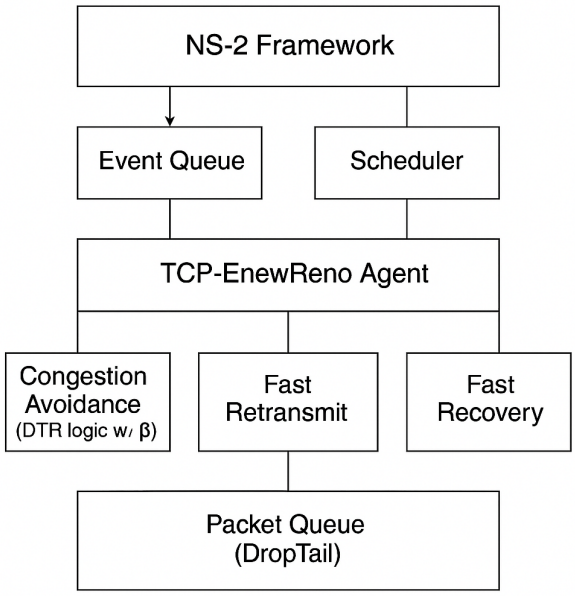

Fig. 1. Internal Architecture of the TCP-ENewReno Agent

2) Fast Retransmit and Fast Recovery

To maintain standard TCP behavior under loss, TCP-ENewReno includes Fast Retransmit which is triggered by three duplicate ACKs, indicating likely packet loss, and Fast Recovery which temporarily inflates CWND to allow continued transmission without full timeout.

These mechanisms ensure robustness and prevent unnecessary throughput degradation.

3) Packet Queue Management

TCP-ENewReno models congestion using a DropTail queue at each node with DropTail behavior implementing FIFO queuing where buffer overflow causes packet drops, while congestion manifestation occurs when queue buildup increases RTT, which in turn affects DTR values and CWND decisions.

This simple model aligns with NS-2's event-driven simulation while enabling realistic queue behavior analysis.

4) Module Interoperability and NS-2 Integration

Each module interfaces with key NS-2 components where the event scheduler handles ACK and timeout events, the link layer provides RTT samples and delivery notifications, and the tracing system logs CWND changes and transmission statistics.

D. TCP-ENewReno Congestion Control Algorithm

The *TCP-ENewReno* algorithm builds upon traditional TCP mechanisms such as slow start, fast retransmit, and fast recov-

ery, while incorporating a Delay-to-Throughput Ratio (DTR)-based congestion control strategy to improve adaptability under dynamic network conditions.

At the core of the algorithm, DTR is computed using observed RTT and throughput. The change in DTR (ΔDTR) between successive ACKs guides the adjustment of the congestion window (CWND): it decreases CWND when congestion is likely, increases it under available bandwidth, and maintains it during stability.

Algorithm 1 TCP-ENewReno Congestion Control Algorithm

Initialize cwnd $\leftarrow$ initial_window
Initialize $\text{DTR}_{\text{prev}} \leftarrow 0$
On ACK received:
$\text{DTR} \leftarrow \frac{\text{RTT}}{\text{Throughput}}$
$\Delta\text{DTR} \leftarrow \text{DTR}_{\text{prev}} - \text{DTR}$
$\text{DTR}_{\text{prev}} \leftarrow \text{DTR}$
if $\Delta\text{DTR} > \beta$ **then**
  $\text{cwnd} \leftarrow \text{cwnd} \times 0.8$
else if $\Delta\text{DTR} < -\beta$ **then**
  $\text{cwnd} \leftarrow \text{cwnd} + \frac{\alpha}{\text{cwnd}}$
end if
On 3 duplicate ACKs:
Perform Fast Retransmit and enter Fast Recovery =0

DTR is calculated as:

$$\text{DTR} = \frac{\text{RTT}}{\text{Throughput}}, \quad \Delta\text{DTR} = \text{DTR}_{\text{prev}} - \text{DTR} \tag{3}$$

CWND is then adjusted as:

$$\text{cwnd} = \begin{cases} \text{cwnd} \times 0.8 & \text{if } \Delta\text{DTR} > \beta \\ \text{cwnd} + \frac{\text{cwnd}}{\alpha} & \text{if } \Delta\text{DTR} < -\beta \\ \text{cwnd} & \text{otherwise} \end{cases} \tag{4}$$

Here, α controls the aggressiveness of CWND growth, and β defines the DTR sensitivity threshold. These parameters require tuning based on network dynamics. Packet loss is handled using Fast Retransmit and Fast Recovery upon receiving three duplicate ACKs.

IV. Results

This section presents a comprehensive comparative analysis of three TCP variants—TCP Reno, TCP NewReno, and TCP ENewReno—evaluated across four critical performance metrics: *Throughput*, *End-to-End Delay*, *Congestion Window Size (CWND)*, and *Jain's Fairness Index (JFI)*.

A. Performance Metrics

This section presents a granular evaluation of the performance of TCP Reno, TCP NewReno, and the proposed TCP ENewReno across four key metrics—*Throughput*, *End-to-End Delay*, *Congestion Window Size (CWND)*, and *Jain's Fairness Index (JFI)*. These metrics collectively assess the reliability, responsiveness, efficiency, and fairness of congestion control mechanisms under identical simulation conditions.

1) Throughput

Definition: Throughput is defined as the rate at which data packets are successfully delivered from the sender to the receiver over a network path. It is measured in megabits per second (Mbps) and is a direct indicator of a protocol's efficiency in utilizing available bandwidth.

TABLE I
AVERAGE THROUGHPUT COMPARISON

TCP Variant	Average Throughput (Mbps)
TCP ENewReno	5.37
TCP NewReno	3.99
TCP Reno	3.69

TCP ENewReno achieved the highest average throughput in Fig I among the three variants Fig 2, demonstrating a significant improvement of approximately 34.5% over TCP Reno and around 34.6% over TCP NewReno. This performance gain can be attributed to its adaptive congestion window adjustment mechanism, which estimates real-time bandwidth based on acknowledgment (ACK) intervals. This allows TCP ENewReno to make more informed decisions about data transmission rates, avoiding both underutilization and congestion collapse.

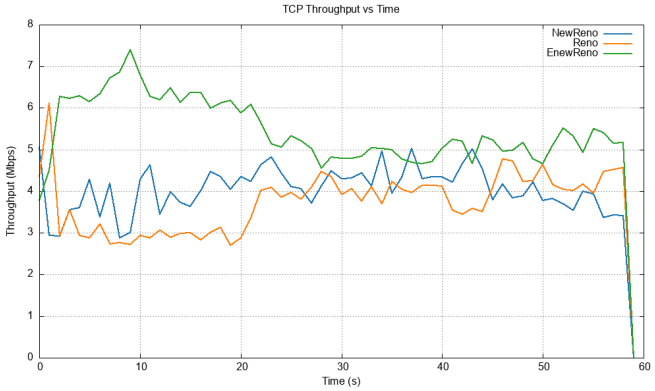

Fig. 2. Throughput Graph for Tcp-Variants

TCP NewReno also shows noticeable improvement over Reno, thanks to its enhanced fast recovery and retransmission strategies. However, it still lacks the real-time adaptability that ENewReno offers. TCP Reno, relying on basic congestion control and slow-start mechanisms, performs the least effectively, particularly under high-density and high-latency conditions that are common in next-generation networks.

The results suggest that adaptive congestion algorithms like ENewReno are better suited for environments with dynamic link characteristics—a common feature in mobile and high-density networks.

2) End-to-End Delay

Definition: End-to-End Delay measures the total time taken for a data packet to travel from the source to the destination. It includes all forms of delay such as transmission, propagation, queuing, and processing delays. This metric is crucial for real-time applications like video conferencing, and online gaming.

TABLE II
AVERAGE END-TO-END DELAY COMPARISON

TCP Variant	Avg. End-to-End Delay (ms)
TCP Reno	3.20661
TCP NewReno	3.20010
TCP ENewReno	3.20663

The delay for all three variants fluctuated between 0.5 ms and 3.2 ms, with TCP Reno showing the highest variability and occasional spikes. This instability is attributed to Reno's lack of proactive congestion handling and its reliance on conservative retransmission strategies, which result in delayed recovery during packet loss events.

TCP NewReno demonstrated moderate improvement, thanks to its enhanced fast retransmit and recovery mechanisms. However, it still exhibited occasional jitter under network fluctuations, affecting its delay consistency.

TCP ENewReno achieved the most stable delay performance, maintaining minimal deviation throughout the simulation in Table II. This consistency is a result of its feedback-based congestion control using the Delay-to-Throughput Ratio (DTR), which enables it to react promptly to network state changes. Such behavior is especially beneficial for latency-sensitive applications where consistent round-trip times are critical.

3) Congestion Window Size (CWND)

Definition: The Congestion Window (CWND) is a TCP state variable that limits the amount of data a sender can transmit before receiving an acknowledgment. A higher CWND value generally implies better bandwidth utilization, provided it does not lead to congestion.

TABLE III
AVERAGE CWND SIZE

TCP Variant	Average CWND Size (in segments)
TCP ENewReno	4.20
TCP Reno	4.17
TCP NewReno	2.64

TCP ENewReno achieved the largest average CWND size in Table III, indicating its superior capability to aggressively utilize available bandwidth without causing congestion. This is enabled by its real-time estimation of bandwidth, which permits dynamic CWND scaling Fig 3.

TCP Reno, while achieving a CWND nearly as high as ENewReno, lacks adaptability and thus risks congestion under bursty traffic. Its static congestion control does not account for current network conditions.

TCP NewReno, on the other hand, maintained a smaller CWND. Its conservative growth approach prioritizes stability over aggressiveness, resulting in less efficient utilization of network resources, particularly when capacity is underused.

A dynamically tuned CWND, as demonstrated by TCP ENewReno, is crucial for maintaining high throughput in modern networks that exhibit variable link quality and mobility.

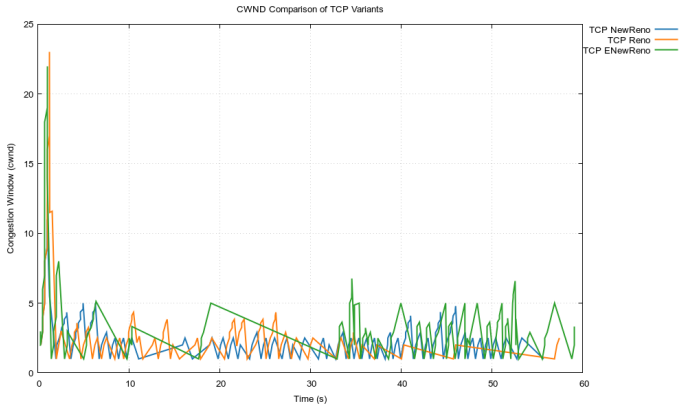

Fig. 3. Comparision of Congestion Window(CWND) for Tcp-Variants for Tcp-Variants

4) Jain's Fairness Index (JFI)

Definition Jain's Fairness Index measures the equality of resource allocation among multiple flows. A JFI value close to 1 indicates optimal fairness, where all users receive approximately equal bandwidth.

$$\text{JFI} = \frac{\left(\sum_{i=1}^{n} x_i\right)^2}{n \cdot \sum_{i=1}^{n} x_i^2}$$

Where:

- x_i = throughput of the i^{th} flow
- n = total number of flows

TABLE IV
JAIN'S FAIRNESS INDEX COMPARISON

TCP Variant	JFI Score
TCP ENewReno	0.716
TCP NewReno	0.715
TCP Reno	0.686

TCP ENewReno demonstrated the highest fairness index in Table IV, indicating that it can equitably allocate bandwidth across multiple concurrent flows. Its congestion window adapts in real-time, which helps avoid flow starvation or bandwidth hoarding.

TCP NewReno also shows commendable fairness in Fig 4, slightly below ENewReno, thanks to its refined retransmission logic. TCP Reno, however, had the lowest JFI, which implies inefficient bandwidth sharing—likely due to abrupt window reductions and slower recovery from congestion events.

V. CONCLUSION AND FUTURE WORK

The Work presents a simulation-based performance analysis of TCP Reno, NewReno, and ENewReno using NS2, with a focus on throughput and fairness. The results show that TCP ENewReno delivers superior performance by efficiently managing congestion through Explicit Congestion Notification (ECN), resulting in higher throughput and improved fairness as measured by Jain's Fairness Index. This underscores the

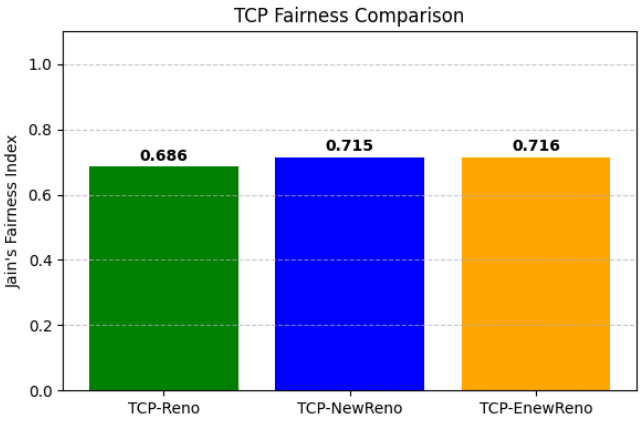

Fig. 4. Comparision of Jain's Fairness Index for Tcp-Variants

importance of advanced congestion control mechanisms in maintaining stable and balanced performance in multi-flow wired networks. The study also highlights the limitations of traditional TCP variants in handling fairness and suggests that enhancements like ECN can significantly boost performance. As future work, the approach can be extended to include real-time traffic scenarios, evaluation under dynamic topologies, and benchmarking against newer TCP variants like TCP BBR. Additionally, integrating machine learning-based congestion predictors and cross-layer optimization strategies could further improve adaptability and robustness in evolving network environments.

REFERENCES

[1] Michael Baumann and Cordula Petersen. Tcp and ntcp: a basic introduction. *RAYS-ROME-*, 30(2):99, 2005.

[2] Mark Allman, Vern Paxson, and Ethan Blanton. Tcp congestion control. https://www.rfc-editor.org/rfc/rfc5681, 2009. RFC 5681, Internet Engineering Task Force.

[3] Andrew S. Tanenbaum and David J. Wetherall. *Computer Networks*. Prentice Hall, 5th edition, 2011.

[4] Van Jacobson. Congestion avoidance and control. *ACM SIGCOMM Computer Communication Review*, 18(4):314–329, 1988.

[5] Amol P Pande and SR Devane. Study and analysis of different tcp variants. In *2018 Fourth International Conference on Computing Communication Control and Automation (ICCUBEA)*, pages 1–8. IEEE, 2018.

[6] Sally Floyd. Tcp reno: Slow start, congestion avoidance, fast retransmit, and fast recovery algorithms. In *RFC 2001*, 1997.

[7] Jitendra Padhye, Vlad Firoiu, Don Towsley, and Jim Kurose. Modeling tcp reno performance: A simple model and its empirical validation. *IEEE/ACM Transactions on Networking*, 8(2):133–145, 2000.

[8] Sally Floyd and Tom Henderson. Tcp newreno: Congestion control. *RFC 2582*, 1999.

[9] Alexander Afanasyev, Nick Tilley, Peter Reiher, and Leonard Kleinrock. An analysis of tcp congestion control with multiple algorithms. In *Proceedings of the 2010 IEEE/ACM International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems*, pages 327–336. IEEE, 2010.

[10] E. Elankovan, R. Arulselvan, and G. Manimaran. An enhanced tcp newreno for improving fairness and throughput in heterogeneous networks. *International Journal of Communication Systems*, 28(17):2700–2718, 2015.

[11] Raj Jain, Dah-Ming W. Chiu, and William R. Hawe. A quantitative measure of fairness and discrimination for resource allocation in shared computer systems. *DEC Research Report TR-301*, 1984.

[12] Svilen Ivanov, André Herms, and Georg Lukas. Experimental validation of the ns-2 wireless model using simulation, emulation, and real network. In *Communication in Distributed Systems-15. ITG/GI Symposium*, pages 1–12. VDE, 2007.

[13] Jon Postel and Joyce Reynolds. File transfer protocol (ftp), 1985. RFC 959.

[14] Ning Xu, Saurabh Rangwala, Krishna Kant Chintalapudi, Deepak Ganesan, Alan Broad, Ramesh Govindan, and Deborah Estrin. Tcp behavior across a high density cluster-based wireless sensor network. In *Proceedings of the 2005 ACM SIGCOMM workshop on Embedded networked sensors*, pages 13–19. ACM, 2005.

[15] Jeyanthi Rebecca Raj, Kennedy Arputharaj, and P Balasubramanie. Performance analysis of tcp variants using ns-2 simulator. *International Journal of Computer Science and Network Security (IJCSNS)*, 4(12):88–93, 2004.

[16] Hussain Al-Jobouri, Ali H. Al-Bayatti, and Iyad Bashi. Tcp congestion control algorithms: A survey and performance comparison. *International Journal of Computer Science and Network Security*, 7(12):94–102, 2007.

[17] Saleh M. Abdullah et al. Improving the tcp newreno congestion avoidance algorithm on 5g networks. *International Journal of Communication Systems*, 33(18):e4567, 2020.

[18] Kevin Fall and Kannan Varadhan. The ns manual (formerly ns notes and documentation). Technical report, UC Berkeley, LBL, USC/ISI, and Xerox PARC, 2000. Available at https://www.isi.edu/nsnam/ns/.

[19] Tomas Bonato, Yu Zhang, Ang Li, Akanksha Arora, et al. Fastflow: A congestion control algorithm for machine learning workloads in datacenter networks. *Proceedings of the ACM SIGCOMM*, 53(3):1–17, 2023.

[20] Neal Cardwell, Yuchung Cheng, C. Stephen Gunn, Soheil Yeganeh, and Van Jacobson. Bbr: Congestion-based congestion control. In *Communications of the ACM*, volume 60, pages 58–66, 2016.

[21] Radhika Mittal et al. Swift: Delay is not enough in datacenters. In *Proceedings of the Annual Conference of the ACM Special Interest Group on Data Communication (SIGCOMM)*, pages 199–212, 2020.