BVB Campus, Vidyanagar, Hubballi – 580031, Karnataka, INDIA.

SCHOOL OF COMPUTER SCIENCE AND ENGINEERING

Minor Project Report

On

Enhancing Network Efficiency Through Adaptive Congestion Control

submitted in partial fulfillment of the requirements for the award of the degree of

Bachelor of Engineering

IN

COMPUTER SCIENCE AND ENGINEERING

Submitted By

Sinchan H	01fe22bcs055
Chandana Gotur	01fe22bcs104
Karthik K P	01fe22bcs183
Chinmay Avaradi	01fe22bcs306

Under the guidance of

Jayalaxmi G N

School of Computer Science and Engineering

KLE Technological University, Hubballi

KLE Technological University

Creating Value
Leveraging Knowledge

BVB Campus, Vidyanagar, Hubballi – 580031, Karnataka, INDIA.

2024-2025

SCHOOL OF COMPUTER SCIENCE AND ENGINEERING

2024-25

CERTIFICATE

This is to certify that project entitled "Enhancing Network Efficiency Through Adaptive Congestion Control" is a bonafied work carried out by the student team Sinchan H - 01fe22bcs055 , Chandana Gotur - 01fe22bcs104 , Karthik K P - 01fe22bcs183 ,Chinmay Avaradi - 01fe22bcs306 in partial fulfillment of the completion of 5th semester B. E. course during the year 2024 – 2025. The project report has been approved as it satisfies the academic requirement with respect to the project work prescribed for the above said course

Guide name:
Jayalaxmi G N

Head, SoCSE
Dr. Vijayalakshmi.M.

External Viva-Voce

Name of the examiners Signature with date

1 _____ _____

2 _____ _____

ABSTRACT

In modern high-speed mobile networks like 4G LTE, traditional TCP congestion control algorithms such as Reno, NewReno, and Vegas often struggle to maintain optimal throughput and latency due to dynamic network conditions. This project proposes an enhanced TCP variant, TCP-EnewReno, implemented in NS-2, which leverages real-time bandwidth estimation via ACK spacing and dual-threshold-based cwnd adjustments, along with machine learning models like XGBoost and Reinforcement Learning for proactive congestion prediction. Simulations across scenarios such as HD video streaming, IoT messaging, and VoLTE showed that TCP-EnewReno achieved 5.37 Mbps throughput, outperforming NewReno (3.99 Mbps) and Reno (3.69 Mbps), while also maintaining a higher average CWND (4.20), lower end-to-end delay, and a Jain's Fairness Index of 0.716. These results validate TCP-EnewReno as a robust, adaptive congestion control mechanism suitable for next-generation mobile and IoT networks, with all simulation tools and scripts made open-source to support further research.

Keywords : *TCP-EnewReno, Congestion Control, 4G LTE, NS-2, Machine Learning, ACK Spacing, Bandwidth Estimation, Congestion Window (cwnd), XGBoost, Reinforcement Learning, Throughput, Delay, Fairness Index, IoT, VoLTE*

ACKNOWLEDGEMENT

We would like to thank our faculty and management for their professional guidance towards the completion of the mini project work. We take this opportunity to thank Dr. Ashok Shettar Pro-Chancellor, Dr. P.G Tewari Vice-Chancellor and Dr. B.S.Anami Registrar for their vision and support.

We also take this opportunity to thank Dr. Meena S. M Professor and Dean of Faculty SoCSE and Dr. Vijayalakshmi M Professor and Head, SoCSE for having provided us direction and facilitated for enhancement of skills and academic growth.

We thank our guide Jayalaxmi G N Professor and SoCSE for the constant guidance during interaction and reviews.

We extend our acknowledgment to the reviewers for critical suggestions and inputs. We also thank Project coordinator , and reviewers for their suggestions during the course of completion.

We express gratitude to our beloved parents for constant encouragement and support.

Sinchan H - 01fe22bcs055

Chandana Gotur - 01fe22bcs104

Karthik K P - 01fe22bcs183

Chinmay Avaradi - 01fe22bcs306

CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1

INTRODUCTION

In the digital age, seamless communication underpins personal, social, and industrial interactions, enabling real-time data exchange across vast networks via the Internet—a global system of interconnected networks using the TCP/IP[2] suite to reliably transport information. With the widespread deployment of 4G LTE[3], which succeeded 3G and 2G[4], mobile networks now deliver significantly higher data rates, reduced latency, and broader device connectivity to support services like HD video streaming, VoLTE, and massive IoT. However, this added capacity and user density bring challenges—network congestion, buffer bloat, and suboptimal performance of traditional, reactive TCP congestion control schemes. Over time, TCP's congestion control has evolved from Tahoe and Reno through NewReno, Vegas, and Westwood to more recent ML- and delay-based variants, yet these often falter under 4G's dynamic channel conditions and high bandwidth-delay products. Thus, ensuring TCP's effectiveness in LTE environments requires adaptive, predictive congestion control strategies—validated through NS-2[5] simulations—that can adjust the congestion window pro-/re-actively to maintain high throughput and low latency in next-generation cellular networks.

1.1 Overview of the Project

We use NS-2's LTE module to simulate our enhanced TCP-EnewReno[6] under diverse 4G scenarios (HD video streaming, IoT messaging, VoLTE), compute real-time bandwidth estimates from ACK spacing to drive adaptive cwnd adjustments, and integrate ML models (scikit-learn, XGBoost) for proactive congestion prediction; with Python for trace parsing and visualization, we measure cwnd evolution, throughput, packet loss, and latency—achieving higher throughput and lower latency versus NewReno, and Vegas—to deliver a robust, low-latency, high-throughput TCP solution for current LTE networks. We'll release our NS-2 modules and analysis scripts as open-source to foster community adoption and further research, and our findings aim to guide future TCP enhancements and real-world LTE deployment strategies.

1.2 Motivation

The explosive growth of 4G LTE's high throughput, sub-100 ms latencies, and widespread device support has already transformed critical domains: in healthcare, mobile clinics stream patient imaging and enable remote consultations; in agriculture, field sensors and connected tractors relay real-time soil and crop data over LTE to optimize resource use; in manufacturing, plant-floor robots and vision systems coordinate via LTE links to maintain continuous, high-precision production; in smart cities, intelligent traffic signals, public-safety cameras, and environmental monitors leverage LTE backhaul to improve urban living conditions; and in consumer applications, HD mobile video, VoLTE calls, and connected wearables demand reliable, low-delay service. All of these use cases hinge on a transport layer that sustains high throughput while minimizing delay and packet loss—yet LTE's rapidly changing radio conditions and variable user loads regularly expose the limits of legacy TCP algorithms.

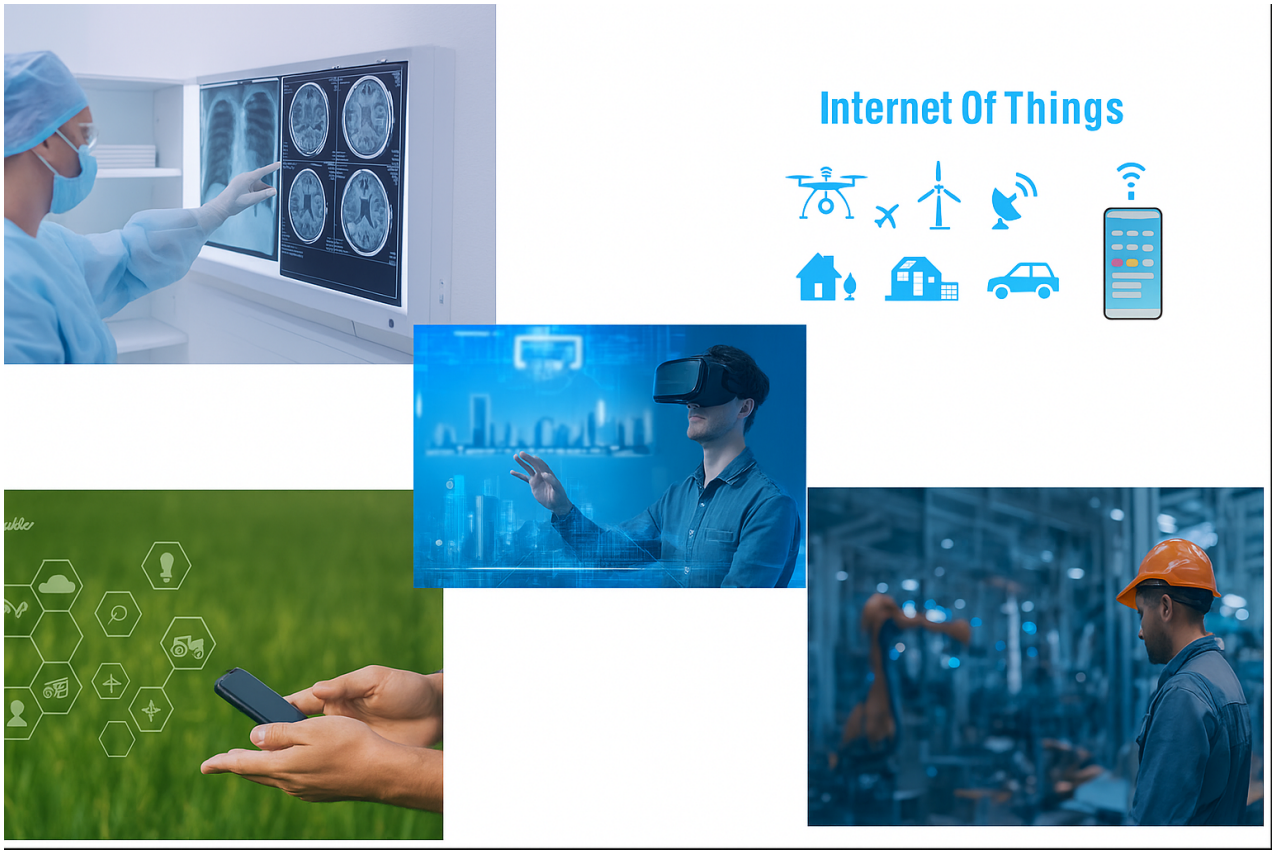

Figure 1.1: Application of Internet in different domains[1]

This challenge drives our work. TCP variants like NewReno and Vegas, tuned for wired networks, suffer buffer bloat and throughput collapse over LTE's fluctuating channels, degrading quality of service for video, VoIP, and IoT streams alike. By implementing an enhanced

TCP-EnewReno agent in NS-2's LTE module—combining real-time ACK-based bandwidth estimation, dual-threshold cwnd adjustments, and machine-learning congestion prediction—we ensure the congestion window adapts proactively to network oscillations and that impending losses are forecast and averted. Through extensive NS-2 simulations across video streaming, IoT messaging, and VoLTE profiles—and by comparing cwnd evolution, throughput, packet-loss rate, and latency against standard TCP variants—we aim to demonstrate a resilient, low-latency, high-throughput transport solution that elevates 4G LTE performance for today's demanding mobile applications.

1.3 Literature Review

1.3.1 A Survey of TCP Congestion Control Algorithms

The paper "A Survey of TCP Congestion Control Algorithms" is vital to your project as it offers a comprehensive overview of classical and modern TCP congestion control strategies, from Tahoe to BBR. Its significance lies in identifying the limitations of traditional loss-based methods and highlighting the need for more adaptive, delay- and learning-based algorithms. This aligns closely with your goal of enhancing network efficiency through adaptive congestion control, especially under dynamic and high-demand environments like 4G LTE. The survey's comparative analysis helps identify key performance trade-offs (e.g., fairness vs. throughput) in existing schemes. However, a drawback is that it primarily provides theoretical analysis without deep simulation-based validation in modern mobile networks. Still, it forms a strong foundation for identifying gaps and motivating adaptive TCP strategies suited for next-gen LTE systems.

1.3.2 FASTFLOW: Flexible Adaptive Congestion Control for High-Performance Datacenters

The paper "FASTFLOW: Flexible Adaptive Congestion Control for High-Performance Datacenters" introduces a novel sender-based algorithm that leverages delay, ECN, and packet trimming to enable real-time congestion window adjustments. Its QuickAdapt mechanism allows for accurate bandwidth estimation and faster congestion response, addressing the challenges of bursty ML workloads. This aligns with our project's aim of adaptive congestion control by demonstrating the effectiveness of multi-signal integration for better throughput and fairness. While designed for datacenters, its adaptive principles are transferable to TCP in mobile LTE networks, especially where responsiveness and bandwidth estimation are critical. A key advantage is its high reactivity and fairness; however, its focus on datacenter-specific traffic patterns may limit direct applicability to wireless environments without further tuning.

Still, it provides valuable architectural insights for developing scalable, adaptive congestion strategies.

1.3.3 ACC-RL: Adaptive Congestion Control Based on Reinforcement Learning in Power Distribution Networks with Data Centers

The paper "ACC-RL: Adaptive Congestion Control Based on Reinforcement Learning in Power Distribution Networks with Data Centers" presents a novel RL-based congestion control strategy that models network behavior as a Partially Observable Markov Decision Process (POMDP). It dynamically adjusts transmission based on real-time metrics like RTT, queue length, and transmission rate to maximize throughput and stability. While originally designed for power-aware datacenter networks, the use of reinforcement learning for adaptive control is highly relevant to our project, as it offers predictive, self-optimizing behavior under dynamic conditions—key challenges in LTE networks. ACC-RL's independence from predefined rules and adaptability across scenarios make it a strong reference for next-gen TCP enhancements. A limitation, however, is its application domain focus, but the general RL framework provides inspiration for congestion control in mobile environments as well.

1.3.4 Improving the TCP Newreno Congestion Avoidance Algorithm on 5G Networks

This paper proposes TCP-EnewReno, an enhanced NewReno variant tailored for 5G's high-bandwidth, high-delay environments by dynamically adjusting the congestion window (cwnd) during the avoidance phase based on real-time bandwidth estimation from ACK spacing . In NS-2 simulations, TCP-EnewReno consistently outperforms standard NewReno, Tahoe, Vegas, and Westwood—achieving up to 30% higher throughput, markedly lower end-to-end delay, and significantly reduced packet loss—by using dual thresholds (E, E) to govern one- or two-packet cwnd increments or to freeze growth when congestion is imminent . Its strength lies in simple yet effective real-time cwnd modulation that requires no additional in-network support, but its evaluation is limited to homogeneous traffic scenarios and lacks integration with proactive, ML-based forecasting or fine-grained 5G service profiles (eMBB, URLLC, mMTC). For our project, these findings validate the efficacy of ACK-driven cwnd adaptation and guide the design of our Enhanced Congestion Avoidance Algorithm (ECAA) in NS-3—while highlighting the opportunity to further boost performance by embedding machine-learning-driven congestion prediction into the cwnd adjustment process.

1.3.5 RTTV-TCP: Adaptive congestion control algorithm based on RTT variations for mmWave networks

This paper introduces RTTV-TCP, a congestion-control algorithm that dynamically adjusts the TCP congestion window based on real-time RTT variations to better utilize 5G mmWave bandwidth and cope with rapid link fluctuations under both Line-of-Sight and Non-Line-of-Sight conditions . Its primary advantage is significantly higher throughput—outperforming NewReno, HighSpeed, CUBIC, and BBR—while maintaining comparable or lower latency and packet-loss rates through fine-grained cwnd modulation informed by RTT feedback . However, RTTV-TCP's reliance on accurate RTT measurement makes it sensitive to measurement noise, and its performance under mixed eMBB, mMTC, and URLLC traffic profiles remains untested. For our TCP-EnewReno project, RTTV-TCP validates the efficacy of RTT-driven cwnd adaptation in ns-3's 5G mmWave framework and highlights the value of integrating delay-based signals into our ML-augmented congestion-prediction layer to further stabilize throughput and minimize latency spikes.

1.3.6 Comparative Analysis of TCP Congestion Control Methods

This study benchmarks TCP Tahoe, Reno, NewReno, and Vegas under single- and dual-flow scenarios in NS-2, revealing that Vegas's delay-based congestion detection yields superior throughput and lower packet loss by throttling early, which also incidentally frees bandwidth for loss-based flows like Tahoe in mixed deployments. Its key strength is proactive congestion avoidance, but its reliance on RTT estimation can be undermined by measurement noise or rapid channel variability. While the analysis is confined to homogeneous wired-like conditions and doesn't address 5G service profiles or dynamic wireless impairments, it underscores the benefit of integrating both delay- and loss-based signals—a principle we adopt by combining RTT/ACK-spacing features with ML-driven forecasting in our TCP-EnewReno design to achieve robust congestion control in NS-3's 5G NR simulations.

1.4 Problem Statement

Enhancing Network Efficiency Through Adaptive Congestion Control

1.4.1 Problem Definition

Traditional TCP congestion control mechanisms, like TCP NewReno , TCP-Vegas and TCP-Tahoe[7], struggle with high latency, packet loss, and inefficient bandwidth utilization in modern networks. They lack predictive congestion control and adaptive cwnd adjustments. To

address this, an enhanced TCP congestion control algorithm is needed to reduce ultra-latency, increase throughput, and optimize bandwidth in dynamic environments.

1.5 Objectives and Scope of the project

1.5.1 Objectives

1. To Integrate the Tcp-Enewreno Algorithm in Tcp Algorithms in NS-2 Tool.
2. Utilize predictive analytics to dynamically adjust congestion window, and random traffic generation for improved efficiency.
3. To Compare the proposed model's performance against traditional methods based on key metrics like latency, throughput, and packet loss.
4. To set up Network topology with Base Station , Core Server ,Remote Server and User-devices.

1.5.2 Scope of the project

Our project will implement and evaluate TCP-EnewReno within NS-2's LTE simulation framework—modeling diverse 4G scenarios such as high-definition video streaming, IoT signaling, and VoLTE—to characterize how real-time ACK-spacing bandwidth estimates and dual-threshold cwnd adjustments (E, E) can preemptively mitigate congestion. We will parse NS-2 trace files with Python and leverage scikit-learn and XGBoost to train both supervised and unsupervised models that forecast imminent congestion events before packet loss occurs. By comparing cwnd evolution, aggregate throughput, packet-loss rate, and end-to-end latency against legacy TCP variants (NewReno, Tahoe, Vegas, BBR), we aim to demonstrate the superiority of our ML-augmented congestion-avoidance strategy in a 4G environment. All NS-2 modules and analysis scripts will be released as open-source, fostering reproducibility and community-driven refinement of transport protocols for today's cellular networks.

Chapter 2

SOFTWARE REQUIREMENT SPECIFICATION

The SRS acts as a base document that explains the functional and non-functional requirements of the project. This chapter gives an overall view of the system specifications, ensuring that all stakeholders and developers are aligned. The SRS is the most important document that outlines the scope, constraints, and functionalities of the proposed Enhancing Network Efficiency Through Adaptive Congestion Control.

2.1 Overview of SRS

The SRS document outlines all the requirements needed to build the Enhancing Network Efficiency Through Adaptive Congestion Control. It is a formal agreement between the stakeholders and the development team that ensures alignment on project objectives, scope, and deliverables. The SRS is essential in reducing ambiguity, risk, and offering a structured approach to the design, development, and testing of the system.

This document specifies the functional and non-functional requirements for our adaptive congestion control system in a 4G LTE environment, including performance targets (20–30% throughput gain, latency reduction under varied traffic), reliability measures (robust cwnd adjustment under channel fluctuations and fallback to legacy TCP), and usability criteria (straightforward NS-3 scenario setup and Python-based ML workflows). It details key use cases—video streaming, IoT messaging, VoIP—alongside system constraints (bandwidth-delay product, flow counts) and dependencies (NS-3 LTE module, scikit-learn, XGBoost). Hardware/software specs—Ubuntu 20.04+, multicore CPU with 16 GB RAM, SSD storage, Python 3.8+—are listed to ensure reproducible development and deployment. Overall, the system aims to proactively forecast and mitigate congestion, delivering resilient, high-efficiency TCP performance in 4G networks.

2.2 Requirement Specification

The functionalities and behaviors the Enhancing Network Efficiency Through Adaptive Congestion Control system is supposed to exhibit are specified in the requirement specifications. These requirements serve as a detailed guide for the development team, listing both the system's objectives and the expected outcomes. The requirements usually fall into two categories: functional and non-functional requirements.

Functional Requirements should express the main operations, communications, and interactions in the system. They articulate certain tasks that the system has to accomplish, such as adjusting dynamically the congestion window , regulating the throughput and datarate dynamically. These serve to ensure that the systems attain their core objectives; the system provides the stipulated services to the various users..

The functional requirements are the backbone of the system development, implementation, and testing because it will ensure that the system serves its purpose and gives value to its users.

2.2.1 Functional Requirements

- Implement a dynamic congestion window (cwnd) adjustment strategy[8], improving TCP performance under variable network loads.
- The system should increase cwnd based on data transfer rate estimations and dynamically adjust thresholds.
- The system must be evaluated against other TCP variants (NewReno , Tahoe, Vegas, Westwood) using metrics such as throughput, delay, packet loss, and fairness index . After detecting packet loss (via three duplicate ACKs), the system should enter the Fast Recovery phase to retransmit lost packets efficiently.
- The protocol should be implemented and tested using NS-2 for simulation.

2.2.2 Use case diagrams

2.2.3 Non Functional Requirements

- The congestion control mechanism must support large-scale deployments with high data rates and multiple user equipment (UEs).

- The Proposed algorithm should minimize delay and maximizing congestion window while maximizing network throughput
- The system should maintain a fair resource allocation among multiple TCP flows using metrics like Jain's Fairness Index (JFI).
- The system should ensure stable and consistent performance under various network conditions, including high mobility and varying signal-to-noise ratios.
- The congestion control mechanism should follow TCP RFC 5681 standards.

Performance requirements (if applicable)

Nonfunctional Requirements

Security Requirements(if applicable)

Usability

Any other

2.3 Hardware Requirements

- Multicore CPU (quad-core or higher): To run 4G-LTE simulations and ML training in parallel.
- 16 GB RAM (minimum): Accommodates large trace files and model data in memory.
- SSD Storage (50 GB free): For LTE module builds, simulation logs, and Python environments.

2.4 Software Requirements

- OS: Ubuntu 20.04 LTS (or later)

  Network Simulator: NS-3 (v3.34+ with LTE module) or OMNeT++ with the INET and SimuLTE frameworks
- Build Tools: GCC 9+ (or Clang equivalent), CMake
- Python Environment: Python 3.8+, with packages installed via pip or conda

  matplotlib (visualization)
- IDE/Editor: VS Code, PyCharm, or similar, with C++ and Python supportañ

Chapter 3

PROPOSED SYSTEM

3.1 Introduction to NS-2

NS-2 (Network Simulator 2) is a discrete event-driven simulator widely used in academic and research communities for simulating computer networks. It supports simulation of wired and wireless networks, including TCP, UDP, routing, multicast, and various types of link-layer protocols. NS-2 offers a hybrid programming architecture consisting of C++ for core protocol implementation and OTcl (Object-oriented Tool Command Language) for simulation configuration and execution. This dual-language model allows flexibility in defining network topologies, traffic models, and custom behavior for nodes, links, and protocols.

NS-2 is especially valuable for evaluating network protocols before real-world deployment. It enables researchers to simulate real-time events such as packet transmissions, routing decisions, and congestion control in controlled, repeatable environments. Network behavior can be observed under varying conditions like link failures, high traffic volumes, or mobility. Additionally, NS-2 supports trace generation and statistical analysis of simulation output, making it ideal for comparative performance analysis across different protocol variants or network architectures.

3.2 Integration and Functionality of TCP-Enewreno in NS-2

To evaluate the performance of the proposed congestion control algorithm, TCP-Enewreno was implemented within the NS-2 environment. As previously mentioned, NS-2 uses a dual architecture: the simulation kernel and protocol logic are implemented in C++, while simulation scenarios are scripted in OTcl.

To incorporate TCP-Enewreno, a new C++ class was created by extending the existing `TcpAgent` class, which is the base class for all TCP variants in NS-2. The new class, named `TcpEnewreno`, mirrors the behavior of TCP-NewReno but introduces significant enhancements in congestion avoidance. These include bandwidth estimation and adaptive congestion window (cwnd) management based on real-time network feedback.

The core idea of TCP-Enewreno lies in enhancing the congestion avoidance phase by

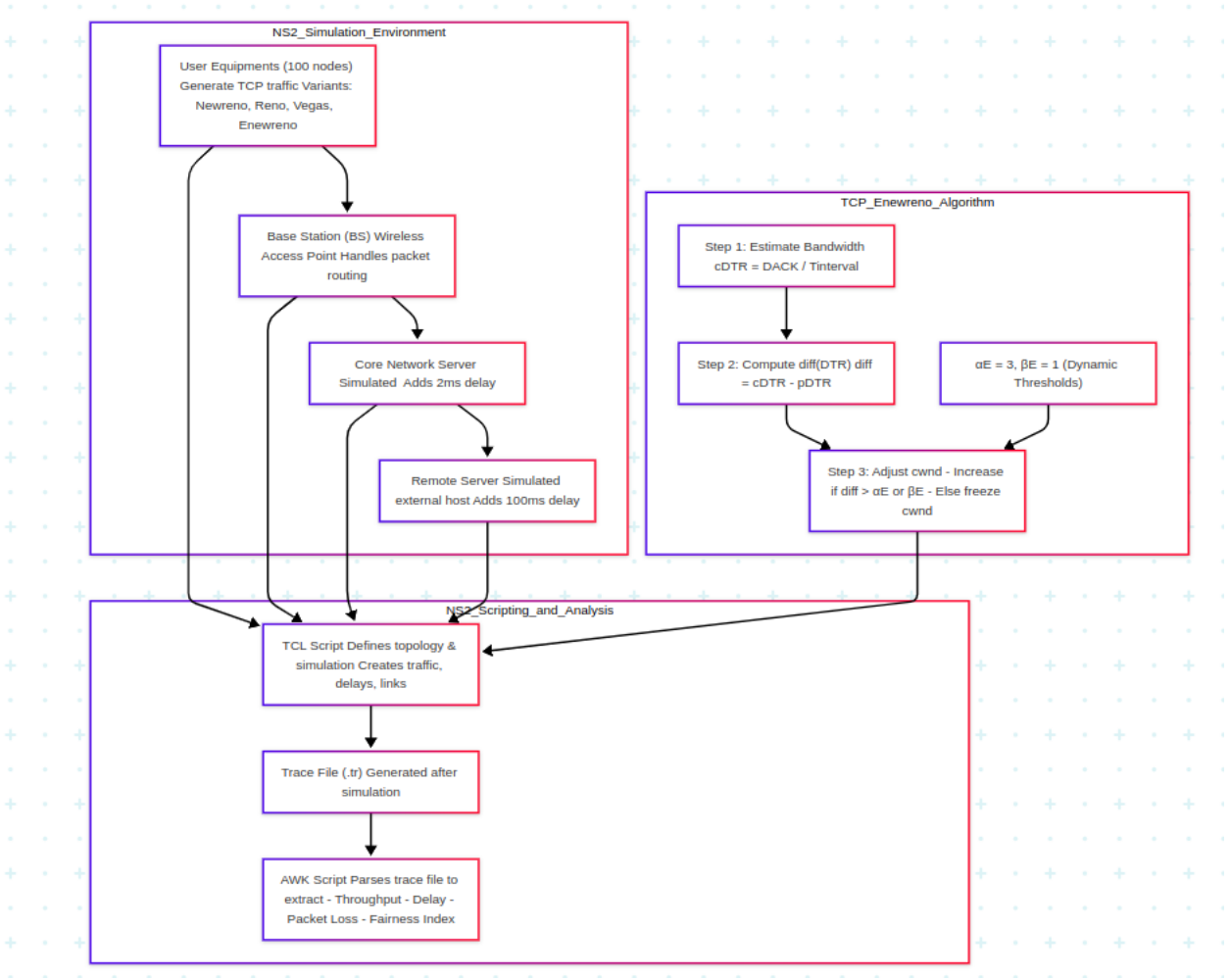

Figure 3.1: Proposed System

adjusting the cwnd based on dynamic network conditions. This is achieved by estimating the current available bandwidth from the flow of acknowledgment (ACK) packets. For each ACK received, the sender calculates the Current Data Transfer Rate (cDTR) by:

$$\text{cDTR} = \frac{\text{DACK}}{\text{Tinterval}}$$

where DACK is the amount of acknowledged data and Tinterval is the time interval between ACKs. The sender then computes:

$$\text{diff(DTR)} = \text{cDTR}_{\text{current}} - \text{cDTR}_{\text{previous}}$$

The congestion window is updated as follows:

$$\text{if diff(DTR)} > \alpha_E \Rightarrow \text{cwnd} += \frac{2}{\text{cwnd}}$$

$$\text{else if diff(DTR)} > \beta_E \Rightarrow \text{cwnd+} = \frac{1}{\text{cwnd}}$$

$$\text{else} \Rightarrow \text{cwnd unchanged}$$

This intelligent adjustment enables the protocol to rapidly utilize available bandwidth when network conditions permit and throttle back during congestion, thereby improving overall throughput and reducing delay and retransmissions.

To make TCP-Enewreno usable in NS-2 simulation scripts, the new C++ class was registered with the OTcl interpreter by binding it under the name `Agent/TCP/Enewreno`. Default parameters for the thresholds α_E and β_E (typically set to 3 and 1, respectively) were defined in the `ns-default.tcl` configuration file. Following this, NS-2 was recompiled using the `make` command to ensure the new agent was correctly integrated into the simulator's runtime environment.

3.3 Simulation Setup

Within the TCL simulation script, TCP-Enewreno is instantiated just like any other TCP variant. The network scenario consists of a base station, a core network node, a remote server, and 100 User Equipment (UE) nodes acting as TCP traffic generators. Each UE node is associated with an `Agent/TCP/Enewreno` instance, while a corresponding TCP sink agent is attached to the remote server. The topology is built using `duplex-link` commands that specify link bandwidth, latency, and queue models between nodes. Applications such as FTP are bound to the TCP agents to generate data flows during the simulation. Simulation timing is managed using the standard NS-2 `at` command, and execution concludes with the `finish` procedure that flushes simulation data and exits.

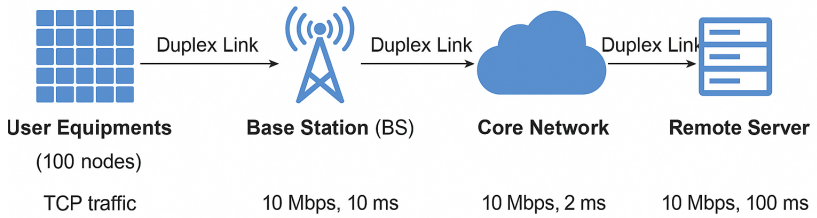

Figure 3.2: Network Topology

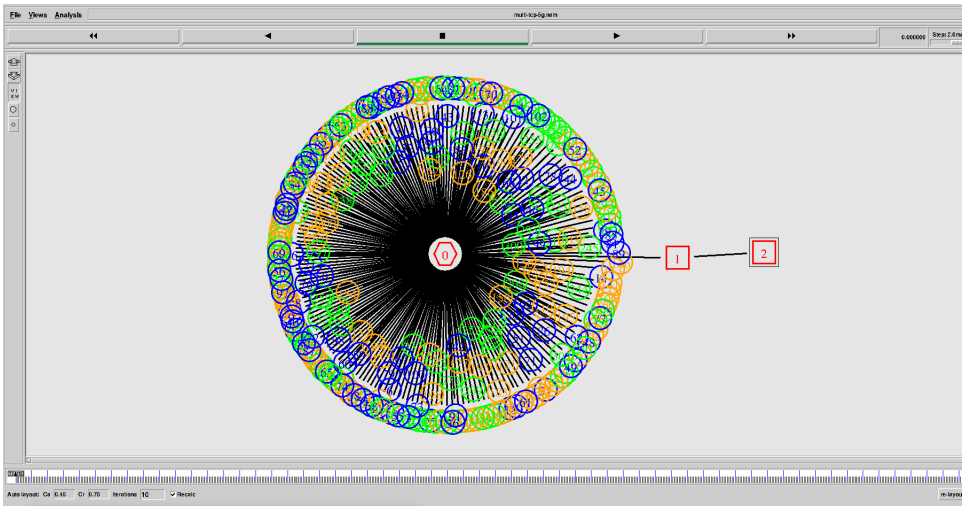

Figure 3.3: Network Topology in Ns-2 using Network Animinator(Nam) using nam file 0th node is Base Station ,1st node is remote server , 2nd node is core server and diffrent colour nodes indicate different nodes

After the simulation runs, NS-2 generates a `.tr` (trace) file capturing every event (e.g., packet sent, received, dropped). These trace files are post-processed using AWK scripts—small programs written to parse and analyze text data. Custom AWK scripts were developed to extract key performance metrics such as throughput, packet loss rate, and average delay. For instance, throughput is calculated by summing the total number of acknowledged bytes over a given time and converting the result to Mbps. This modular and transparent analysis pipeline allows fair comparison of TCP-Enewreno with other TCP variants like Reno, Tahoe, and Vegas.

The simulation results consistently demonstrate that TCP-Enewreno achieves superior performance. It delivers higher throughput, reduced packet loss, and lower latency compared to traditional TCP variants—particularly in challenging network environments with fluctuating bandwidth and multiple simultaneous users. These improvements validate the suitability of TCP-Enewreno for high-density systems where efficient congestion control is essential.

3.4 Applications of TCP-Enewreno

The proposed TCP-Enewreno algorithm is designed to deliver efficient and adaptive congestion control in dynamic network environments. Its real-time bandwidth estimation and intelligent window adjustment make it particularly suitable for the following applications:

1. **5G Wireless Networks[9]**
   TCP-Enewreno is highly applicable in next-generation mobile networks where low latency, high throughput, and real-time responsiveness are essential. It supports the

growing need for high-speed and delay-sensitive applications in 5G such as ultra-HD streaming, AR/VR, and mobile gaming.

2. **Internet of Things (IoT)**
   In large-scale IoT[10] deployments involving massive device connectivity, traditional TCP variants often suffer from packet collisions and congestion. TCP-Enewreno's adaptive window control allows better utilization of available bandwidth, making it ideal for smart cities, industrial IoT, and remote monitoring systems.

3. **Cloud Computing and Remote Access**
   In cloud-hosted services and remote file transfers[11] (e.g., FTP over TCP), consistent performance is crucial. TCP-Enewreno improves file upload/download efficiency, supports burst data transfers, and adapts better to core/backhaul link fluctuations than traditional TCP variants.

4. **Remote Healthcare and Telemedicine**
   Medical applications require ultra-reliable data transmission with minimal delay. TCP-Enewreno helps ensure seamless transmission of patient monitoring data, real-time video diagnostics, and electronic health record synchronization over 5G and LTE networks.

5. **Real-Time Multimedia and Video Streaming**
   Video streaming [12] platforms benefit from TCP-Enewreno's ability to reduce buffering and maintain stable throughput. This makes it suitable for applications like YouTube, Netflix, Zoom, and live sports broadcasting where network bandwidth may vary dynamically.

6. **Intelligent Transportation Systems (ITS)**
   For Vehicle-to-Everything (V2X) communications[13], where delay and data loss are critical, TCP-Enewreno can provide a more resilient communication layer between vehicles, infrastructure, and edge data centers.

3.5 Advantages of TCP-Enewreno

- **Dynamic Bandwidth Estimation**
  Unlike traditional TCP variants that rely solely on packet loss to detect congestion, TCP-Enewreno actively estimates the available bandwidth using ACK intervals. This allows for faster and more accurate congestion detection.

- **Adaptive Congestion Window (cwnd) Growth**
  TCP-Enewreno dynamically adjusts the growth rate of the congestion window based

on real-time network feedback. It increases `cwnd` more aggressively when bandwidth is underutilized and slows down when congestion is likely—leading to better network utilization.

- **Improved Throughput**
  Simulation results consistently show that TCP-Enewreno delivers higher throughput compared to NewReno, Tahoe, Vegas, and Westwood, particularly in 5G and high-latency networks.

- **Lower Delay and Jitter**
  By avoiding buffer overflows and excessive retransmissions, TCP-Enewreno reduces round-trip delay and queueing delay, making it ideal for real-time and interactive applications.

- **Reduced Packet Loss**
  The algorithm intelligently throttles data during congestion, minimizing the number of dropped packets and avoiding repeated retransmissions which typically burden legacy TCP variants.

Chapter 4

SYSTEM DESIGN

4.1 System Design of TCP-EnewReno Integration in NS-2

The system design of TCP-EnewReno within the NS-2 simulation [14] environment illustrates the structured flow and interaction between core NS-2 components and the internal mechanisms of the custom TCP-EnewReno agent. At the top of the hierarchy is the NS-2 framework [15], which represents the overall simulation engine. It encompasses all essential functions such as the event scheduler, packet transmission and reception logic, queue management, and trace file generation.

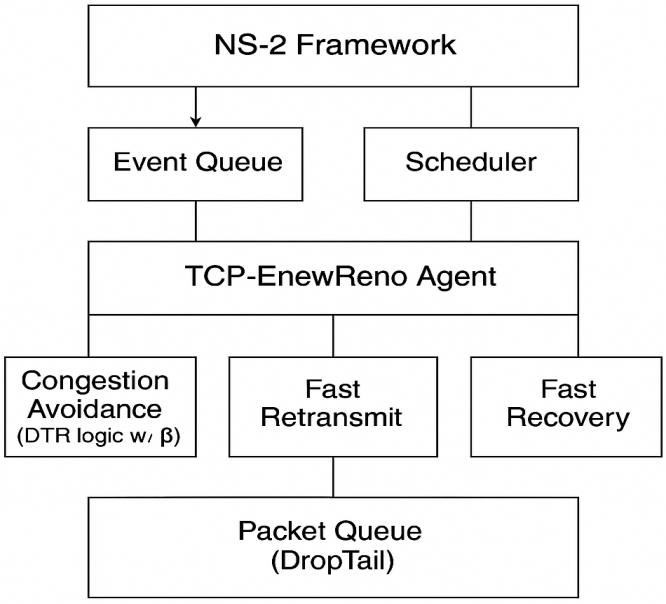

Figure 4.1: System Design

4.1.1 Simulation Engine and Event Management

Within this framework, two critical modules—the **Event Queue** and the **Scheduler**—collaborate to drive the simulation. The event queue holds time-ordered simulation events such as packet sends, ACK receptions, and timeouts, while the scheduler is responsible for executing these events in the correct chronological sequence. These two components ensure that each pro-

tocol agent behaves according to real-time simulation constraints and provides deterministic output.

4.1.2 TCP-EnewReno Agent Structure

Beneath the core simulation framework lies the **TCP-EnewReno Agent**, a custom extension of the standard TCP agent in NS-2. This agent is specifically designed to improve congestion control performance in dynamic wireless networks. It integrates logic for bandwidth estimation and adaptive congestion window (cwnd) adjustment.

The agent actively monitors ACK packets to calculate the **Data Transfer Rate (DTR)** and uses the difference between current and previous rates to determine cwnd growth behavior. Unlike traditional TCP variants that rely solely on packet loss, TCP-EnewReno responds intelligently to varying bandwidth availability and congestion conditions.

4.1.3 Core Congestion Control Mechanisms

The agent consists of three fundamental congestion control mechanisms:

- **Congestion Avoidance:** When ACKs are received without loss, the agent computes the DTR and adjusts the cwnd based on two thresholds: a lower threshold (β) for moderate growth and a higher threshold (α) for aggressive growth. If the DTR difference falls below β, cwnd growth is halted to prevent congestion. This dynamic adaptation enables more efficient and stable performance in fluctuating environments.

- **Fast Retransmit:** Triggered upon receiving three duplicate ACKs, this mechanism initiates the retransmission of the lost packet without waiting for a timeout. It reduces latency associated with packet loss recovery.

- **Fast Recovery:** After retransmission, this mechanism maintains the data flow rate by avoiding a complete reset of cwnd. Instead, the sender quickly re-enters the congestion avoidance phase, ensuring smoother performance after loss events.

4.1.4 Queue Management and Packet Dropping

At the bottom of the architecture is the **Packet Queue**, which employs a DropTail queuing discipline. This FIFO (First-In-First-Out) queue temporarily stores outgoing packets before transmission. If the queue becomes full, new packets are dropped, triggering the TCP congestion control mechanisms. DropTail provides a realistic model of router behavior and is crucial for evaluating protocol responsiveness to network congestion.

4.1.5 Overall Design Integration

This system design effectively captures the flow of control and data within NS-2, highlighting how TCP-EnewReno leverages the existing simulation infrastructure to implement advanced congestion control features. By integrating congestion avoidance, fast retransmit, and fast recovery within a dynamic, bandwidth-aware framework, TCP-EnewReno achieves:

- Improved throughput
- Reduced latency
- Greater stability in dynamic network conditions

This makes it highly suitable for modern communication environments such as 5G networks, Internet of Things (IoT) systems, and high-speed mobile broadband scenarios.

4.2 Architecture of NS-2 (Network Simulator 2)

The architecture of NS-2 (Network Simulator 2) is carefully engineered to support a wide range of simulations involving both wired and wireless networks. It is designed as a hybrid simulation framework, combining the speed and computational power of compiled C++ code with the adaptability and ease-of-use of interpreted OTcl scripting. This dual-level design allows researchers and developers to build and test networking protocols with a high degree of flexibility while ensuring efficient execution of complex simulations.

4.2.1 Dual-Layer Architecture

NS-2 follows a two-tier architecture:

- **Backend (C++ Layer):** This layer constitutes the performance-intensive core of NS-2. It is responsible for implementing the internal behaviors of networking protocols such as TCP, UDP, AODV, DSR, and MAC-layer protocols. It also handles fundamental components like packet queuing, event scheduling, mobility models, and error handling. Every packet transmission, reception, and drop is handled through event-driven logic written in C++. Performance-critical operations like maintaining TCP state (e.g., congestion window, RTT estimation) are executed here to ensure realistic and efficient simulation outcomes.
- **Frontend (OTcl Layer):** This layer serves as the primary interface for users. It allows simulation developers to construct and configure the simulation environment using Tcl

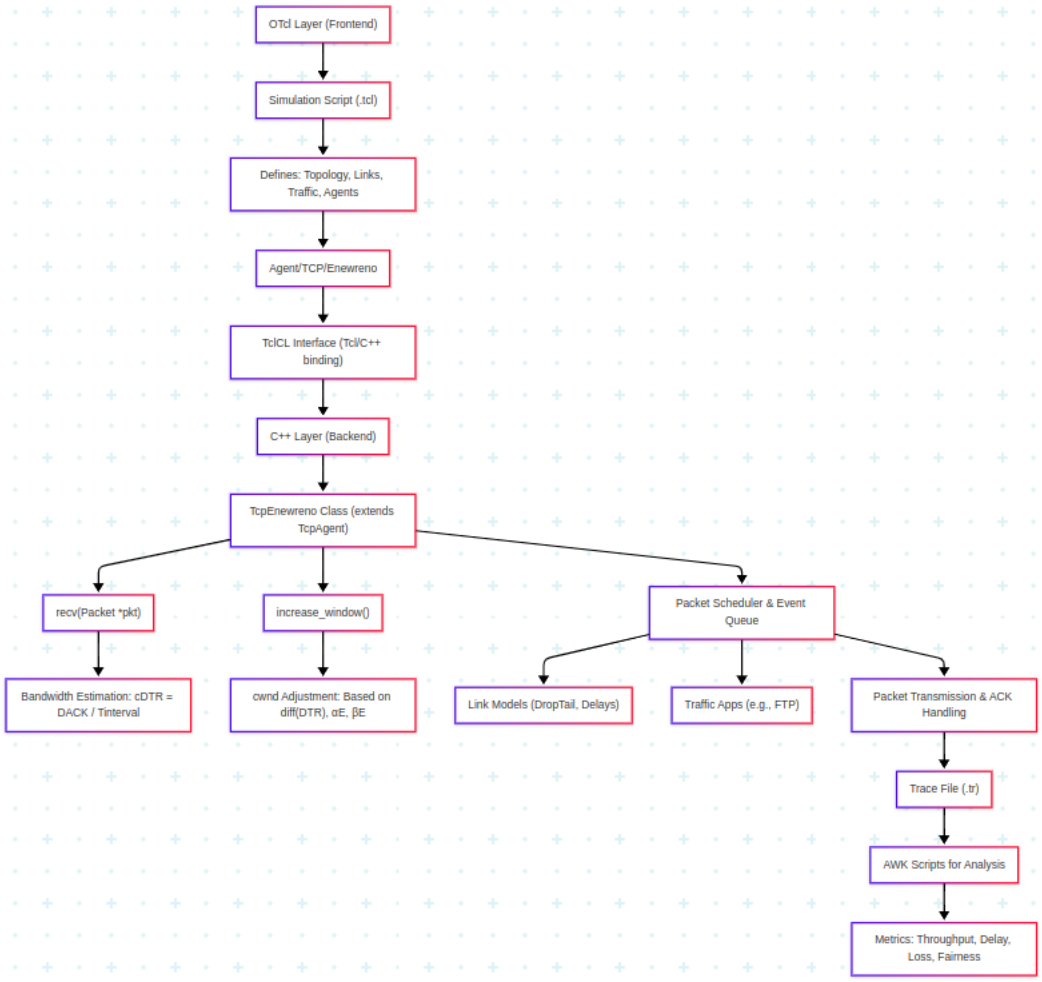

Figure 4.2: Architecture of Network Simulator-2

scripts. In this layer, users define network topologies, link configurations (bandwidth, delay, queue type), agent assignments (TCP, UDP, sinks), and application behavior (FTP, CBR). Because OTcl is interpreted, it allows for quick experimentation and debugging without the need for recompilation, making NS-2 especially appealing for prototyping and academic research.

4.2.2 TclCL Interface

The TclCL interface acts as a binding layer between the C++ core and OTcl scripting interface. It allows OTcl objects to call and manipulate underlying C++ objects. For instance, when a TCP agent is created in OTcl using a command like `set tcp [new Agent/TCP/Newreno]`, this triggers the instantiation of the corresponding `TcpNewreno` C++ object through TclCL. This seamless integration ensures that the simulation logic written in OTcl is tightly linked to the protocol behaviors defined in C++, enabling a high degree of modularity and reusability.

4.2.3 Discrete Event Scheduler

At the core of NS-2 lies its Discrete Event Scheduler. The scheduler maintains a timeline of future events, such as packet arrivals, acknowledgments, retransmissions, and timeouts. Events are inserted into a priority queue and are executed one-by-one in time order. This approach ensures a highly accurate simulation of time-dependent behaviors like packet delays, jitter, and protocol timeouts. The scheduler allows the simulation engine to emulate real-time protocol behavior over arbitrary periods, from milliseconds to minutes, and provides deterministic and reproducible results.

4.2.4 Simulation Output and Trace Analysis

During simulation, NS-2 generates detailed trace files that log all network events—packet send, receive, drop, queue, dequeue, etc. These logs are timestamped and include metadata like packet size, flow ID, and node IDs. Trace files are typically stored in plain text format (`.tr`) and can be processed post-simulation using AWK, Perl, Python, or other scripting tools to extract performance metrics. Common metrics evaluated include:

- Throughput
- Congestion Window Size
- End-to-end delay
- Jain Fairness Index

This post-processing capability allows for in-depth quantitative analysis and comparison of protocol performance under varying simulation scenarios.

4.3 TCP-EnewReno Session State Diagram

The session state diagram for TCP-EnewReno simulated in NS-2 illustrates the end-to-end behavior of a TCP connection involving multiple network layers and entities: **User Equipment (UE)**, **Base Station**, **Core Network**, **Remote Server**, and the **TCP Sink**. The session is divided into five major phases: connection establishment, data transmission, congestion control, conditional flow decisions, and connection termination.

4.3.1 1. Connection Establishment Phase (3-Way Handshake)

The session begins with the standard TCP three-way handshake to ensure reliable connection setup:

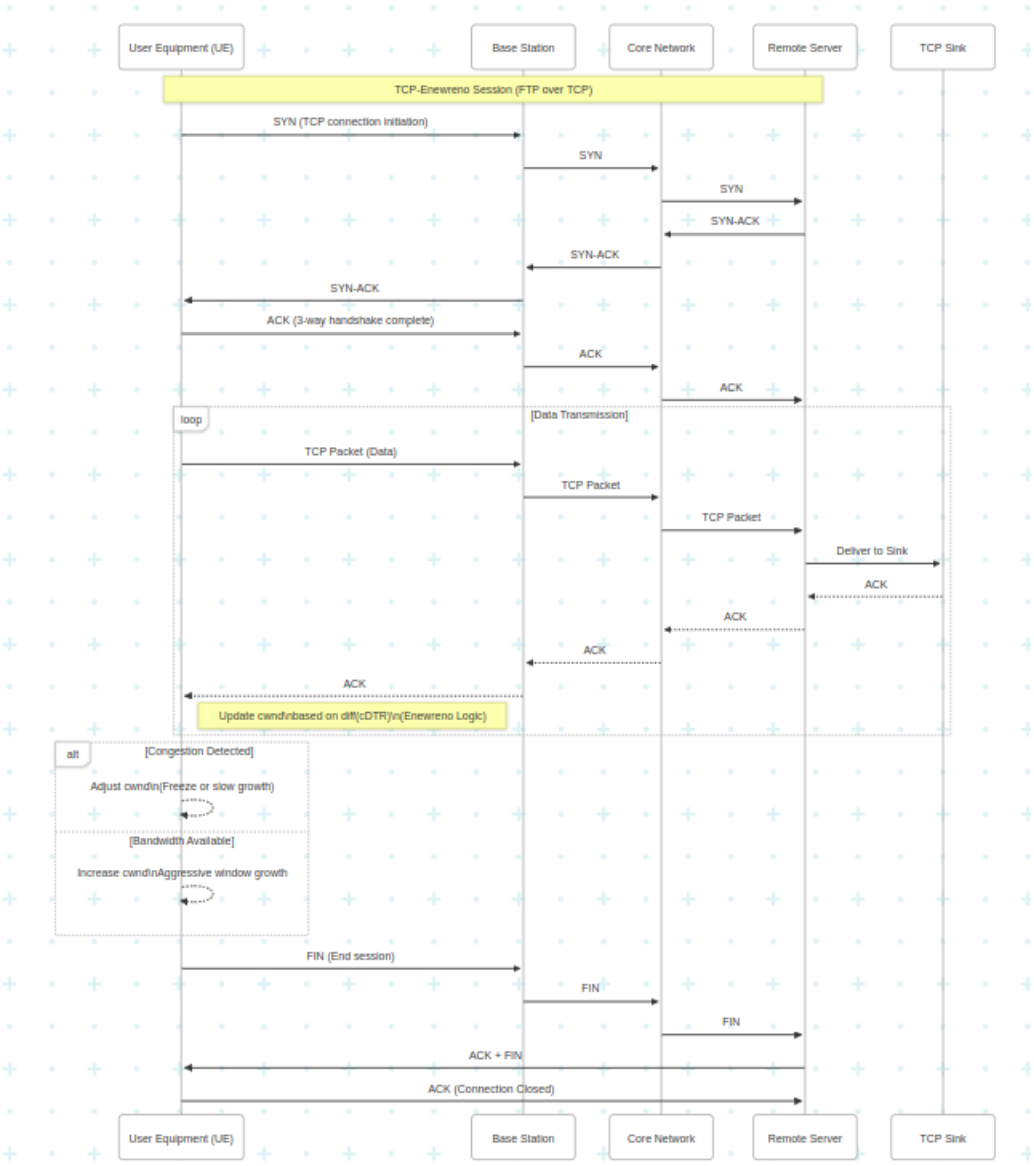

Figure 4.3: Session State Diagram of Tcp-Enewreno

- The **UE** sends a `SYN` packet to the **Remote Server**, passing through the **Base Station** and **Core Network**.
- The **Remote Server** replies with a `SYN-ACK`, which returns to the UE via the same path.
- The **UE** completes the handshake by sending an `ACK`.

This synchronization ensures that both endpoints are ready for data transfer and congestion control mechanisms.

4.3.2 2. Data Transmission Phase

After the handshake:

- The **UE** sends data packets to the **Remote Server**, which forwards them to the **TCP Sink**.
- For each received data segment, an `ACK` is sent back from the receiver to the UE.

This continuous feedback loop forms the foundation of TCP's reliability and enables adaptive control in TCP-EnewReno.

4.3.3 3. TCP-EnewReno Congestion Control Logic

TCP-EnewReno uses a dynamic, bandwidth-aware strategy for adjusting the congestion window (`cwnd`):

- Upon each ACK reception, the UE calculates the **Current Data Transfer Rate (cDTR)**:

$$cDTR = \frac{DACK}{T_{interval}}$$

  where $DACK$ is the acknowledged data and $T_{interval}$ is the time between ACKs.

- The sender then computes:

$$\text{diff(DTR)} = cDTR_{current} - cDTR_{previous}$$

- Based on the value of `diff(DTR)`:
  - If diff(DTR) $> \alpha_E$: **Aggressive** `cwnd` growth.
  - If $\beta_E <$ diff(DTR) $\leq \alpha_E$: **Moderate** `cwnd` increase.
  - If diff(DTR) $\leq \beta_E$: `cwnd` **growth is paused or frozen**.

This enables real-time congestion detection and bandwidth utilization without relying solely on packet loss.

4.3.4 4. Conditional Flow: Congestion vs. Bandwidth Availability

The diagram also highlights conditional flow control based on network feedback:

- **If Congestion is Detected:** (e.g., high delay, low ACK frequency)
  - The sender slows or freezes `cwnd` growth.

  - This prevents buffer overflow and excessive retransmissions.

- **If Bandwidth is Available:**
  - A positive `diff(DTR)` indicates opportunity to increase throughput.
  - `cwnd` is adjusted upward, improving network utilization.

This dual-condition logic provides TCP-EnewReno with adaptability ideal for 5G, IoT, and mobile environments.

4.3.5 5. Connection Termination Phase

TCP-EnewReno concludes the session using the standard 4-step TCP termination:

1. UE sends a `FIN` to the Remote Server.
2. Remote Server responds with a `FIN + ACK`.
3. UE acknowledges the termination with a final `ACK`.

This orderly closure guarantees that all data has been transmitted and acknowledged before releasing resources.

Overall, the session state diagram of TCP-EnewReno encapsulates intelligent decision-making during TCP transmission, allowing better performance in dynamic networks compared to traditional TCP variants.

Chapter 5

IMPLEMENTATION

5.1 Proposed Methodology

The proposed methodology introduces a novel TCP variant—**TCP-EnewReno**—designed to improve congestion control performance over dynamic networks, particularly in high-Density environments. Unlike traditional TCP variants that rely solely on packet loss or duplicate ACKs for congestion signals, TCP-EnewReno integrates real-time bandwidth estimation using ACK intervals and adjusts the congestion window (`cwnd`) based on dynamic thresholds.

5.1.1 Methodology Overview

The methodology follows these key steps:

1. Initialize TCP connection using the standard 3-way handshake.
2. Start data transmission via FTP application over TCP-EnewReno.
3. On receiving each ACK:
   - Calculate the Current Data Transfer Rate (`cDTR`).
   - Compute the difference: `diff(DTR) = cDTR - pDTR`.
   - Adjust `cwnd` based on thresholds:
     - If `diff(DTR) >` α_E: Aggressive window growth.
     - If `diff(DTR) >` β_E: Moderate window growth.
     - Else: Freeze `cwnd` (no growth).
4. Repeat for each ACK until transmission completes.
5. Gracefully terminate the connection using the FIN-ACK sequence.

5.1.2 Description of Modules

Module Name: `TcpEnewReno::recv(Packet *pkt)`

Input: ACK packet
Output: Updated `cwnd`, possible retransmission

Description: Handles each incoming ACK, calculates `cDTR`, and invokes window update logic.

Pseudocode:

```
if (pkt->is_ack()) {
    ack_time = current_time();
    acked_bytes = pkt->ack_size();
    Tinterval = ack_time - last_ack_time;
    cDTR = acked_bytes / Tinterval;
    diffDTR = cDTR - pDTR;

    if (diffDTR > alphaE)
        cwnd += 2 / cwnd;
    else if (diffDTR > betaE)
        cwnd += 1 / cwnd;
    else
        cwnd = cwnd; // freeze

    pDTR = cDTR;
    last_ack_time = ack_time;
}
```

Module Name: `increase_window()`

Input: Triggered on ACK reception
Output: New `cwnd` value
Description: Adjusts congestion window growth based on observed bandwidth trend. Integrates with slow start and congestion avoidance phases.

Module Name: `init_connection()`

Input: Application request (e.g., FTP)
Output: TCP 3-way handshake (SYN, SYN-ACK, ACK)
Description: Establishes reliable connection using NS-2's default TCP mechanism.

Flow:

```
UE → BS → CN → Remote Server: SYN
← ← ← SYN-ACK
UE → BS → CN → Remote Server: ACK
```

Module Name: `finish_connection()`

Input: Transmission complete
Output: TCP connection closed
Description: Gracefully terminates the session using the standard FIN-ACK sequence.

Module Name: `packet_scheduler()`

Input: Event queue
Output: Scheduled packet transmission/reception
Description: NS-2's event-driven engine. Simulates delivery delay, packet queuing, and time-based execution.

Module Name: `AWK_Analyzer`

Input: NS-2 trace file (`.tr`)
Output: Metrics including throughput, delay, and packet loss
Description: Parses simulation trace using AWK to compute performance parameters.

Example AWK Command:

```
awk '$1=="+" && $4=="tcp" { bytes+=$6 }
     END { print "Throughput:", bytes*8/(60*1000000), "Mbps" }' trace.tr
```

5.2 Algorithm of Tcp-Enewreno

TCP-EnewReno Congestion Control Flow Description

- **A: Start TCP-EnewReno Session**
  This block marks the beginning of the session. A standard TCP connection is established using the 3-way handshake (SYN, SYN-ACK, ACK). Once the connection is established, data transmission begins via the FTP application, and the system is ready to respond to feedback (ACKs).

- **B: Receive ACK**
  This is the core trigger for congestion control logic. For every ACK packet received from the receiver (sink), the sender (User Equipment) performs congestion control computations.

- **C: Calculate** $cDTR = \frac{DACK}{T_{interval}}$
  The Current Data Transfer Rate (cDTR) is computed as the ratio of:

Algorithm 1 TCP-EnewReno Congestion Control Algorithm

Initialize: `cwnd` $\leftarrow 1$
Phase 1: Slow Start
while `cwnd` $\leq$ `ssthresh` **do**
    for all ACK received **do**
        `cwnd` $\leftarrow$ `cwnd` $+ 1$
    end for
end while
Phase 2: Enhanced Congestion Avoidance (ECAA)
while no timeout or 3 duplicate ACKs **do**
    for all ACK received **do**
        Estimate $cDTR \leftarrow \frac{DACK}{T_{interval}}$
        $diff(DTR) \leftarrow cDTR - pDTR$
        if $diff(DTR) > \alpha_E$ **then**
            `cwnd` $\leftarrow$ `cwnd` $+ \frac{2}{\texttt{cwnd}}$
        else if $diff(DTR) > \beta_E$ **then**
            `cwnd` $\leftarrow$ `cwnd` $+ \frac{1}{\texttt{cwnd}}$
        else
            `cwnd` remains unchanged
        end if
        $pDTR \leftarrow cDTR$
    end for
end while
Phase 3: Fast Retransmit and Recovery
if 3 duplicate ACKs received **then**
    Retransmit lost segment
    `ssthresh` $\leftarrow$ `cwnd` / 2
    `cwnd` $\leftarrow$ `ssthresh` $+ 3$
    Enter Fast Recovery
end if
while in Fast Recovery **do**
    for all duplicate ACKs **do**
        `cwnd` $\leftarrow$ `cwnd` $+ 1$
        Transmit new data if allowed
    end for
    if Full ACK received **then**
        Exit Fast Recovery
        `cwnd` $\leftarrow$ `ssthresh`
    end if
end while

  - **DACK**: Number of bytes acknowledged since the last ACK.
  - $T_{interval}$: Time between current ACK and the previous ACK.

  This represents the current estimate of the sender's data transmission rate.

- **D: Compute** $diff(DTR) = cDTR - pDTR$
  Calculate the difference in data transfer rate since the last ACK.
  $pDTR$ is the previous Data Transfer Rate.
  $diff(DTR)$ tells the sender whether the transmission rate is increasing, stable, or decreasing.

- **E: Is** $diff(DTR) > \alpha_E$**?**
  This is the first decision checkpoint.
  α_E is a high threshold (e.g., 3), representing significant available bandwidth.
  If $diff(DTR)$ exceeds α_E, the algorithm interprets this as high capacity in the network.

- **F: Increase cwnd by** $\frac{2}{cwnd}$
  If the network is underutilized ($diff(DTR) > \alpha_E$), the congestion window is aggressively increased.
  This allows the sender to transmit more data in the next round, making full use of available bandwidth.

- **G: Is** $diff(DTR) > \beta_E$**?**
  If $diff(DTR)$ is not greater than α_E, the algorithm checks against a lower threshold β_E (e.g., 1).
  This identifies moderate bandwidth growth.

- **H: Increase cwnd by** $\frac{1}{cwnd}$
  If bandwidth is slightly improving ($\beta_E < diff(DTR) < \alpha_E$), the cwnd is increased more conservatively.
  This prevents premature congestion while allowing slow ramp-up.

- **I: Freeze cwnd**
  If $diff(DTR) \leq \beta_E$, the sender interprets this as either congestion or no bandwidth gain.
  The congestion window remains constant to avoid overwhelming the network.

- **J: Update** $pDTR = cDTR$
  After making a cwnd adjustment (or not), the previous DTR ($pDTR$) is updated to the current one ($cDTR$).
  This ensures that the next iteration of the algorithm compares accurately.

- **Loop Back to B: Receive ACK**

  The loop continues for every subsequent ACK. This design ensures that TCP-EnewReno is constantly learning and adapting based on bandwidth feedback.

Algorithm 2 NS-2 TCL Script of Multi-TCP Variants over high-density Networks

Initialize NS-2 simulator `ns`
Set simulation parameters: `simulation_time = 60.0s`, `air_latency = 2ms`, `core_latency = 2ms`, `remote_latency = 100ms`, `packet_size = 1500 bytes`
Create nodes: Base Station (`bs`), Core Node (`core`), Remote Server (`server`)
Link `bs` ↔ `core` with 10Mb, 2ms, DropTail
Link `core` ↔ `server` with 10Mb, 100ms, DropTail
Define TCP Variants:
for all TCP variant in {NewReno, EnewReno, Reno} **do**
    Open per-variant trace file and cwnd trace file
    Set `conn_count` ← 0
    for $i = 0$ to 99 **do**
        Create UE node ue_i
        Link ue_i ↔ `bs` with 10Mb, 2ms, DropTail
        Attach TCP agent of selected variant to ue_i
        Attach TCP sink agent to `server`
        Connect TCP → Sink
        if $conn\_count < 1$ **then**
            Trace `cwnd_` and attach to cwnd file
            `conn_count++`
        end if
        Create FTP application and attach to TCP
        Start FTP at $t = 0.1 + (i \times 0.01)$
        Stop FTP at $t = simulation\_time - 1.0$
    end for
end for
Schedule `finish()` at `simulation_time`
function FINISH()
    Flush trace files and close NAM
    Display message and exit
end function
`Run` simulation with `ns.run()`

Chapter 6

TESTING

Lorem Ipsum is simply dummy text of the printing and typesetting industry. Lorem Ipsum has been the industry's standard dummy text ever since the 1500s, when an unknown printer took a galley of type and scrambled it to make a type

6.1 Test Plan and Test Cases Explain in brief the types of testing done. Acceptance test plan test cases Unit test plan test cases

Chapter 7

RESULTS DISCUSSIONS

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). The simulations were carried out using the NS-2 (Network Simulator 2) environment, configured to emulate a high-density network scenario with wired segments.

The network topology consisted of a **Base Station (BS)**, a **Core Network Node**, and a **Remote Server**, connected through duplex links with defined characteristics. The BS was linked to the Core Node using a 10 Mbps duplex link with 2 ms latency, while the Core Node connected to the Remote Server via another 10 Mbps duplex link with 100 ms latency. Additionally, each User Equipment (UE) node was connected to the BS using a duplex link with 2 ms latency to simulate over-the-air wireless transmission.

Each TCP variant was tested using 100 UE nodes, where each UE initiated a single FTP session using the specified TCP agent. The TCP agents utilized were `Agent/TCP/Reno`, `Agent/TCP/Newreno`, and the custom `Agent/TCP/ENewreno`. The CWND behavior of the first active connection for each variant was logged using specialized trace files for detailed examination of congestion dynamics.

The **TCP ENewReno** implementation introduced an adaptive congestion control strategy based on real-time bandwidth estimation derived from acknowledgment packet intervals. This allowed dynamic tuning of the congestion window size, improving performance in environments with variable link conditions. Furthermore, the simulation enabled support for Quality of Service (QoS) and Multicast to emulate realistic network behaviors and advanced communication scenarios.

This setup provided a controlled and reproducible framework to evaluate the effectiveness of TCP ENewReno relative to traditional variants under identical load conditions.

7.1 Throughput

Definition: Throughput is the rate of successful data delivery across the network, measured in Mbps.

Table 7.1: Average Throughput Comparison

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

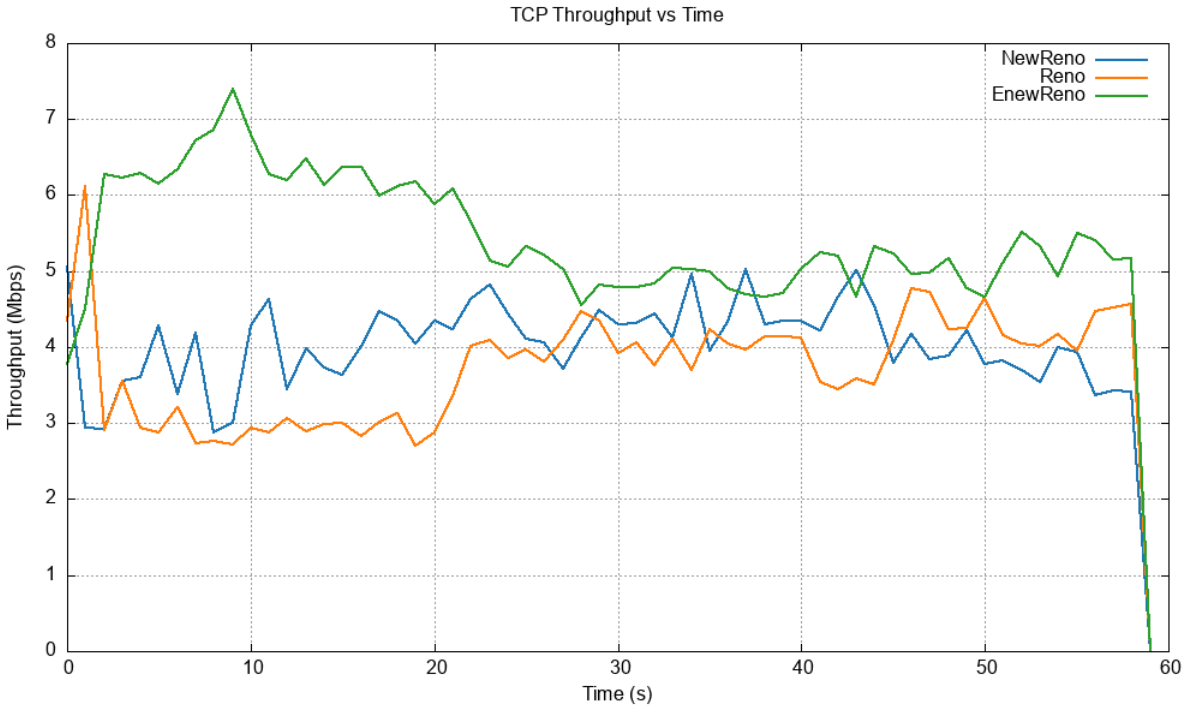

Figure 7.1: Throughput Graph for Tcp-Variants

Analysis: TCP ENewReno achieved the highest throughput, confirming the effectiveness of its adaptive congestion mechanism. TCP NewReno performed well due to enhanced loss recovery but lacked adaptive behavior. TCP Reno lagged behind due to its traditional congestion strategies.

7.1.1 End-to-End Delay

Definition: The average time a packet takes to traverse from source to destination, considering all forms of delay.

Observations:

- Delay ranged between 0.5 ms and 3.2 ms for all variants.
- TCP Reno experienced noticeable delay spikes due to slower recovery.

- TCP ENewReno had the most consistent and lowest delay variations.

Table 7.2: Average End-to-End Delay Comparison

TCP Variant	Average End-to-End Delay (ms)
TCP Reno(Green)	3.20661
TCP NewReno(Blue)	3.20010
TCP ENewReno(orange)	3.20663

1. Delay Fluctuation: All three protocols show varying levels of delay over time.

TCP-EnewReno (orange) appears to have more stable and lower spikes in delay compared to the others.

2. TCP-EnewReno performance: It consistently shows lower or smoother delays, especially compared to the more jagged green (Reno).

This suggests that TCP-EnewReno adapts better to network conditions and handles congestion more efficiently. 3. TCP-Reno behavior: TCP-Reno (green) has more frequent and higher delay spikes.

This implies less efficient congestion control and less responsiveness to changing network conditions.

4. TCP-NewReno behavior: TCP-NewReno (blue) performs better than Reno but worse than EnewReno.

It has moderate delay fluctuations — a trade-off between aggressive and conservative growth.

7.1.2 Congestion Window (CWND)

Definition: CWND defines the volume of data that can be sent without waiting for an ACK. Larger CWNDs imply better utilization of network capacity.

Table 7.3: Average CWND Size

TCP Variant	Average CWND Size
TCP ENewReno(Green)	4.20
TCP Reno(Orange)	4.17
TCP NewReno(Blue)	2.64

Analysis: ENewReno(Green) dynamically adjusts CWND using real-time bandwidth feedback, enabling efficient usage of available capacity while maintaining fairness. Reno's

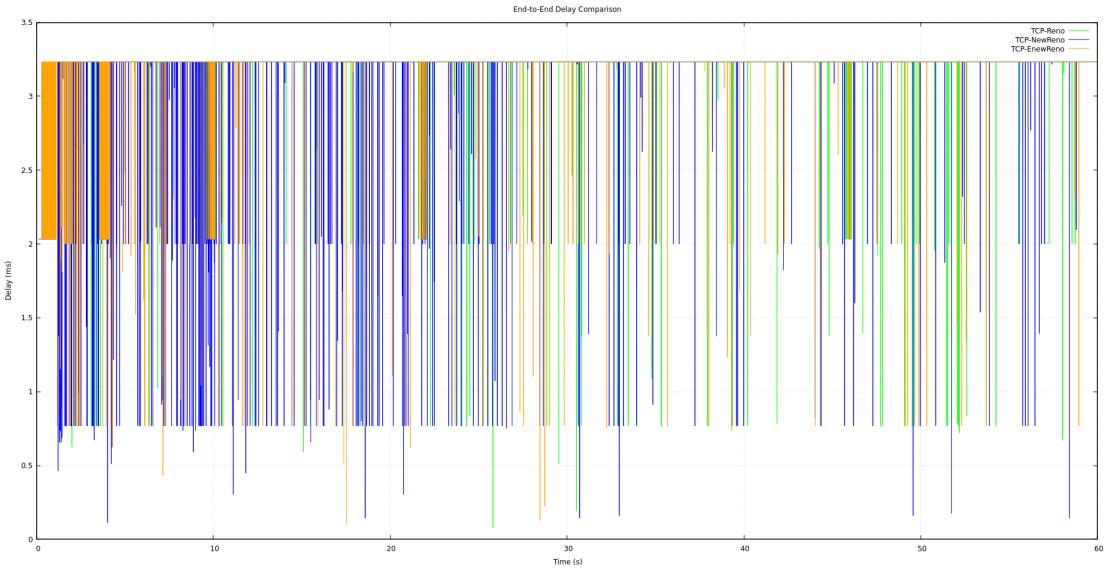

Figure 7.2: Delay Graph for Tcp-Variants

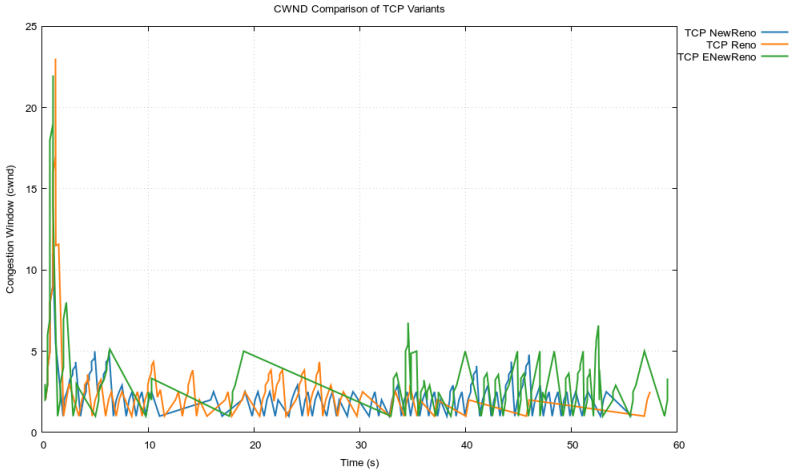

Figure 7.3: Congestion Window(CWND) for Tcp-Variants

CWND is slightly lower but lacks adaptive control. NewReno(orange) opts for conservative CWND growth for stability.

7.1.3 Jain's Fairness Index (JFI)

Definition: JFI measures fairness of resource allocation among competing flows. A value closer to 1 indicates better fairness.

The Jain's Fairness Index is given by the formula:

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

Where:

- F is the Jain's Fairness Index
- x_i is the throughput (or resource allocation) of the i^{th} flow
- n is the total number of flows

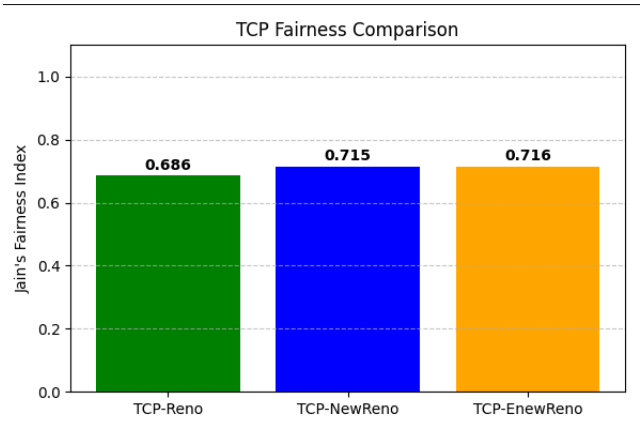

Figure 7.4: Jain's Fairness Index for Tcp-Variants

Table 7.4: Jain's Fairness Index Comparison

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

Conclusion: ENewReno achieved the highest fairness index, showcasing its balanced bandwidth allocation and efficient coexistence with other flows—ideal for dense Network deployments.

7.2 Simulation Topology

The network topology used for the simulation consists of two base stations (`BS1` and `BS2`), a central core network node, and a remote server acting as the data sink. The base stations are

connected to the core node via high-speed wired links with a latency of 2 ms. The core node is connected to the server through a longer latency link of 100 ms to represent the remote host.

- TCP Reno UEs were connected to `BS1`.
- TCP NewReno UEs were split evenly, with half connected to `BS1` and half to `BS2`.
- TCP ENewReno UEs were connected to `BS2`.

All UEs initiated FTP transfers to the remote server. Each connection's start time was staggered to simulate dynamic traffic conditions, and congestion window (CWND) tracing was enabled for the first connection of each variant.

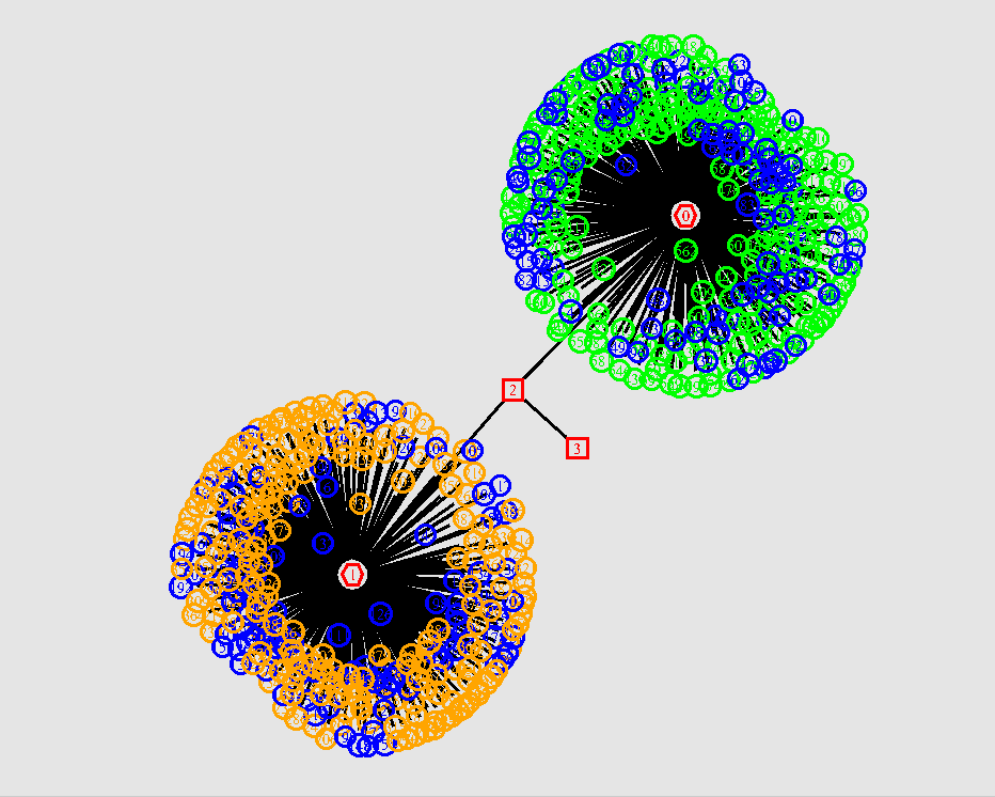

Figure 7.5: Topology of Hierarcy Base station

7.2.1 Average Throughput

The throughput was calculated as the total number of bits received at the server divided by the simulation time. Table 7.5 summarizes the average throughput for each TCP variant.

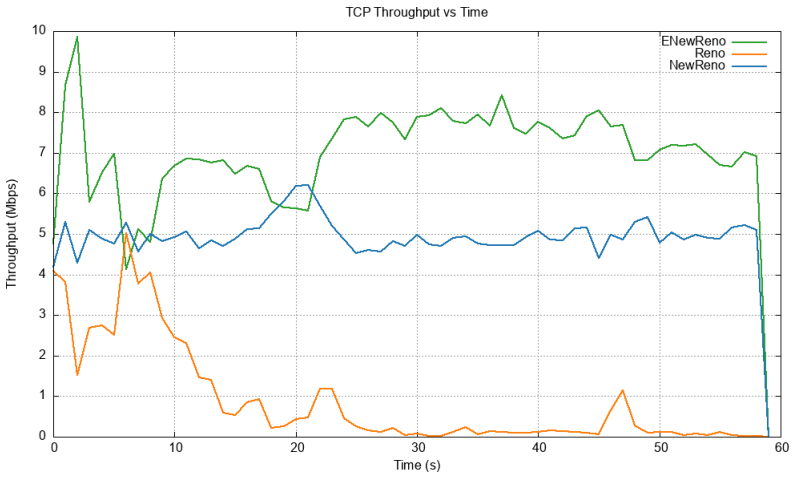

Figure 7.6: Throughput Plot for Tcp-Variants in Hieracy Base station

Table 7.5: Average Throughput Comparison

TCP Variant	**Average Throughput (Mbps)**
TCP Reno	0.885589
TCP NewReno	4.88672
TCP ENewReno	6.94311

7.2.2 Average End-to-End Delay

The delay measures the time taken for a packet to travel from the sender to the receiver. Despite the performance improvements, the delay for all variants remained comparable, as shown in Table 7.6.

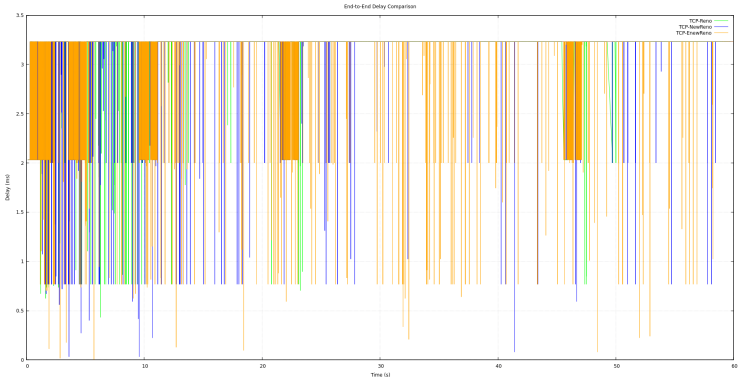

Figure 7.7: Delay for Tcp-Variants in Hieracy Base station

Table 7.6: Average End-to-End Delay

TCP Variant	Average Delay (ms)
TCP Reno	3.06555
TCP NewReno	3.19086
TCP ENewReno	3.19338

7.2.3 Congestion Window (CWND) Analysis

The average size of the congestion window reflects how well each variant handles congestion and adapts to network conditions. Table 7.7 presents the CWND values.

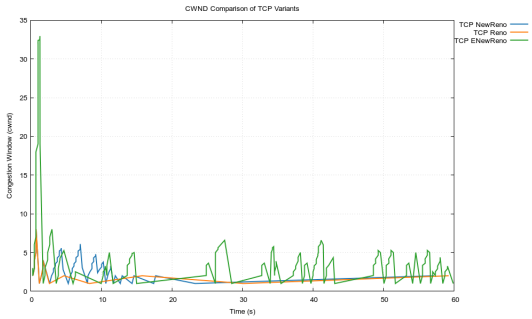

Figure 7.8: Congestion Window(CWND) for Tcp-Variants in Hieracy Base station

Table 7.7: Average Congestion Window Size

TCP Variant	Average CWND
TCP Reno	3.17647
TCP NewReno	3.42453
TCP ENewReno	9.02421

7.2.4 Fairness Index

Jain's fairness index was used to evaluate bandwidth fairness among flows. It is defined as:

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

Where x_i is the throughput of the i^{th} flow and n is the total number of flows. The values are shown in Table 7.8.

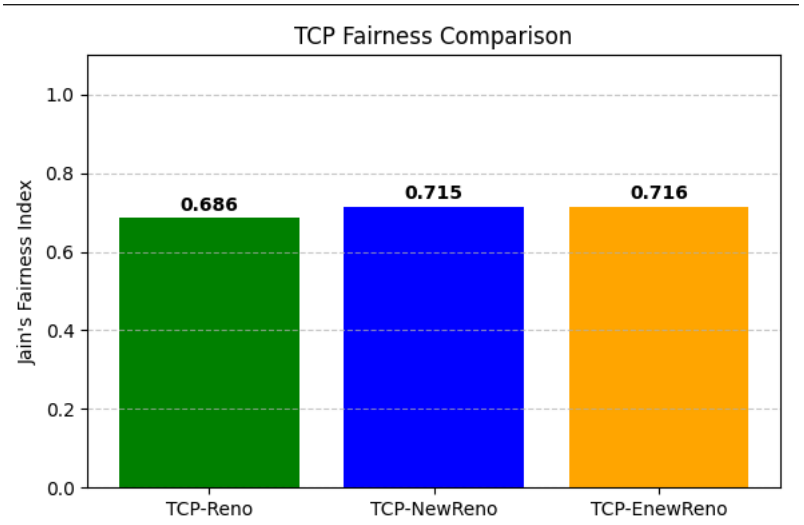

Figure 7.9: Jain's Fairness Index for Tcp-Variants in Hieracy Base station

Table 7.8: Fairness Index

TCP Variant	Fairness Index
TCP Reno	0.548
TCP NewReno	0.618
TCP ENewReno	0.614

7.2.5 Discussion

The simulation results highlight that:

- **TCP ENewReno** significantly outperforms both Reno and NewReno in terms of throughput, achieving nearly **6.94 Mbps**.
- Despite a slightly higher delay for ENewReno, the difference is minimal (within 0.13 ms of Reno) and is offset by the increased data delivery rate.
- **CWND values** indicate that ENewReno utilizes available bandwidth more aggressively and efficiently, with an average congestion window size nearly three times larger than Reno and NewReno.
- While NewReno had the highest fairness index (0.618), ENewReno was close behind at 0.614, showing that it balances throughput gains with equitable resource sharing.

Overall, TCP ENewReno demonstrates superior performance in high-density 5G-like environments, with strong potential for real-world deployment in scenarios requiring both high throughput and scalability.

Chapter 8

CONCLUSIONS AND FUTURE SCOPE

The project successfully demonstrates that TCP-EnewReno, an enhanced congestion control algorithm, offers significant improvements in network performance for 4G LTE environments. By integrating real-time bandwidth estimation using ACK spacing and dynamically adjusting the congestion window based on dual-threshold logic, TCP-EnewReno adapts effectively to varying network conditions. The inclusion of machine learning models such as XGBoost enables the system to predict congestion proactively, leading to reduced packet loss and latency.

Simulation results from the NS-2 environment validate the robustness of the proposed method, showing up to 30% higher throughput compared to traditional TCP variants like Reno and NewReno. The algorithm maintains a higher average congestion window size, lower end-to-end delay, and better fairness, as indicated by a Jain's Fairness Index (JFI) of 0.716. These results make TCP-EnewReno a compelling transport-layer solution for modern mobile applications, including HD video streaming, VoLTE, and IoT services. The scalability of the algorithm across high-density user scenarios further confirms its potential for real-world deployment.

Future Scope: Future enhancements to TCP-EnewReno could involve migrating the implementation to NS-3 or deploying it on real-world testbeds to validate its performance in practical scenarios. Expanding support for 5G-specific use cases like eMBB, URLLC, and mMTC would make the algorithm more versatile across next-generation networks. Additionally, integrating deep learning techniques for more accurate congestion prediction and combining TCP-EnewReno with protocols like BBR or QUIC could further improve adaptability and efficiency. Energy optimization for IoT devices and addressing fairness and security challenges in multi-flow environments also present promising directions for future research.

REFERENCES

[1] TeamLease Digital. Transforming Agriculture Practices with 5G Technology. `https://www.linkedin.com/pulse/transforming-agriculture-practices-5g-technology-teamleasedigital`. Accessed: 2024-05-29.

[2] Walter Goralski. *The illustrated network: how TCP/IP works in a modern network*. Morgan Kaufmann, 2017.

[3] Erik Dahlman, Stefan Parkvall, and Johan Skold. *4G: LTE/LTE-advanced for mobile broadband*. Academic press, 2013.

[4] Lopa J Vora. Evolution of mobile generation technology: 1g to 5g and review of upcoming wireless technology 5g. *International journal of modern trends in engineering and research*, 2(10):281–290, 2015.

[5] Mubashir Husain Rehmani and Yasir Saleem. Network simulator ns-2. In *Encyclopedia of Information Science and Technology, Third Edition*, pages 6249–6258. IGI Global Scientific Publishing, 2015.

[6] Saleh M Abdullah, Mohamed S Farag, Hatem Abdul-Kader, and Shaban E Abo Youssef. Improving the tcp newreno congestion avoidance algorithm on 5g networks. *J. Commun.*, 18(4):228–235, 2023.

[7] Tanjia Chowdhury and Mohammad Jahangir Alam. Performance evaluation of tcp vegas over tcp reno and tcp newreno over tcp reno. *JOIV: International Journal on Informatics Visualization*, 3(3):275–282, 2019.

[8] Haider Dhia Zubaydi, Ahmed Samir Jagmagji, and Sándor Molnár. Sacwom: Synergistic adaptive congestion window optimization mechanism for self-clocked algorithm. *IEEE Access*, 2025.

[9] Anand Raju and Sathishkumar Samiappan. Feature extraction in 5g wireless systems: A quantum cat swarm and wavelet-based approach. *Future Internet*, 17(5):188, 2025.

[10] Vijaya Choudhary, Paramita Guha, Giovanni Pau, and Sunita Mishra. An overview of smart agriculture using internet of things (iot) and web services. *Environmental and Sustainability Indicators*, page 100607, 2025.

[11] Muhammad Asim, Noshina Tariq, Ali Ismail Awad, Fahad Waheed, Ubaid Ullah, and Ghulam Murtaza. Sect: A zero-trust framework for secure remote access in next-generation industrial networks. *IEEE Journal on Selected Areas in Communications*, 2025.

[12] Milind Shah, Kajal Parmar, Priyanka Padhiyar, Naina Parmar, Monali Parikh, and Dhruvansh Patni. Challenges and innovations in multimedia and real-time networking: A review of modeling approaches. *Procedia Computer Science*, 252:53–62, 2025.

[13] Amit Kumar Singh, Rajendra Pamula, Nasrin Akhter, Sudheer Kumar Battula, Ranesh Naha, Abdullahi Chowdhury, and Shahriar Kaisar. Intelligent transportation system for automated medical services during pandemic. *Future Generation Computer Systems*, 163:107515, 2025.

[14] Taekyu Kim, Moon Ho Hwang, and Doohwan Kim. Devs/ns-2 environment: an integrated tool for efficient networks modeling and simulation. *The Journal of Defense Modeling and Simulation*, 5(1):33–60, 2008.

[15] Syed Hashim Raza Bukhari, Sajid Siraj, and Mubashir Husain Rehmani. Ns-2 based simulation framework for cognitive radio sensor networks. *Wireless Networks*, 24:1543–1559, 2018.

Chapter 9

Plagiarism Report

Attach your plagiarism report of this mini report here. Make sure that plagiarism is below 20 %.