VAASAN AMMATTIKORKEAKOULU
UNIVERSITY OF APPLIED SCIENCES

Vu Duc Trung

5G LINK PERFORMANCE OF DIFFERENT TCP CONGESTION CONTROL AVOIDANCE SCHEMES

Technology and Communication
2023

ACKNOWLEDGEMENTS

I would like to express my sincere gratitude to my supervisor, Gao Chao, for his valuable guidance and support throughout this project. I would also like to extend my appreciation to the entire team at the 5G lab in Technobothnia for their assistance and cooperation in providing the necessary resources and facilities for the successful completion of this project. Thank you all for your contributions.

VAASAN AMMATTIKORKEAKOULU
UNIVERSITY OF APPLIED SCIENCES
Information Technology

ABSTRACT

Author	Vu Duc Trung
Title	5G Link Performance of Different TCP Congestion Control Avoidance Schemes
Year	2023
Language	English
Pages	53
Name of Supervisor	Gao Chao

The current 5G mobile networks are already utilizing all-IP networks, employing the TCP/IP protocol stack to transmit application data for each 5G user. Consequently, investigating the efficiency of TCP over 5G radio connections becomes a significant and intriguing subject. Additionally, TCP has congestion control mechanisms to manage variations in network links, and numerous algorithms have been created to enhance communication performance. Thus, the goal of this thesis is to evaluate and compare the performance of various TCP congestion control algorithms.

To achieve this goal, the thesis will analyse the data collected from testing using software tool. The testing will be conducted in two primary phases, starting with the installation and configuration of the necessary tools and software. The second phase will involve the actual testing and data collection, which will provide insight into the effectiveness of different congestion control algorithms.

Through analysis and comparison of different TCP congestion control avoidance schemes, this thesis has identified a superior algorithm that outperforms others. This discovery holds significant value in the ongoing advancement and exploration of 5G networks. The insights gained from this study can assist network administrators and researchers in making well-informed choices regarding the selection of the most suitable TCP congestion control algorithm for their specific requirements.

Keywords 5G cellular networks, TCP congestion control algorithms, throughput, and link performance

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

ACK	Acknowledgement
AIMD	Additive Increase Multiplicative Decrease
AR	Augmented Reality
BBR	Bottleneck Bandwidth and Round-trip propagation time
CCA	Congestion Control Algorithm
cwnd	Congestion window value
ECN	Explicit Congestion Notification
HTCP	Hamiltonian TCP
IW	initial window
MIMO	Multiple input Mutilple output
NR	New Radio
NSA	Non-Standalone
OFDM	Orthogonal Frequency Division Multiple
OS	Operating System
QAM	Quadrature Amplitude Modulation
RRM	Radio Resource Management
RTO	retransmission timeout
RTT	Round-trip Time
SA	Standalone
SMSS	Sender Maximum Segment Size
ssthresh	Slow Start threshold
TCP	Transmission Control Protocol
TDD	Time Division Duplex
UDP	User Datagram Protocol
VR	Virtual Reality

1 INTRODUCTION

We commence by providing a brief overview of the various generations of cellular networks, encompassing comprehensive information on wireless networks. This chapter encompasses the essential characteristics of each cellular network in question. Additionally, this section incorporates the rationale behind undertaking this thesis and outlines the anticipated results of the project. Finally, a thorough description of the structure of the thesis will be presented.

1.1 5G Cellular Networks

The latest mobile network technology, Fifth Generation (5G), offers faster data transmission with the help of several technologies. One of the key technologies in 5G is millimetre wave (mm Wave) spectrum, which provides faster data transmission than previous generations. However, this spectrum has a shorter range and can be easily blocked by obstacles such as buildings and trees. Therefore, 5G coverage is limited to short distances, but it can achieve communication at a throughput up to 10Gbps. Massive MIMO is another key technology in 5G, which uses many antennas to increase network capacity and improve data transfer speeds. Beamforming technology is also successfully employed to enhance signal strength, improving the quality and reliability of wireless connections. 5G also utilizes advanced modulation techniques, such as Orthogonal Frequency-Division Multiplexing (OFDM) and Quadrature Amplitude Modulation (QAM) to increase data rates and spectral efficiency. These technologies allow simultaneous transmission of multiple data streams over the same frequency band, which enhances data capacity and throughput. (Sadeeq, Aljahmany, Zebari, & Rizgar, 2020) (Rappaport, 2013) (Sun, 2020).

1.2 Motivation of Approaching This Project

Currently, the global wireless of 5G mobile telecommunication is on the rise due to its high-speed transmission capabilities, offering throughput of up to 10Gbps. This technology is specifically designed to cater to bandwidth-intensive applications such as Virtual Reality (VR) and Augmented Reality (AR), Internet of Things

(IoT), Big Data, and more. However, in the existing networks, high data transmission often faces challenges such as packet loss and burst packets, primarily caused by network congestion at intermediate points. To address this issue, Transmission Control Protocol (TCP) incorporates Congestion Control Algorithms (CCA) as a crucial feature to ensure network stability during times of increased network load. The effective implementation of a suitable congestion control algorithm can significantly enhance the performance of TCP in 5G communications. The choice of an appropriate TCP congestion control scheme plays a crucial role in determining the throughput of a network. Furthermore, selecting the suitable congestion control scheme for a particular network setting can enhance TCP performance by increasing throughput, decreasing latency, and minimizing packet loss. In contrast, utilizing an unsuitable CCA can lead to suboptimal performance and inefficient use of network resources. Hence, choosing the right CCA can have a considerable impact on the efficiency and effectiveness of TCP communication within a specific network context. Regrettably, the research on the performance of various TCP congestion avoidance schemes in 5G networks is limited. As a result, this thesis has tested the performance of 5 TCP CCAs in Technobothnia laboratory in Vaasa, Finland.

1.3 Objective and Outcomes

The primary objective of the thesis is to evaluate and compare the performance of different congestion control schemes in 5G New Radio (NR) networks. The thesis aims to conduct comprehensive experiments using realistic scenarios and measurements to assess the effectiveness of various congestion control algorithms in 5G networks. The research focuses on studying how different congestion control algorithms perform in terms of throughput, latency, packet loss, and more.

The research outcomes are anticipated to yield valuable insights into the performance of congestion control algorithms in 5G networks, particularly in managing congestion and optimizing performance in 5G NR networks across different scenarios. The findings may also shed light on the trade-offs and limitations of various congestion control schemes and provide practical recommendations for their application in specific use cases, such as VR, AR, IoT, and other applications in 5G networks.

1.4 Thesis Structure

The thesis follows the following structure. Chapter 1 provides a concise overview of 5G wireless cellular networks, highlighting their key features. Chapter 2 delves into the necessary theoretical background for this project. This chapter includes definitions of User Datagram Protocol (UDP), TCP, and CCAs, along with an explanation of Linux commands for TCP congestion control avoidance schemes. Chapter 3 focuses on the testing implementation, discussing the chosen testing approach, equipment utilized, and installation instructions. Chapter 4 is dedicated to presenting and discussing the results obtained. Finally, the last chapter, the conclusion, provides a brief summary of the project.

2 BACKGROUND KNOWLEDGE

In this section, we will examine the explanations of UDP and TCP in the field of telecommunications. Following that, we will discuss five different CCAs that will be utilized in this thesis for testing objectives. Prior to delving into the testing process, it would be beneficial to review the definitions to gain a thorough understanding. Additionally, this chapter includes information on the Linux command used to adjust the CCA in the operating system and instructions on its utilization.

2.1 User Datagram Protocol (UDP)

The User Datagram Protocol (UDP) is a network protocol that operates in a connectionless protocol. UDP is commonly used in scenarios where real-time communication and low latency are crucial, such as multimedia streaming, online gaming, and voice over IP (VoIP) applications. Its connectionless nature allows for faster transmission speeds, as there is no need to establish and maintain a continuous connection between sender and receiver (Stevens, 1994). When testing network performance, the tool "iperf" is commonly used with UDP to measure packet loss ratio, jitter, and round-trip delay. These metrics provide insights into network quality and reliability. In this article, we will explore the capabilities of UDP and how iperf can be utilized to assess network performance. Understanding UDP and iperf helps optimize networks for enhanced efficiency and user experience.

2.2 Transmission Control Protocol (TCP)

The Transmission Control Protocol (TCP) is a widely used network protocol that operates in a connection-oriented protocol that provides a reliable, ordered, and error-checked connection between two network endpoints. TCP is designed to facilitate communication between applications running on different hosts, and it offers several key characteristics that make it an ideal protocol for a wide range of network communication scenarios. Firstly, TCP is connection-oriented, which means that a reliable connection must be established between the two endpoints before any data can be transmitted. This connection is established using a three-way handshake process, where the sending host initiates the connection, the receiving host acknowledges the connection, and the sending host confirms the acknowledgment. Once the connection is established, TCP provides a virtual circuit between the two endpoints that enables reliable and ordered data transfer. Secondly, TCP offers end-to-end transmissions, which means that the data is transmitted from the sender to the receiver without intermediate hops or modifications. This ensures that the data arrives at the receiver in the same order and state as it was sent by the sender. Subsequently, TCP provides reliable transmissions, which means that it uses error detection and recovery mechanisms to ensure that the data is transmitted accurately and efficiently. TCP detects and retransmits any lost or corrupted packets, and it also implements flow control mechanisms to prevent network congestion and optimize data transfer rates. Finally, TCP supports inter-process communication, which means that it allows applications running on different hosts to exchange data and communicate with each other. TCP provides a socket interface that enables applications to establish a connection and exchange data over the network. This makes TCP an essential protocol for a wide range of applications, including web browsing, email, file transfers, and many others. (Loshin, 2003)

As TCP relies on a connection-oriented approach, two participants are necessary to establish the connection. Furthermore, since TCP ensures reliable data transport, all data transmissions must be acknowledged once they have been received. Therefore, the following sequence of steps is required. Firstly, to establish a TCP circuit between two processes, the initiating process sends a request message that does not contain an acknowledgement since it is requesting to start a circuit, which may not be approved. The message usually includes information such as the requesting process's socket and the intended recipient's socket. Secondly, if a process is willing to open the requested socket, it responds with an acknowledgement, signaling that the circuit is halfway complete. The final step in establishing the connection requires host(A) to send an acknowledgement to host(B). At this point, both ends of the circuit have sent and received data along with their corresponding acknowledgements (Sarolahti, Koponen, Kompella, & Ylianttila, 2002).

2.3 TCP Congestion Control Algorithm Schemes

TCP is the most widely used transport protocol on the internet today and has been evolving for over four decades (Cerf & Kahn, 1974). One of its crucial features is its congestion control algorithms, which are vital in maintaining network stability during high traffic periods. When a packet loss is detected by a TCP sender, its transmission rate is reduced as per the congestion control principles, assuming that the packet was dropped due to congestion in the network (Pasi & Kuznetsov, 2002). When the level of congestion in a network becomes very high, packets may be delayed or lost. As a result, TCP will retransmit the lost packets, potentially leading to even more congestion. Without proper congestion control algorithms, the network may become overloaded and eventually fail, leading to a significant decrease in performance known as congestion collapse (Kozierok, 2005).

TCP congestion control mechanisms were created to address network congestion in wired networks. These methods come into play when congestion occurs, and their objective is to regulate the transmission rate to avoid network congestion. In the early TCP standard, RFC 793, there was little mention of TCP congestion control mechanisms due to implementation challenges in detecting congestion. However, subsequent upgrades, such as those described in (Loshin, 2003), elaborated on the mechanisms discussed in RFC 793. (Allman, 2009) introduced several new congestion control algorithms, including TCP Slow Start, Congestion Avoidance, Fast Retransmit, and Fast Recovery.

For the TCP congestion control algorithm to effectively manage congestion, it is necessary to first detect or anticipate congestion through various methods. This is typically achieved by measuring network delays, using network-supported Explicit Congestion Notification (ECN), or identifying lost packets, which are acknowledged through additional ACKs of previously transmitted packets (Stevens, 1994). Whenever the congestion or coming congestion is detected, sending speed will be slowed down under window control mechanism. To represent the estimated available capacity of the network, a new value is employed, referred to as the Congestion Window value, or *cwnd* for short. The actual usable window size of the sender, denoted by W, is then calculated as the minimum value between the receiver's advertised window size (*awnd*) and *cwnd* (Stevens, 1994).

W = min(*cwnd*, awnd) (1)

Then, the discussion turns to two primary TCP algorithms: Slow Start and Congestion Avoidance, which were initially introduced (Allman, 2009).

2.3.1 Slow Start and Congestion Avoidance

The slow start and congestion avoidance algorithms are established in situations where a fresh TCP connection is established, or a loss is identified because of a Retransmission Timeout (RTO). Additionally, it can be triggered when a transmitting TCP becomes inactive for a period of time (Stevens, 1994). To regulate the quantity of data being sent into the network, a TCP sender must utilize Congestion Avoidance and Slow Start algorithms. These algorithms necessitate the addition of two variables to the TCP per-connection state (Allman, 2009). When a TCP connection is initiated, it enters the slow start phase by transmitting a specific number of segments, known as the Initial Window (IW), after the SYN exchange (Stevens, 1994). While the original value of IW was one Sender Maximum Segment Size (SMSS), according to RFC5681, a larger value is now permitted. The SMSS refers to the largest segment that the sender can send, exclusive of the TCP/IP headers and options (Allman, 2009).

The congestion window size should remain unchanged by the SYN/ACK and its acknowledgment, as specified in RFC3390. In addition, if either the SYN or SYN/ACK packet is lost, the sender must utilize an initial window of one segment that contains no more than SMSS bytes after transmitting the SYN correctly (Stevens, 1994). Slow Start Threshold (*ssthresh*) is a congestion control algorithm parameter. It is the threshold limit on the size of the congestion window during the slow start phase of TCP's congestion control algorithm. The slow start phase occurs when a new TCP connection is established or when TCP recovers from a loss event.

When the value of *cwnd* is less than ssthresh, the slow start algorithm is utilized, whereas the congestion avoidance algorithm is utilized when *cwnd* is greater than *ssthresh*. If cwnd and *ssthresh* are equal, the sender may choose to employ either the slow start or the congestion avoidance algorithm. (Allman, 2009)

During slow start, TCP increases the value of *cwnd* by a maximum of SMSS bytes for each received ACK that acknowledges new data. The slow start phase concludes when either *cwnd* surpasses *ssthresh* or congestion is detected. (Allman, 2009)

On the other hand, while in congestion avoidance mode, *cwnd* is increased by approximately one full-sized segment for each Round-Trip Time (RTT). The congestion avoidance mode persists until congestion is identified (Allman, 2009). This algorithm is aimed at gaining more capacity by incrementing *cwnd* by nearly one segment for each window's worth of data that is transferred successfully from the sender to the receiver (Stevens, 1994).The fundamental rules for increasing *cwnd* during congestion avoidance are: It is allowable to increase *cwnd* by SMSS bytes, it is recommended to increase *cwnd* per equation (3) at least once per RTT, *cwnd* should not be increased by more than SMSS bytes. (Allman, 2009)

If a TCP sender detects segment loss through the retransmission timer and the concerned segment has not yet been resent through the retransmission timer, the value of *ssthresh* must be adjusted to no more than the value provided in equation. (Allman, 2009)

2.3.2 Fast Retransmit and Fast Recovery

Fast retransmit and fast recovery are algorithms used in TCP congestion control to improve network performance. Fast retransmit detects and recovers from lost packets by triggering retransmission based on the arrival of duplicate acknowledgments. Fast recovery helps to avoid congestion collapse by allowing the sender to continue sending data after the loss of a packet without waiting for a timeout.

Duplicate acknowledgments can result from either lost segments or out-of-order delivery. In the past, TCP implementations had to wait for a timer to expire before retransmitting the lost segment if it was indeed lost. However, if the segment was delivered out of order, it would eventually be acknowledged, and the sender would not have to retransmit it. (Loshin, 2003)

To identify and recover from lost data, the TCP sender is advised to employ the "fast retransmit" algorithm, which utilizes duplicate acknowledgments to detect data loss. When three duplicate acknowledgments are received, the fast retransmit algorithm infers the loss of a segment and triggers its retransmission without waiting for the retransmission timer to expire (Allman, 2009). Once the fast retransmit algorithm sends the presumed missing segment, the "fast recovery" algorithm takes over the transmission of new data until a non-duplicate acknowledgment is received. Instead of resetting *cwnd* only one segment size, it is reset to the last value of the *ssthresh* (Stevens, 1994). Slow start is not initiated because the receipt of duplicate acknowledgments not only signals the loss of a segment but also suggests that segments are still leaving the network. (Allman, 2009)

During the recovery period, the fast recovery algorithm enables the congestion window to increase by one SMSS for every received acknowledgment (ACK). This temporary inflation of *cwnd* allows for the transmission of an additional new packet for each ACK received until a non-duplicate or "good" ACK is received. At that point, TCP exits recovery mode and reduces the congestion window back to its original size (Stevens, 1994).

2.3.3 Reno

Reno is a variant of the TCP congestion control algorithm that uses the Additive Increase Multiplicative Decrease (AIMD) approach. Reno TCP is the default congestion control algorithm used in many operating systems, including Linux and

Windows. It has been widely studied and improved upon and is considered a cornerstone of TCP congestion control. (Floyd, 2001)

Reno consists of three main components: Slow Start, Congestion Avoidance, and Fast Retransmit. A disadvantage of the Tahoe algorithm is that every time a retransmission occurs, TCP reverts to the Slow Start phase. This results in underutilization of the available bandwidth. (Stevens, 1994)

When Reno receives three duplicate ACKs, it performs a fast retransmit and enters a phase called fast recovery. During this phase, the congestion window is halved instead of being set to 1 MSS, and the *ssthresh* is set to the new congestion window. The slow start phase is skipped in this case (Ross, Kurose, & Keith, 2012).

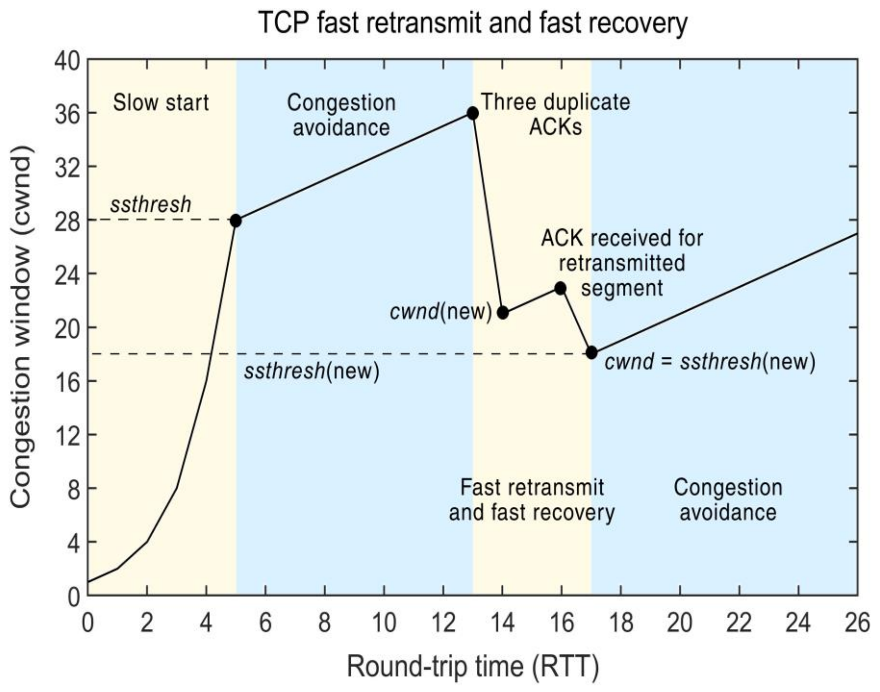
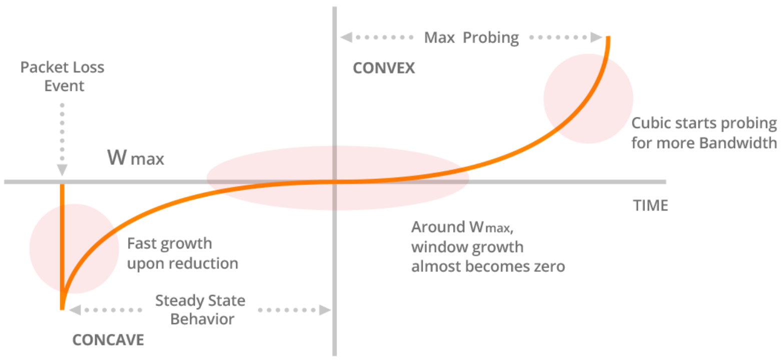


Figure 1. Cwnd behaviour in TCP Reno

2.3.4 Cubic

CUBIC (Congestion Control for TCP) is a TCP congestion control algorithm that was introduced in 2006 (Ha, Rhee, & Lisong, CUBIC: A New TCP-Friendly High-Speed TCP Variant, 2008b). CUBIC uses a new congestion control approach based on the analysis of the cubic function that allows it to increase the congestion window size more aggressively and more fairly than other algorithms, while reducing the delay and maintaining high network utilization. CUBIC uses a cubic equation that considers the time elapsed since the last congestion event. Additionally, CUBIC uses the concave and convex aspects of the cubic function to determine how much to increase the congestion window. Figure 2 shows the growth function of CUBIC (Ha, Rhee, & Lixia, CUBIC, 2008a).



Packet Loss Event
W max
Max Probing
CONVEX
Cubic starts probing for more Bandwidth
TIME
Around Wmax, window growth almost becomes zero
Fast growth upon reduction
Steady State Behavior
CONCAVE

Figure 2. Growth function in CUBIC

CUBIC employs the given mathematical function to determine the growth of its congestion window (Ha, Rhee, & Lixia, CUBIC, 2008a).

Once receiving an ACK in congestion avoidance, CUBIC computes the window growth rate during the next RTT period using equation (4) and sets W (t+ RTT) as a target value of *cwnd.* CUBIC operates in three modes depending on the value of *cwnd*. When *cwnd* is smaller than the window size that TCP would achieve at time t after the last loss event, it is in the TCP mode. If *cwnd* is less than W_{max}, it is in the concave region, while if *cwnd* is greater than W_{max}, it is in the convex region.

2.3.5 Westwood

Westwood is a TCP congestion control algorithm designed to improve the performance of TCP in high-speed and long-distance networks by addressing issues related to delayed acknowledgment and packet loss. It adjusts the sending rate of TCP based on the available network bandwidth and the round-trip time. TCP Westwood utilizes information from the ACK stream to improve congestion control parameters such as *ssthresh* and *cwnd*. It estimates an "Eligible Rate" which is used to update these parameters during loss indication or its "Agile Probing" phase. It also employs Persistent Non-Congestion Detection (PNCD) to detect prolonged lack of congestion and initiate the Agile Probing phase to quickly utilize dynamic bandwidth (Mascolo, Casetti, Gerla, Sanadidi, & Wang, 2001). The Westwood algorithm adjusts the interval size based on the congestion level. If congestion is low, the interval will be small, and vice versa. When a packet is dropped, the Westwood algorithm computes a new bandwidth delay product (BDP) and assigns it to *cwnd* rather than reducing it by half (Stevens, 1994).

2.3.6 BBR

In 2016, Google developed a congestion control algorithm called Bottleneck Bandwidth and Round-trip propagation time (BBR) (Cardwell, Cheng, Gunn, Yeganeh, & Jacobson, 2017). BBR is designed for TCP and aims to achieve high throughput and low latency simultaneously. It achieves this by continuously estimating the available bandwidth and RTT. BBR adjusts the sending rate of data packets based on continuous estimation of available bandwidth and RTT. It is highly effective in networks with high bandwidth and variable RTT, including data center and cellular networks.

Unlike most congestion control algorithms that are based on packet loss to detect congestion and adjust transmission rates, BBR and TCP Vegas are model-based algorithms. Instead of relying on packet loss, BBR estimates the maximum bandwidth and round-trip time of the network to build a model. As packets are delivered, acknowledgments are received, which produce rate samples to record the amount of data delivered over time (Yi, 2022). With the increase in network interface controller speed from megabit per second to gigabit per second, buffer bloat-induced latency is becoming a more accurate indicator of maximum throughput than packet loss. As a result, model-based CCAs such as BBR, which offer higher throughput and lower latency, are becoming a more reliable alternative to popular loss-based algorithms like Cubic (Yi, 2022).

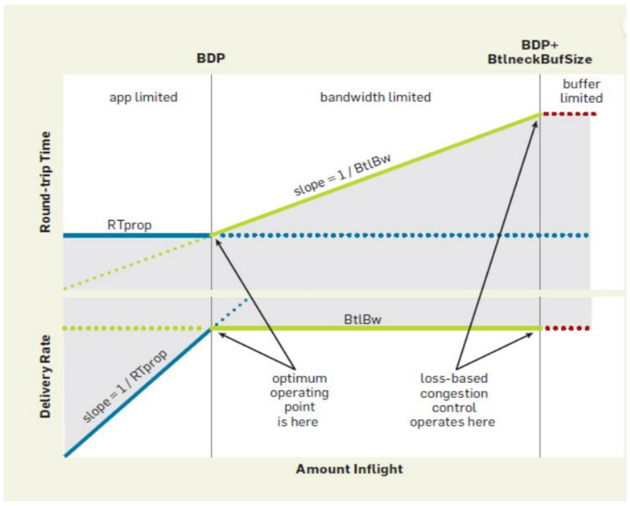


Figure 3. Delivery rate and round-trip time

The graph depicted in Figure 3 displays the changes in delivery rate and RTT according to the amount of data that has been sent but not yet acknowledged. The constraints of RTprop are represented by blue lines, BtlBw constraints by green lines, and the bottleneck buffer by red lines. It is impossible to operate in the regions that are shaded since they would violate at least one constraint. The transitions between constraints result in three distinct regions, namely, app-limited, bandwidth-limited, and buffer-limited, each having different behavior (Cardwell, Cheng, Gunn, Yeganeh, & Jacobson, 2017).

2.3.7 HTCP

HTCP, or Hamiltonian TCP, is a congestion control algorithm used in TCP to optimize network performance by balancing the transmission rate of data packets with the amount of data already in transit and the capacity of the network. It uses AIMD to control TCP's congestion window size (Armitage, Stewart, Welzl, & John, 2008). HTCP's unique approach to congestion control is based on a mathematical model derived from Hamiltonian mechanics, which allows it to more accurately estimate the amount of data that can be safely transmitted without overwhelming the network. This results in improved network utilization and a more stable flow of data, leading to faster and more reliable communication.

In TCP research, it is a common approach to generalize Standard TCP2 by defining the growth and reduction of *cwnd* in terms of two constants: alpha (α) and beta (β). By knowing the SMSS of the sender, it is possible to determine that during the congestion avoidance phase, the *cwnd* will increase by no more than α × SMSS per RTT. After a congestion event, the *cwnd* will be reduced to β × *cwnd*. This method provides a more precise specification of the congestion control algorithm. H-TCP's initial reaction to a congestion event is like that of NewReno, where alpha (α) is set to 1 until the value of delta (δ) exceeds a specified threshold called δ_1. When δ exceeds δ_1, alpha is calculated using a quadratic formula like equation (7).

$$\alpha = 1 + 10 \times (\delta - \delta_1) + (\frac{\delta-\delta_1}{2})^2 \quad (7)$$

δ_1 represents the time that must elapse after a congestion event before H-TCP’s own α increase function comes into (Armitage, Stewart, Welzl, & John, 2008).In H-TCP, the recommended value for the threshold δ_1 is 1 second, as recommended by Hamilton. This means that after a congestion event, H-TCP behaves similarly to NewReno and sets the value of α to 1 for a duration of one second. After one second, H-TCP's *cwnd* grows more aggressively if no further congestion events occur.

2.4 Linux Commands to Alternate TCP Congestion Control Schemes

In the Linux operating system, numerous CCAs are available. Some CCAs are included in the kernel by default, while others require compiling from the source code to be used. This flexibility allows users to choose and customize the appropriate CCA for their specific network needs. This implementation contains unique features that differentiate it from other TCP implementations, which may be of interest to protocol designers working with TCP (Pasi & Kuznetsov, 2002).

The RPi4 was used in this project came with Raspbian Bullseye (kernel version number: 5.10). This OS supports TCP Reno and Cubic. If we want to display a catalogue of available CCA, we can execute a specific command (Jason, 2021):

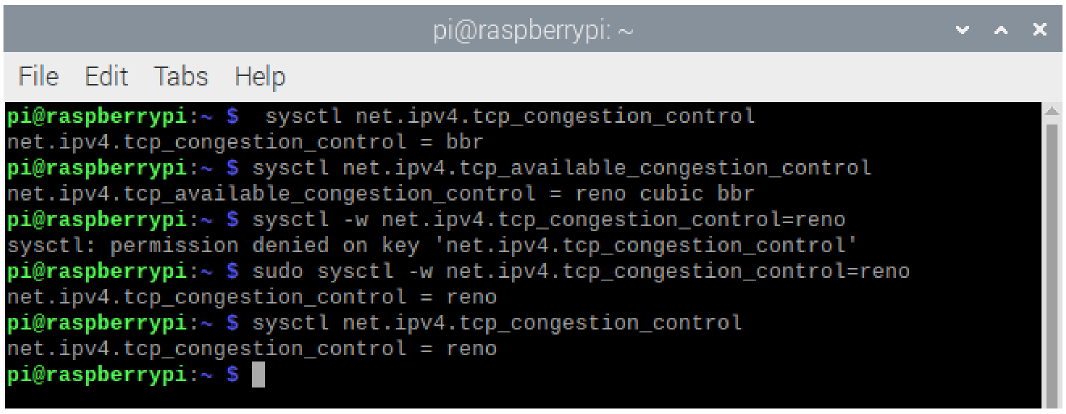
```
sysctl net.ipv4.tcp_available_congestion_control
```

Alternatively, To the purpose is to verify which CCA is presently being utilized by the device, the following command can be executed (Jason, 2021):

```
sysctl net.ipv4.tcp_congestion_control
```

The CCA mentioned in this list can be readily implemented onto the operating system. To choose and apply a different CCA in the kernel, we can employ this command (Jason, 2021):

```
sudo sysctl -w net.ipv4.tcp_congestion_control=<CCA name>
```

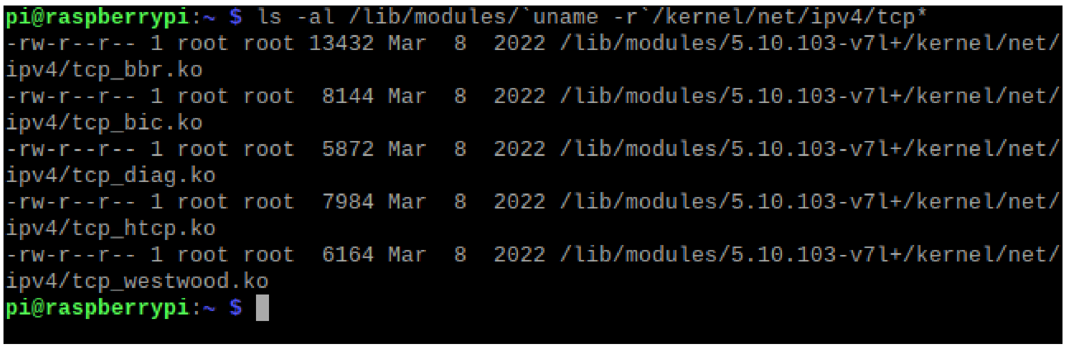


```
pi@raspberrypi:~ $  sysctl net.ipv4.tcp_congestion_control
net.ipv4.tcp_congestion_control = bbr
pi@raspberrypi:~ $ sysctl net.ipv4.tcp_available_congestion_control
net.ipv4.tcp_available_congestion_control = reno cubic bbr
pi@raspberrypi:~ $ sysctl -w net.ipv4.tcp_congestion_control=reno
sysctl: permission denied on key 'net.ipv4.tcp_congestion_control'
pi@raspberrypi:~ $ sudo sysctl -w net.ipv4.tcp_congestion_control=reno
net.ipv4.tcp_congestion_control = reno
pi@raspberrypi:~ $ sysctl net.ipv4.tcp_congestion_control
net.ipv4.tcp_congestion_control = reno
pi@raspberrypi:~ $ 
```

Figure 4. Apply new CCA for OS

Within the Linux system, there is an abundance of Congestion Control Algorithms that are compatible with the operating system. However, not all of them have been loaded onto the kernel, which means that some may not appear in the available command check. In order to display a list of all the CCA that have been implemented in the kernel but are not yet loaded, this command can be used (Jason, 2021):

```
ls -al /lib/modules/`uname -r`/kernel/net/ipv4/tcp*
```

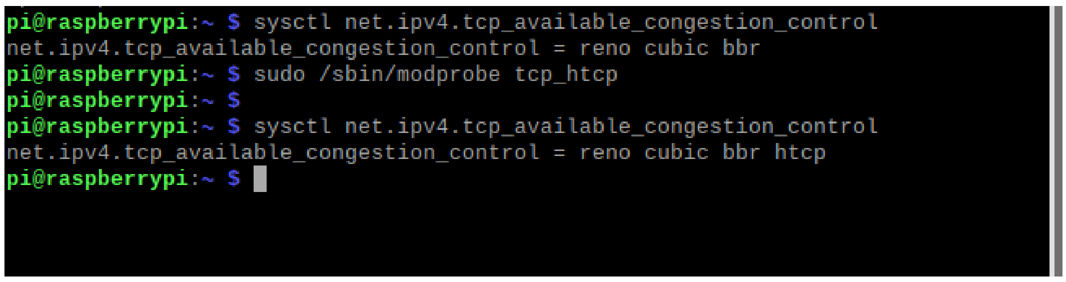


```
pi@raspberrypi:~ $ ls -al /lib/modules/`uname -r`/kernel/net/ipv4/tcp*
-rw-r--r-- 1 root root 13432 Mar  8  2022 /lib/modules/5.10.103-v7l+/kernel/net/
ipv4/tcp_bbr.ko
-rw-r--r-- 1 root root  8144 Mar  8  2022 /lib/modules/5.10.103-v7l+/kernel/net/
ipv4/tcp_bic.ko
-rw-r--r-- 1 root root  5872 Mar  8  2022 /lib/modules/5.10.103-v7l+/kernel/net/
ipv4/tcp_diag.ko
-rw-r--r-- 1 root root  7984 Mar  8  2022 /lib/modules/5.10.103-v7l+/kernel/net/
ipv4/tcp_htcp.ko
-rw-r--r-- 1 root root  6164 Mar  8  2022 /lib/modules/5.10.103-v7l+/kernel/net/
ipv4/tcp_westwood.ko
pi@raspberrypi:~ $
```

Figure 5. List all implemented CCA in the current OS.

After obtaining a list of available but unloaded CCA, we can easily load the desired CCA sing the "modprobe" command:

```
sudo /sbin/modprobe <tcp_CCA>
```



```
pi@raspberrypi:~ $ sysctl net.ipv4.tcp_available_congestion_control
net.ipv4.tcp_available_congestion_control = reno cubic bbr
pi@raspberrypi:~ $ sudo /sbin/modprobe tcp_htcp
pi@raspberrypi:~ $
pi@raspberrypi:~ $ sysctl net.ipv4.tcp_available_congestion_control
net.ipv4.tcp_available_congestion_control = reno cubic bbr htcp
pi@raspberrypi:~ $
```

Figure 6. Loading new CCA to kernel.

Figure 6 illustrates that our new CCA has been successfully implemented in our kernel, enabling us to conveniently utilize it for TCP communication objectives.

3 TESTING IMPLEMENTATION

In this section, the purpose is to present a comprehensive solution for effectively assessing and evaluating the performance of various TCP congestion control avoidance schemes. This involves proposing suitable methodologies and software tools that can be utilized specifically for the purpose of testing these schemes. Furthermore, this chapter encompasses an in-depth exploration of the essential equipment required for conducting the project, including detailed instructions on how to properly install and set up said equipment. Ultimately, the section will provide a detailed account of the step-by-step process involved in conducting the testing, ensuring a thorough understanding of the entire testing progress.

3.1 Proposed Solution

Our objective is to test different CCAs over a 5G link, which necessitates the installation of a 5G hat device. In this case, we are using a Raspberry Pi 4 as a 5G hat, and we can easily modify the CCA as required. The tests are carried out using a network performance measurement tool called “iperf3” which is available on any Linux system. To evaluate the TCP performance of different CCAs, we can analyze the transmission rates or speed rates to determine which algorithm performs optimally. Totally, 5 CCAs was used like discussed in section 2.3 TCP Congestion Control Algorithm Schemes.

During the testing process, it is important to note that the throughput, delay, and other relevant parameters may vary significantly and may not remain stable. Moreover, there is a possibility of congestion or data loss occurring during the testing, which can lead to erroneous results. Therefore, it is imperative to ensure the reliability and accuracy of data collection. To achieve this, each CCA has been tested over 3 different durations, as 1, 5, and 10 minutes respectively, and each duration has been tested 10 times to eliminate temporal radio link fluctuations.

In our project, our system is both Non-Standalone (NSA) our 5G system operates as a Time Division Duplex (TDD) system utilizing a 50MHz band ranging from

3.900GHz to 3.950GHz. The TDD technology enables the base station to dynamically allocate time slots for both uplink and downlink transmission using a Radio Resource Management (RRM) algorithm. As a result, the uplink and downlink speeds experienced by each mobile terminal can vary considerably. Therefore, it is important to consider both directions when analyzing the data.

The data was collected in the JSON format initially. However, organizing and analyzing data in this format can be challenging. Therefore, the data was converted from JSON to XLSX format, which can be easily handled by Excel for further analysis and interaction with the data.

In summary, to compare the data, we will calculate the average throughput in the downlink and uplink for three different intervals and the overall average of all tests. Latency will be assessed using the same method for comparison. Additionally, the behavior of throughput during the 10-minute testing period will be plotted in a chart to illustrate its reaction over time. The congestion window values will also be analyzed using this method.

3.2 Equipment and Installation

3.2.1 Equipment

For this project, The SIM8200EA-M2 was selected. The SIM8200EA-M2 is a 5G HAT module that could be attached to Raspberry Pi devices, providing them with 5G connectivity. The module is based on the Qualcomm Snapdragon X55 5G modem and supports both SA (Standalone) and NSA 5G modes. It is designed to enable developers to create 5G-powered IoT (Internet of Things) applications with ease. This modem is able to connect to any commercial 5G network with the operator’s SIM card.

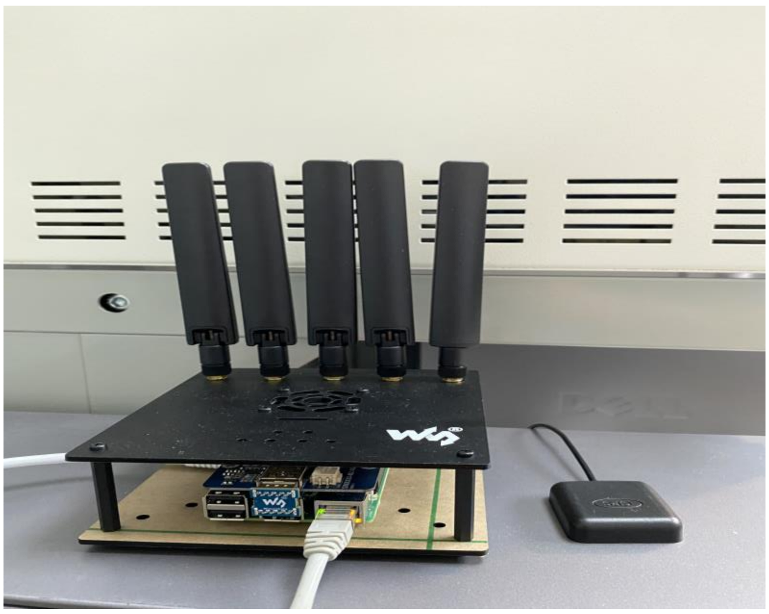


Figure 7. Sim8200ea-m2 5g device

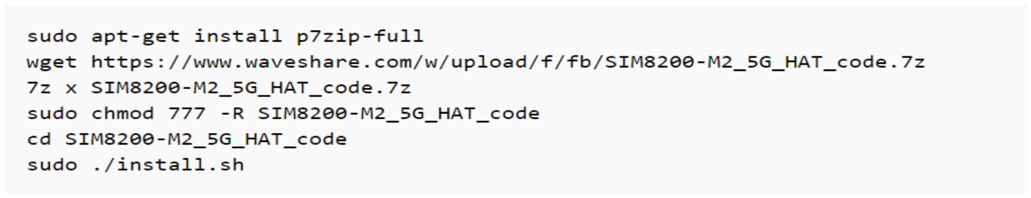
Hence, the 5G network used in this project is an experimental system from Amarisoft. The base station is called Callbox (as in Figure 8). Using such a network allows us to test 5G link performance without involving other parts of the Internet. We setup an Iperf server as edge server right beside the Callbox and connected them with a 10Gbps Ethernet switch, hence the intervention of other. Amarisoft LTE Call box is a device designed for LTE testing and network emulation. It can simulate a complete LTE network with eNodeB (base station) and EPC (core network) components, allowing for testing of LTE devices and applications in a controlled environment.

Figure 8. Arimasoft 5G base station

In this project, Iperf3 is utilized to evaluate the performance of different CCAs in the context of 5G communication. Iperf3 is a widely used network performance measurement tool that assesses the maximum achievable bandwidth on IP networks. It is commonly employed to analyse network performance, detect network issues, and resolve network bottlenecks. By using iperf3, we can easily and effectively determine the optimal CCA performance during 5G communication.

3.2.2 Installation

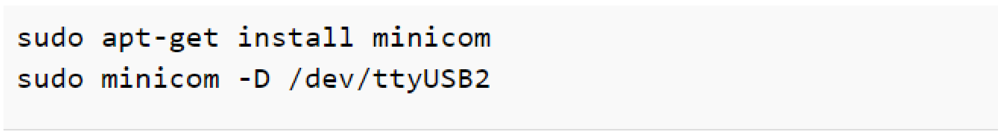
To establish a connection between the 5G hat device and the 5G base station, manufacturer of the application can be utilized. The user guide provides instructions on how to install and use the application, which can be easily installed by following the designated commands:



```
sudo apt-get install p7zip-full
wget https://www.waveshare.com/w/upload/f/fb/SIM8200-M2_5G_HAT_code.7z
7z x SIM8200-M2_5G_HAT_code.7z
sudo chmod 777 -R SIM8200-M2_5G_HAT_code
cd SIM8200-M2_5G_HAT_code
sudo ./install.sh
```

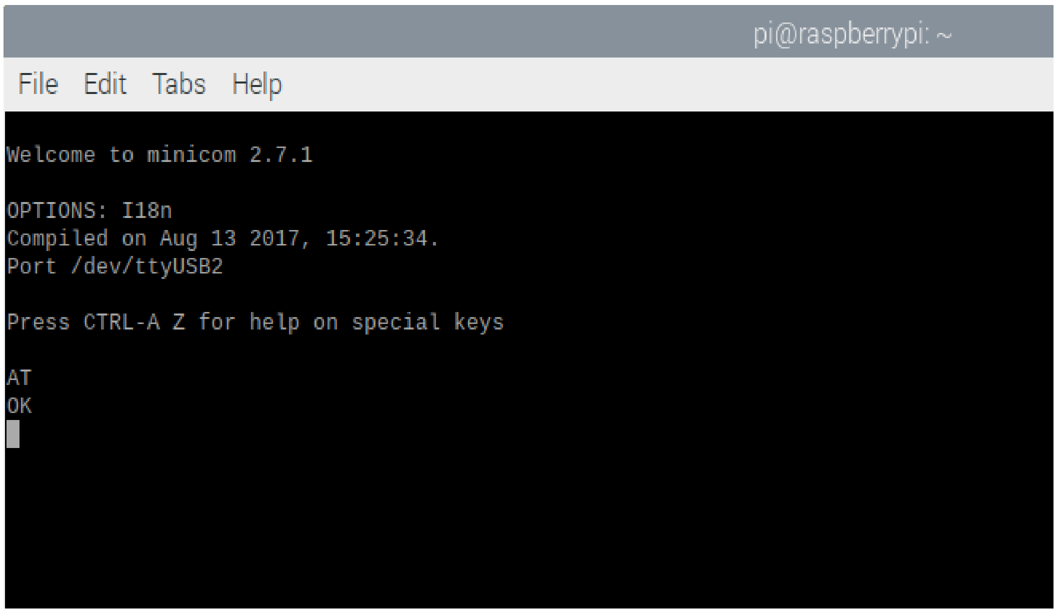
Figure 9. 5G hat driver installation on RPi4

Next, it is necessary to use the minicom application that has been developed for AT command testing:



```
sudo apt-get install minicom
sudo minicom -D /dev/ttyUSB2
```

Figure 10. AT testing command

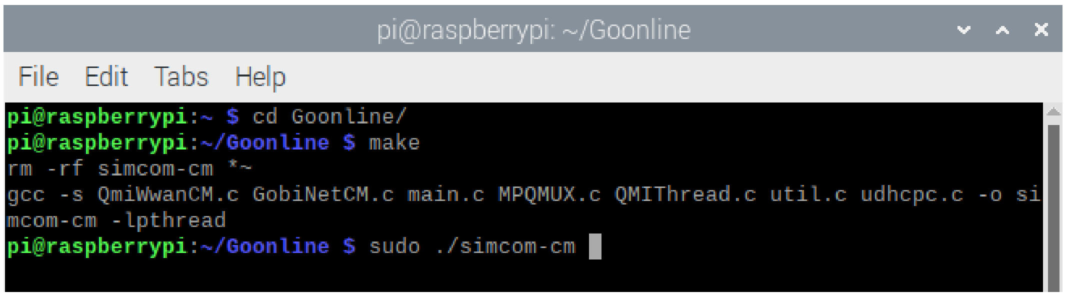


```
Welcome to minicom 2.7.1

OPTIONS: I18n
Compiled on Aug 13 2017, 15:25:34.
Port /dev/ttyUSB2

Press CTRL-A Z for help on special keys

AT
OK
```

Figure 11. AT testing in minicom interface

After a successful AT testing, a connection to the 5G network can be established using the following command (Waveshare, 2023):

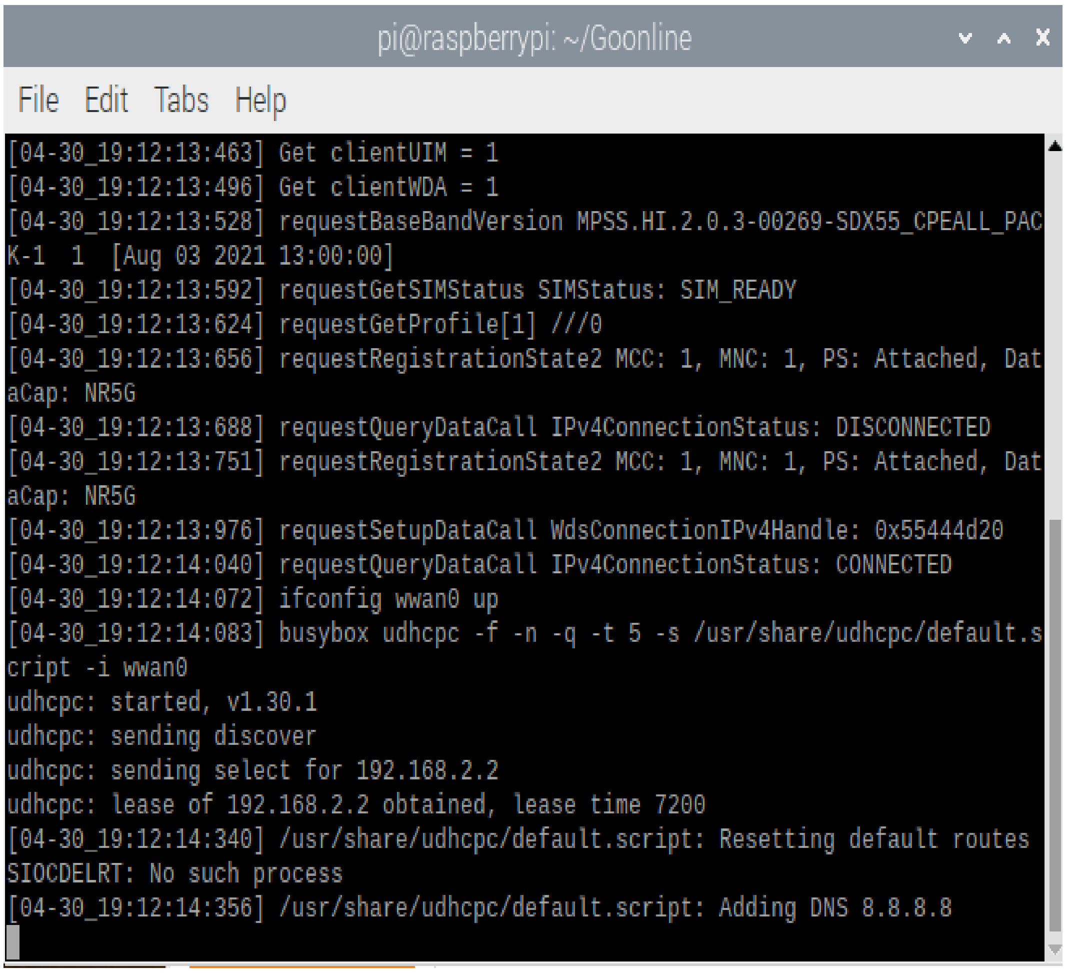
```
cd Goonline

make

sudo ./simcom-cm
```



```
pi@raspberrypi:~ $ cd Goonline/
pi@raspberrypi:~/Goonline $ make
rm -rf simcom-cm *~
gcc -s QmiWwanCM.c GobiNetCM.c main.c MPQMUX.c QMIThread.c util.c udhcpc.c -o si
mcom-cm -lpthread
pi@raspberrypi:~/Goonline $ sudo ./simcom-cm
```

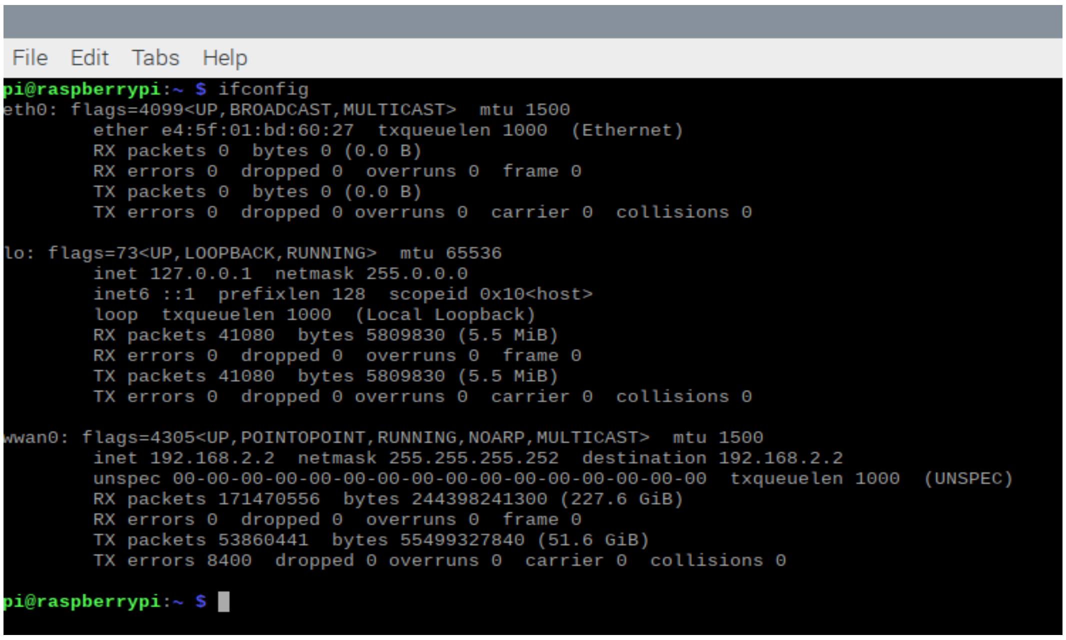
Figure 12. Connect to 5G base station.

Each time we need to connect to 5G, we can simply navigate to the "Goonline" directory and execute the application without having to create a new configuration.



```
[04-30_19:12:13:463] Get clientUIM = 1
[04-30_19:12:13:496] Get clientWDA = 1
[04-30_19:12:13:528] requestBaseBandVersion MPSS.HI.2.0.3-00269-SDX55_CPEALL_PAC
K-1  1  [Aug 03 2021 13:00:00]
[04-30_19:12:13:592] requestGetSIMStatus SIMStatus: SIM_READY
[04-30_19:12:13:624] requestGetProfile[1] ///0
[04-30_19:12:13:656] requestRegistrationState2 MCC: 1, MNC: 1, PS: Attached, Dat
aCap: NR5G
[04-30_19:12:13:688] requestQueryDataCall IPv4ConnectionStatus: DISCONNECTED
[04-30_19:12:13:751] requestRegistrationState2 MCC: 1, MNC: 1, PS: Attached, Dat
aCap: NR5G
[04-30_19:12:13:976] requestSetupDataCall WdsConnectionIPv4Handle: 0x55444d20
[04-30_19:12:14:040] requestQueryDataCall IPv4ConnectionStatus: CONNECTED
[04-30_19:12:14:072] ifconfig wwan0 up
[04-30_19:12:14:083] busybox udhcpc -f -n -q -t 5 -s /usr/share/udhcpc/default.s
cript -i wwan0
udhcpc: started, v1.30.1
udhcpc: sending discover
udhcpc: sending select for 192.168.2.2
udhcpc: lease of 192.168.2.2 obtained, lease time 7200
[04-30_19:12:14:340] /usr/share/udhcpc/default.script: Resetting default routes
SIOCDELRT: No such process
[04-30_19:12:14:356] /usr/share/udhcpc/default.script: Adding DNS 8.8.8.8
```

Figure 12. 5G connecting application interface.



```
pi@raspberrypi:~ $ ifconfig
eth0: flags=4099<UP,BROADCAST,MULTICAST>  mtu 1500
        ether e4:5f:01:bd:60:27  txqueuelen 1000  (Ethernet)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 41080  bytes 5809830 (5.5 MiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 41080  bytes 5809830 (5.5 MiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

wwan0: flags=4305<UP,POINTOPOINT,RUNNING,NOARP,MULTICAST>  mtu 1500
        inet 192.168.2.2  netmask 255.255.255.252  destination 192.168.2.2
        unspec 00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00  txqueuelen 1000  (UNSPEC)
        RX packets 171470556  bytes 244398241300 (227.6 GiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 53860441  bytes 55499327840 (51.6 GiB)
        TX errors 8400  dropped 0 overruns 0  carrier 0  collisions 0

pi@raspberrypi:~ $ 
```

Figure 13. “ifconfig” shows a “wwan0” interface with IP address 192.168.2.2 is added to the RPi4 Internet interface.

The successful connection of the 5G hat to the edge server was confirmed by the establishment of a 5G link, and the hat was assigned the IP address 192.168.2.2 at interface wwan0, indicating a successful connection to the 5G base station.

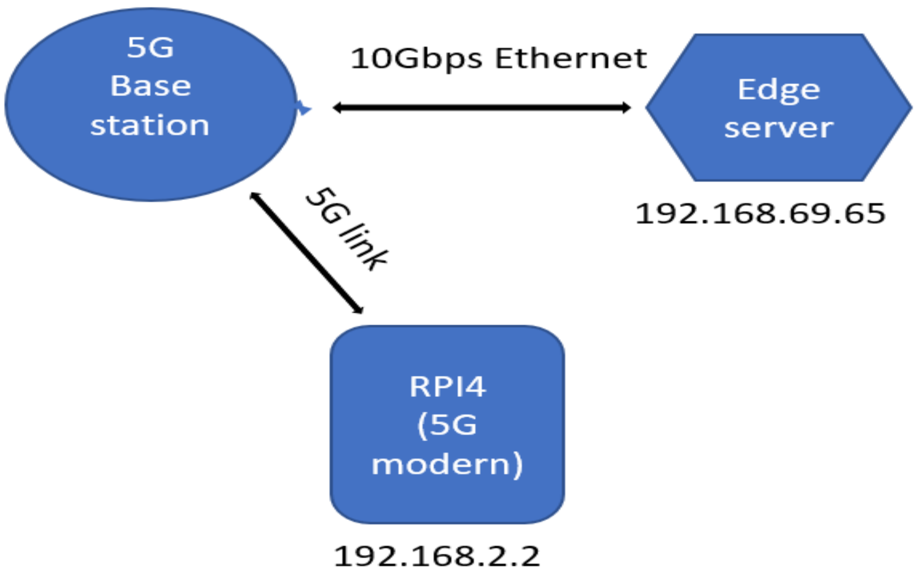
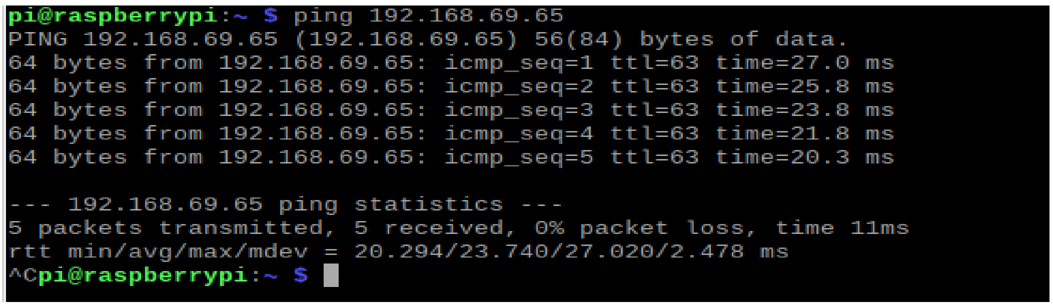


Figure 14. The network topology

Based on Figure 15, the IP address assigned to the edge server in this network topology is 192.168.69.65. The Raspberry Pi is linked to a 5G network and assigned the IP address 192.168.2.2. Communication between the edge server and Raspberry Pi takes place at a frequency of 3.9 GHz. Additionally, the image presents the signal power for both transmission and reception signals.

Our next step is to verify the connection between the edge server and the Raspberry Pi by using the ping command to send a message to the IP address of the edges ever, which is 192.168.69.65:



```
pi@raspberrypi:~ $ ping 192.168.69.65
PING 192.168.69.65 (192.168.69.65) 56(84) bytes of data.
64 bytes from 192.168.69.65: icmp_seq=1 ttl=63 time=27.0 ms
64 bytes from 192.168.69.65: icmp_seq=2 ttl=63 time=25.8 ms
64 bytes from 192.168.69.65: icmp_seq=3 ttl=63 time=23.8 ms
64 bytes from 192.168.69.65: icmp_seq=4 ttl=63 time=21.8 ms
64 bytes from 192.168.69.65: icmp_seq=5 ttl=63 time=20.3 ms

--- 192.168.69.65 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 11ms
rtt min/avg/max/mdev = 20.294/23.740/27.020/2.478 ms
^Cpi@raspberrypi:~ $
```

Figure 15. Ping to edge for connection testing

Once the ping command confirms a successful connection between the edge server and the Raspberry Pi, testing can be initiated using the iperf3 tool.

The current scenario involves the utilization of various parameters within the iperf3 tool to carry out testing that is more precise and efficient:

Table 1. The list of iperf options used in the tests.

Parameter	Utilizing
-c <server IP address>	To set host device as client
-p <port number>	Set port number for this connection
-i <time (in second)>	a time interval between each test
-t <time (in second)>	to specify the duration of the test
-s	to start the iperf3 tool in server mode
-C <CCA>	to set the congestion control algorithm

-J	to specify that the output should be in JSON format
-R	to perform a reverse(downlink) test

Initially, a server was established on the edge server using iperf3, with the port number designated as 7778, and the reporting interval was configured to be 1 second to enable better monitoring of the data transfer:

```
iperf3 -s -i 1 -p 7778
```

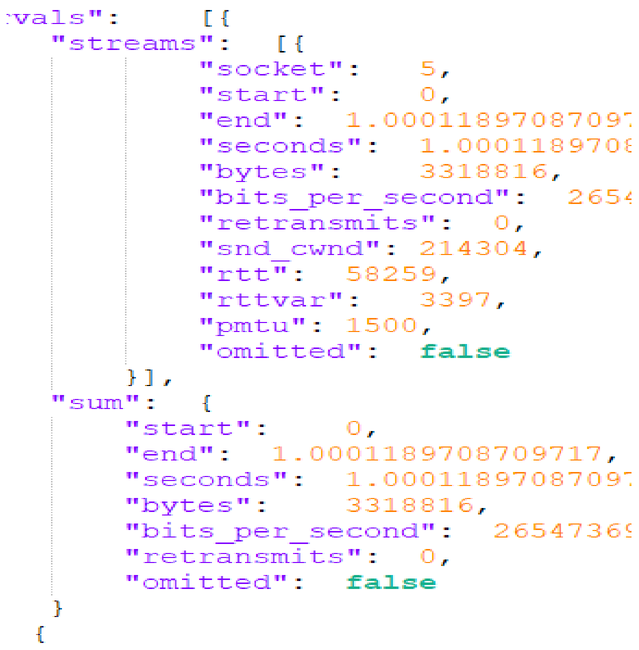
One the client side, which is RPi4, an iperf connection was established to the server. To begin with, I established a connection was established to the iperf3 server, which was assigned the IP address of 192.168.69.65, using the same port number as the server, which is 7778. The reporting interval is also set as 1 second to ensure that dynamic changes can be observed during the tests. Additionally, the -J flag was utilized to save the collected data in the JSON file format which would be stored in the current working directory. This approach enables the uplink to be tested with minimal effort in default settings, while testing the downlink can be achieved by using the -R flag. The uplink refers to the transmission of data from the client to the server, while the downlink refers to the transmission of data from the server to the client. Through this method, the client testing phase can be conducted with relative ease and efficiency. Following by this command:

```
iperf3 –c 192.168.69.65 -p 7778 -i 1 -t 600 -C cubic -J >
test.json  (uplink)

iperf3 –c 192.168.69.65 -p 7778 -i 1 -t 600 -C cubic -R -J >
test.json (downlink)
```

The provided iperf commands are utilized on the client device (RPi4). Each command designates a client (-c) connected to a server IP (192.168.69.65) with port 7778 (-p). The reporting interval is set to 1 second (-i), the test duration is 600 seconds (-t), and the cubic CCA algorithm (-C) is employed. The results of the test are saved in a JSON file (-J > test.json). The second command specifically focuses on evaluating the downlink performance (-R). For a more comprehensive understanding of each iperf option, please refer to Table 1.

3.3 Testing Progress

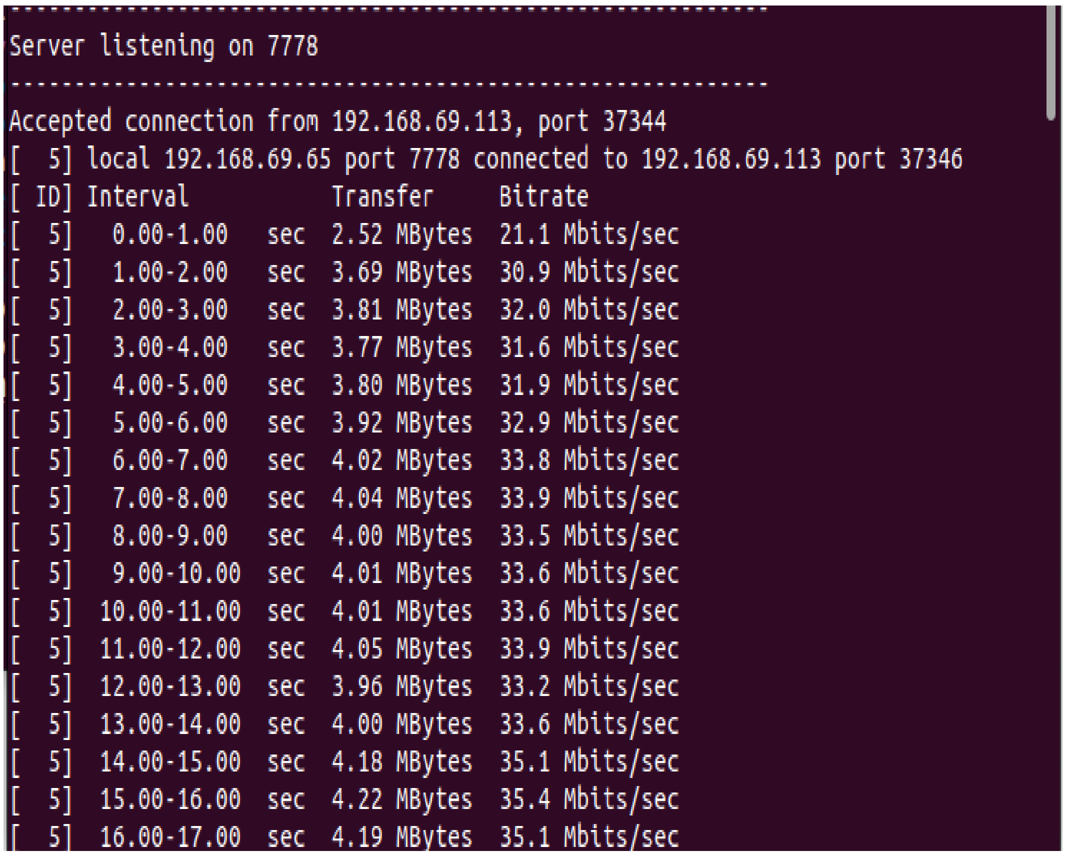
“iperf” is able to save instantaneous results and many other details in a JSON-formatted file. Figure 17 shows a part of such a file.



```
vals":      [{
    "streams":   [{
                "socket":    5,
                "start":     0,
                "end":   1.00011897087097
                "seconds":   1.000118970
                "bytes":     3318816,
                "bits_per_second":   2654
                "retransmits":   0,
                "snd_cwnd": 214304,
                "rtt":   58259,
                "rttvar":    3397,
                "pmtu": 1500,
                "omitted":   false
            }],
    "sum":   {
            "start":     0,
            "end":   1.000118970870 9717,
            "seconds":   1.00011897087097
            "bytes":     3318816,
            "bits_per_second":   26547369
            "retransmits":   0,
            "omitted":   false
    }
  {
```

Figure 16. Data in JSON format

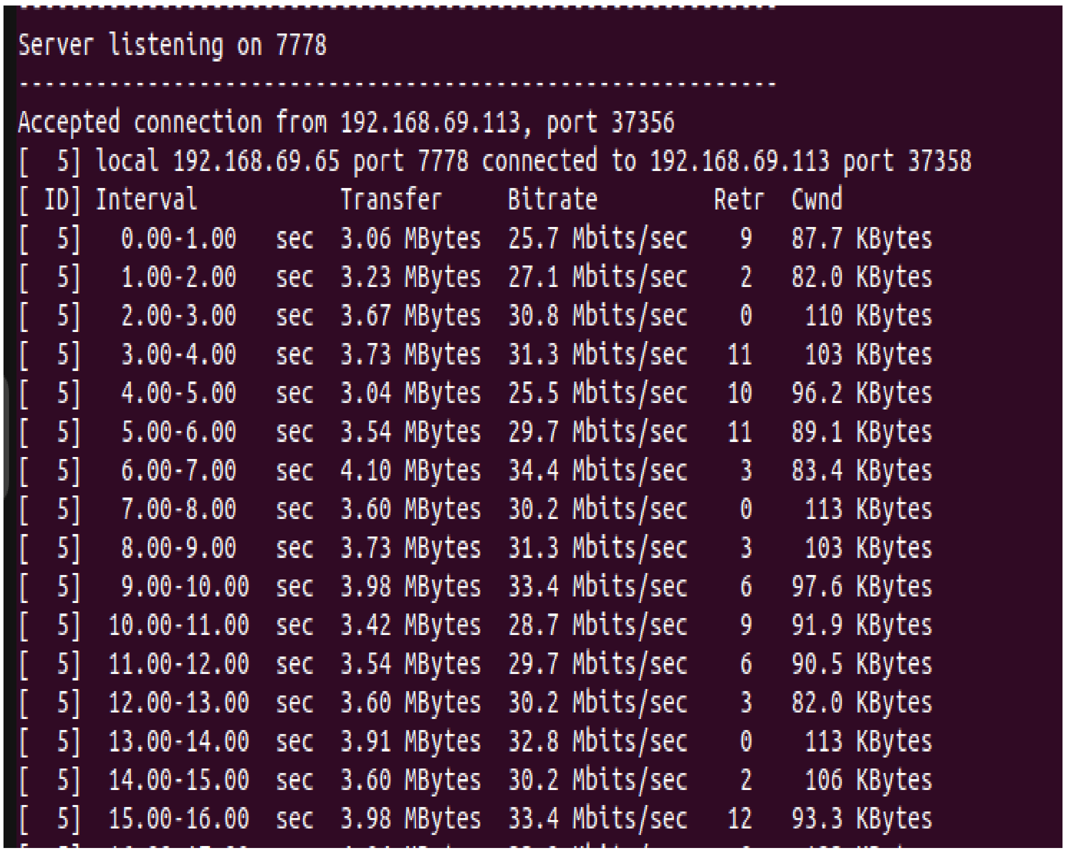
The data during testing using iperf3 is available in two formats, namely the JSON file and the server output. Upon running the tests, the server interface displays the data in real-time, providing a continuous update of the intervals and throughput (bit rate) every second. Additionally, the interface also shows the settings that were configured prior to running the tests. This real-time display of data allows for a thorough and precise assessment of the performance of the network during testing.



```
-----------------------------------------------------------
Server listening on 7778
-----------------------------------------------------------
Accepted connection from 192.168.69.113, port 37344
[  5] local 192.168.69.65 port 7778 connected to 192.168.69.113 port 37346
[ ID] Interval           Transfer     Bitrate
[  5]   0.00-1.00   sec  2.52 MBytes  21.1 Mbits/sec
[  5]   1.00-2.00   sec  3.69 MBytes  30.9 Mbits/sec
[  5]   2.00-3.00   sec  3.81 MBytes  32.0 Mbits/sec
[  5]   3.00-4.00   sec  3.77 MBytes  31.6 Mbits/sec
[  5]   4.00-5.00   sec  3.80 MBytes  31.9 Mbits/sec
[  5]   5.00-6.00   sec  3.92 MBytes  32.9 Mbits/sec
[  5]   6.00-7.00   sec  4.02 MBytes  33.8 Mbits/sec
[  5]   7.00-8.00   sec  4.04 MBytes  33.9 Mbits/sec
[  5]   8.00-9.00   sec  4.00 MBytes  33.5 Mbits/sec
[  5]   9.00-10.00  sec  4.01 MBytes  33.6 Mbits/sec
[  5]  10.00-11.00  sec  4.01 MBytes  33.6 Mbits/sec
[  5]  11.00-12.00  sec  4.05 MBytes  33.9 Mbits/sec
[  5]  12.00-13.00  sec  3.96 MBytes  33.2 Mbits/sec
[  5]  13.00-14.00  sec  4.00 MBytes  33.6 Mbits/sec
[  5]  14.00-15.00  sec  4.18 MBytes  35.1 Mbits/sec
[  5]  15.00-16.00  sec  4.22 MBytes  35.4 Mbits/sec
[  5]  16.00-17.00  sec  4.19 MBytes  35.1 Mbits/sec
```

Figure 17. Server interface in Cubic uplink testing

Figure 18 displays the server interface utilized during uplink testing, which exhibits fundamental data obtained during the test. However, in the case of downlink testing, the server interface shows more extensive information, such as the retransmission rate and congestion window values. This distinction in the type of data displayed is since the congestion window value and retransmission rate are set only at the sender, and the retransmission rate is sent from the sender rather than the receiver. Therefore, to examine these metrics, the JSON file obtained during the downlink testing phase is analysed.



```
-----------------------------------------------------------
Server listening on 7778
-----------------------------------------------------------
Accepted connection from 192.168.69.113, port 37356
[  5] local 192.168.69.65 port 7778 connected to 192.168.69.113 port 37358
[ ID] Interval           Transfer     Bitrate         Retr  Cwnd
[  5]   0.00-1.00   sec  3.06 MBytes  25.7 Mbits/sec    9   87.7 KBytes
[  5]   1.00-2.00   sec  3.23 MBytes  27.1 Mbits/sec    2   82.0 KBytes
[  5]   2.00-3.00   sec  3.67 MBytes  30.8 Mbits/sec    0    110 KBytes
[  5]   3.00-4.00   sec  3.73 MBytes  31.3 Mbits/sec   11    103 KBytes
[  5]   4.00-5.00   sec  3.04 MBytes  25.5 Mbits/sec   10   96.2 KBytes
[  5]   5.00-6.00   sec  3.54 MBytes  29.7 Mbits/sec   11   89.1 KBytes
[  5]   6.00-7.00   sec  4.10 MBytes  34.4 Mbits/sec    3   83.4 KBytes
[  5]   7.00-8.00   sec  3.60 MBytes  30.2 Mbits/sec    0    113 KBytes
[  5]   8.00-9.00   sec  3.73 MBytes  31.3 Mbits/sec    3    103 KBytes
[  5]   9.00-10.00  sec  3.98 MBytes  33.4 Mbits/sec    6   97.6 KBytes
[  5]  10.00-11.00  sec  3.42 MBytes  28.7 Mbits/sec    9   91.9 KBytes
[  5]  11.00-12.00  sec  3.54 MBytes  29.7 Mbits/sec    6   90.5 KBytes
[  5]  12.00-13.00  sec  3.60 MBytes  30.2 Mbits/sec    3   82.0 KBytes
[  5]  13.00-14.00  sec  3.91 MBytes  32.8 Mbits/sec    0    113 KBytes
[  5]  14.00-15.00  sec  3.60 MBytes  30.2 Mbits/sec    2    106 KBytes
[  5]  15.00-16.00  sec  3.98 MBytes  33.4 Mbits/sec   12   93.3 KBytes
```

Figure 18. Server interface in Cubic downlink testing

The data collected will be divided into three intervals, which were previously discussed as 1 minute, 5 minutes, and 10 minutes. These data sets will be analysed using Microsoft Excel. By utilizing Excel, the collected data can be visually presented through charts, making it easier to compare the performance of different congestion control schemes. The overall performance of the schemes will be assessed through the comparison of the data obtained from the various CCA configurations.

4 RESULTS AND DISCUSSION

This section will encompass various aspects, including the throughput performance of five different schemes within a one-minute timeframe, as well as the latency, throughput, congestion window value, and retransmission rate. By carefully analyzing these factors, we can obtain more detailed and precise data to draw a conclusion. Additionally, this section will also incorporate tests conducted in alternative wireless environments, specifically focusing on WiFi, to showcase our results.

4.1 Throughput Behavior of Five Schemes

Once the JSON files are examined, the information will be incorporated into a line graph to contrast five distinct plans. Figure 20, the throughput behavior during a one-minute uplink test is displayed. The graph illustrates that all five schemes exhibit similar stability during the one-minute test period, with no noticeable differences. Furthermore, the graph demonstrates that there is little variation in the uplink test results among the five schemes.

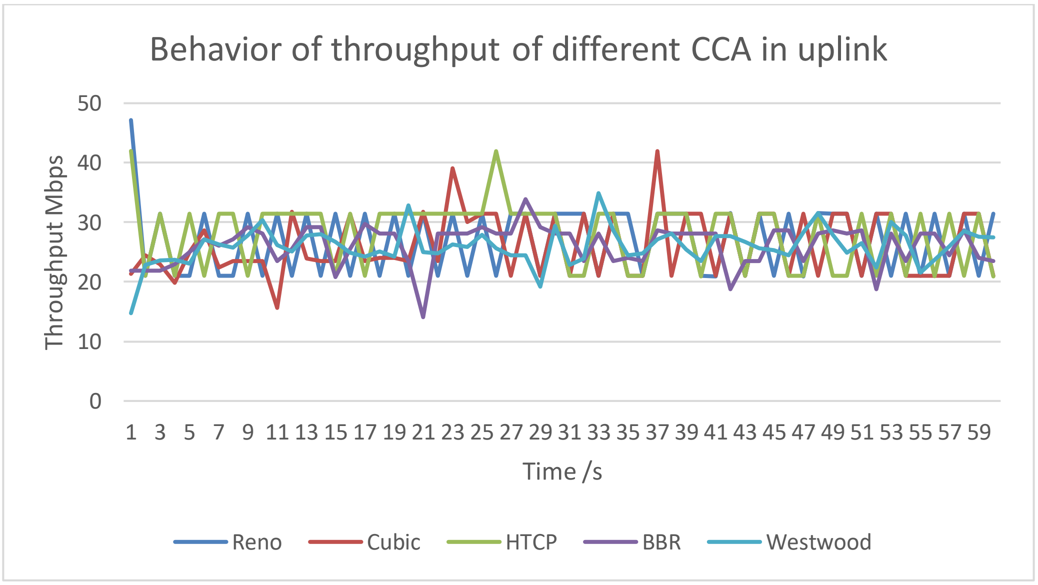


Figure 19. Throughput behaviour of 5 CCAs in uplink testing

Moving on to the next set of data, Figure 21 displays the comparison of the average throughput of the five different schemes tested in downlink. The results show that BBR outperformed the other schemes, with an average throughput of around 80 Mbps, while the other schemes fluctuated around 30 Mbps and remained stable in that range. However, it is worth noting that BBR's performance also exhibited significant fluctuations, ranging from 60 Mbps to 110 Mbps, indicating that although it achieved high performance, it also showed instability over the 5G network. This instability could be considered a disadvantage when seeking to achieve the best possible performance.

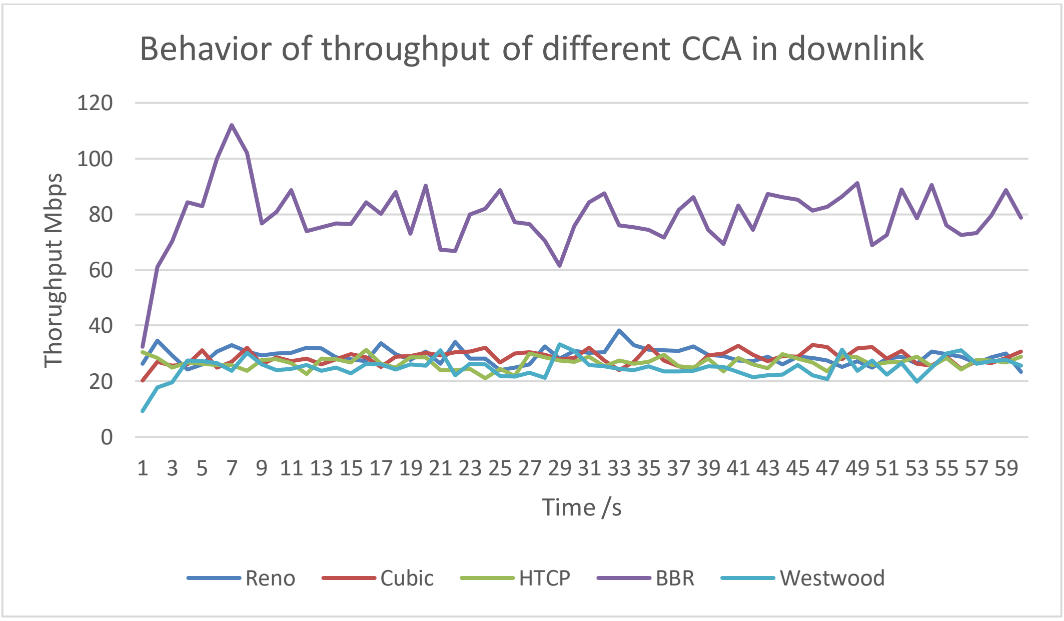


Figure 20. Throughput behaviour of 5 CCAs in downlink testing

Nonetheless, these findings provide valuable insights into the performance of different schemes in 5G networks and can aid in the optimization of network performance in the future.

4.2 Average Latency of Five Schemes

The objective of this section is to study how the performance of different TCP congestion control avoidance schemes is affected by latency. To achieve this, we extracted the RTT data from the JSON files and presented it in a column chart for better visualization of the CCA's performance. The column chart provides a clear summary of the RTT values for each of the tested schemes. The chart provides a summary of the data obtained from testing periods of 1 minute, 5 minutes, and 10 minutes, allowing for easy comparison between the different schemes. Additionally, we calculated the total average latency for all testing periods to provide an overall view of the performance of each CCA.

From a neutral standpoint, it can be observed that in the latency chart for one minute testing, the BBR still exhibits superior performance compared to other schemes, as shown in Figure 21. Conversely, the Westwood scheme shows the poorest performance in this regard.

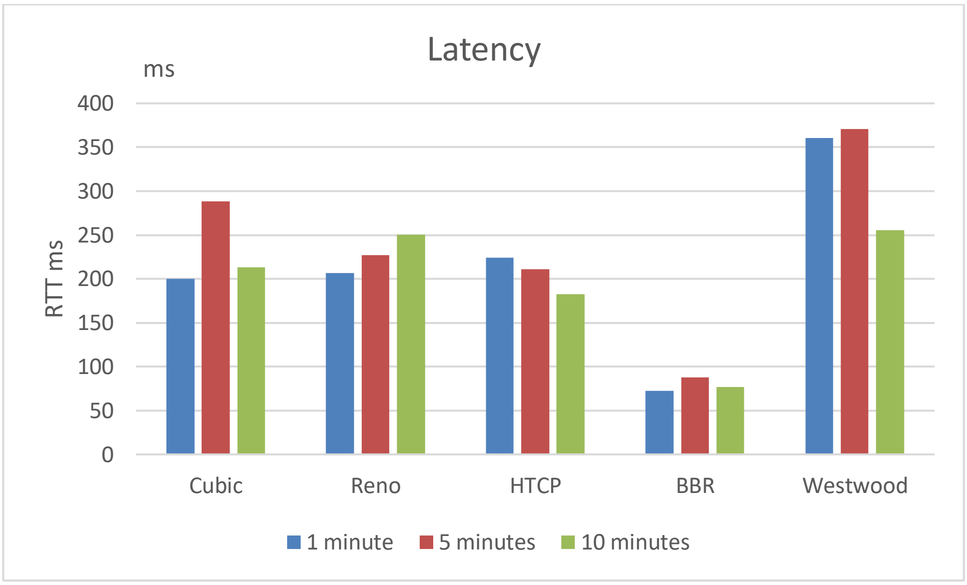


Figure 21. Average latency of 5 CCAs in 3 different intervals

In contrast, Cubic, Reno, and HTCP exhibit a comparable level of latency, suggesting that there is not much variation in their performance. This observation holds true for both the 5-minute and 10-minute testing periods, indicating the reliability and consistency of the results.

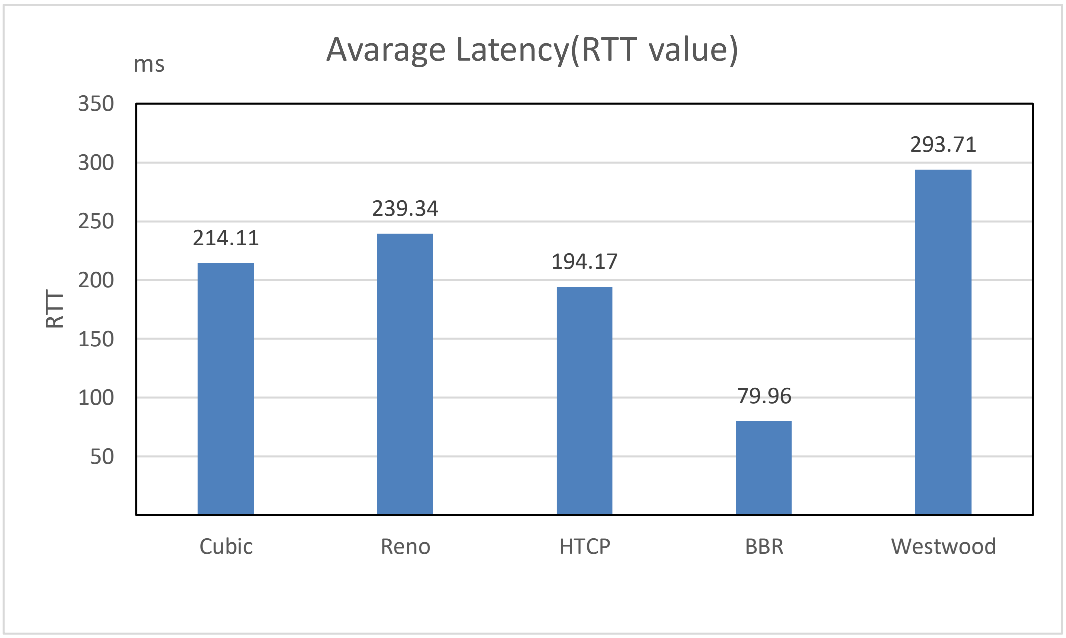


Figure 22. Average latency of 5 CCAs in all times

The findings reveal that BBR outperforms other TCP congestion avoidance schemes in terms of latency. Additionally, BRR also shows good results, possibly due to its ability to handle burst traffic. On the other hand, the results also demonstrate that the Westwood algorithm performs poorly in 5G networks, as indicated by its high latency.

4.3 Average Throughput of Five Schemes

The goal of this section is to assess the performance of five different schemes in 5G link by comparing their throughput. To achieve this, we gathered the necessary data from a JSON file named "bits_per_second" and used it to create a chart that provides a clear visualization of the results. To conduct an effective comparison, we focused on two channels uplink and downlink.

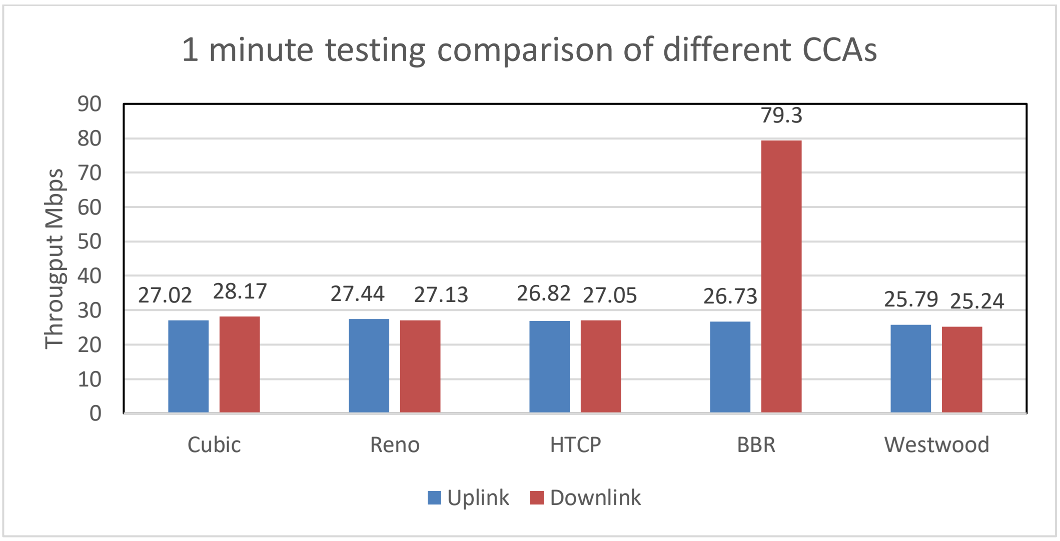


Figure 23. Average throughput of 5 CCAs in 1 minute testing at both way

In this section, the aim is to compare the performance of five different TCP congestion control algorithms schemes in the 5G link, by examining their throughput. The data collected from the JSON files contain the bit-per-second variable, which is used to create a chart for better visualization and comparison. The chart covers two channels, namely the downlink and uplink channels. Figure 24 shows that there is no significant difference in the throughput of the five schemes in both uplink and downlink channels, except for the BBR scheme which outperforms others in terms of throughput. The bitrate of the BBR scheme is significantly higher than the others.

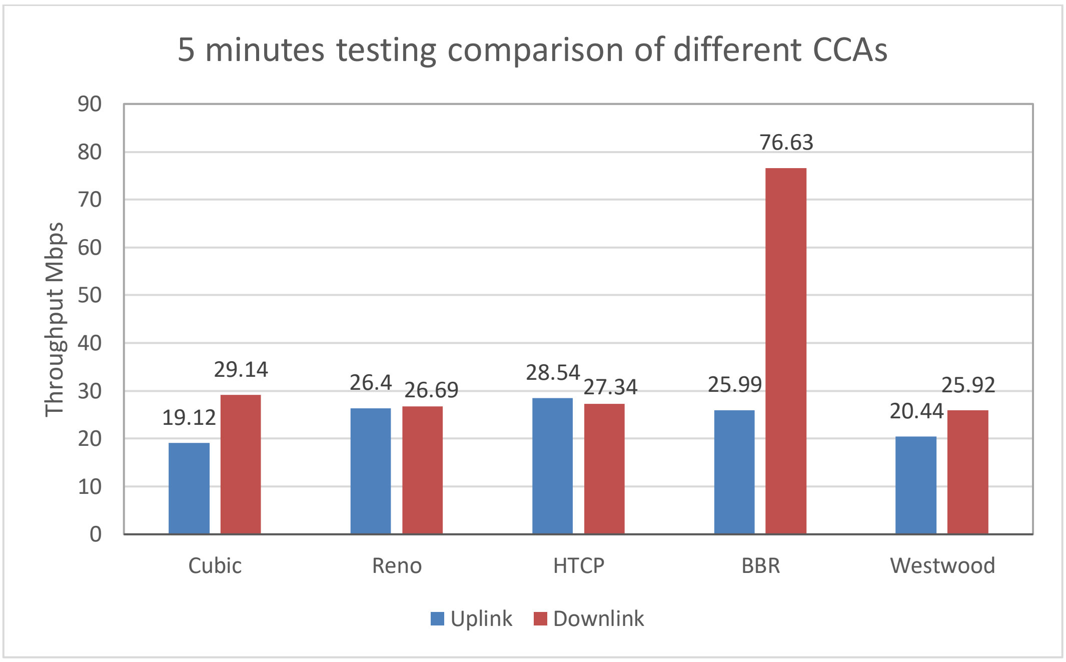


Figure 24. Average throughput of 5 CCAs in 5-minute testing at both way

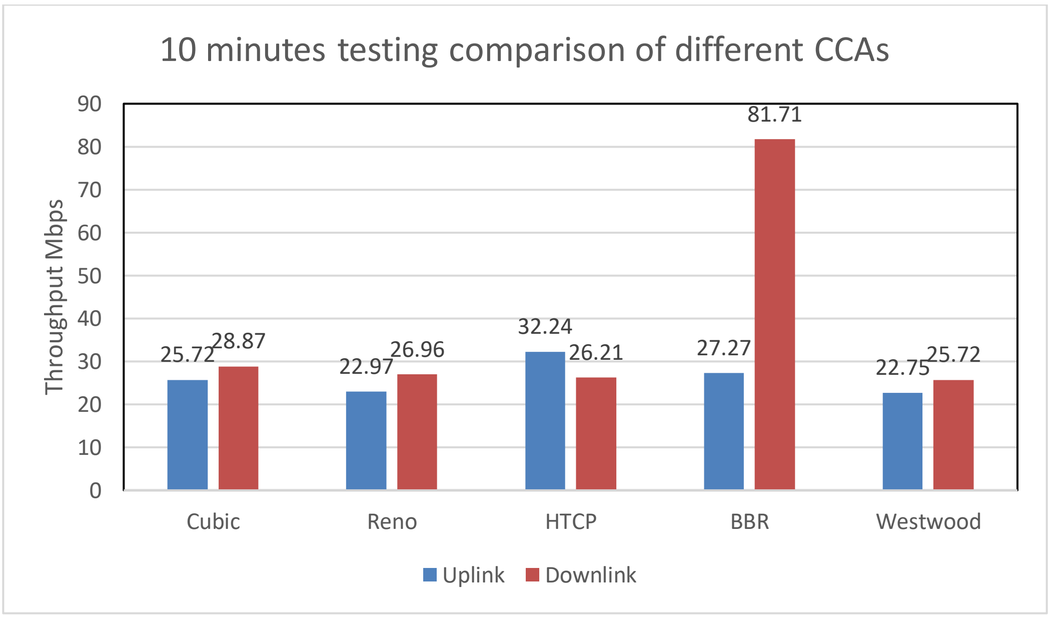


Figure 25. Average throughput of 5 CCAs in 10-minute testing at both way

Based on the data presented in figures 25 and 26, BBR still shows the best performance in the downlink channel, while HTCP has improved in the uplink channel compared to the other schemes. On the other hand, the worst performer in both channels is Westwood, which demonstrates significantly slower performance compared to the other schemes.

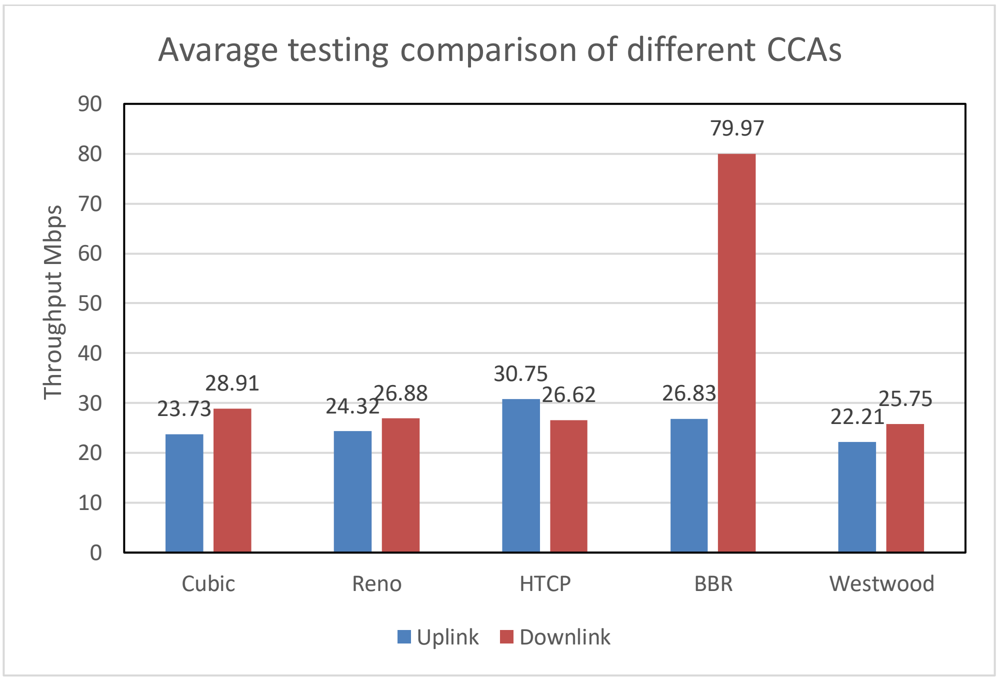


Figure 26. Average throughput of 5 CCAs in total testing at both way

In the concluding analysis of throughput, the results are further refined and demonstrate consistency with the previous testing. BBR continues to exhibit dominant performance in this aspect, with HTCP also exhibiting a promising result. Conversely, Westwood emerges as the weakest performer in this comparison. It is possible that Westwood's design caters more towards a fairness network, rather than prioritizing performance in high-speed, 5G networks. Overall, the data provides a reliable and accurate comparison between the different TCP congestion avoidance schemes, allowing for informed decisions to be made in selecting the appropriate scheme for specific network needs.

4.4 Congestion Window Behavior of Five Schemes

The purpose of this section is to examine the behavior of *cwnd* and compare it between the five different schemes. To collect the *cwnd* data, we utilized the max_snd_cwnd parameter from the JSON files that we collected during the testing. Regrettably, it was possible to gather only *cwnd* values from the uplink direction, as data collection was limited to the client side using the iperf3 tool. Consequently, the collected *cwnd* values solely reflect the perspective from the client side, and no data from the server side was obtained.

Our goal is to analyze and visualize the *cwnd* data to make meaningful comparisons between the schemes.

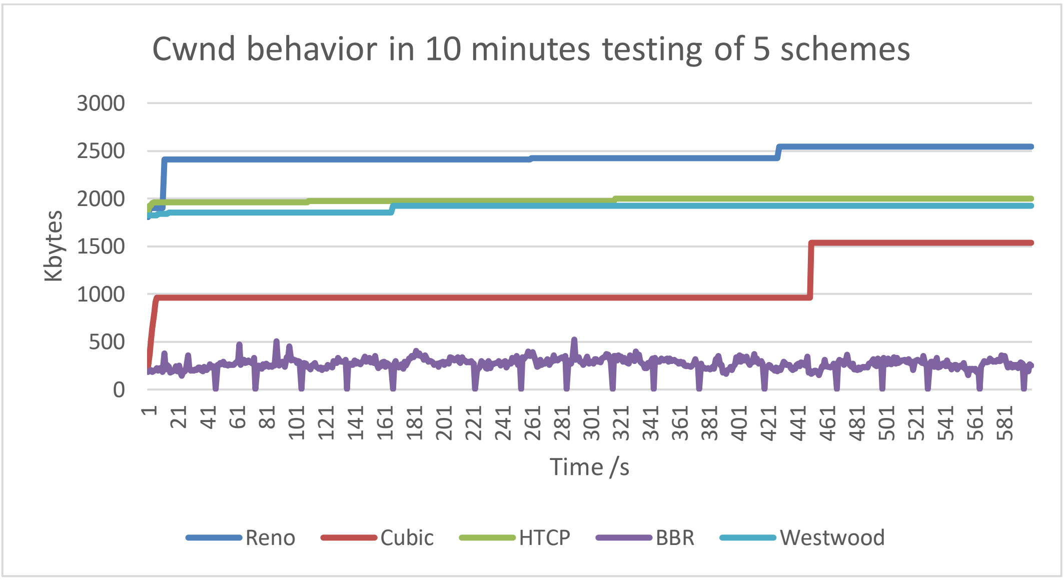


Figure 27. Cwnd value behaviour of 5 CCA for 10 minutes

From the analysis of Figure 28, it is observed that the behaviour of the congestion window value in the BBR scheme is quite interesting. The *cwnd* value is found to be unstable and fluctuates between 200 Kbytes to 400 Kbytes, with a sinusoidal-like pattern in the graph. At times, the value even drops to 0, which could be a phase where congestion is being avoided by the schemes.

In contrast to BBR, the *cwnd* values of other TCP schemes appear, according to Figure 28, to be more stable, with higher average values than BBR. This suggests that the other schemes may prioritize stability over throughput optimization, resulting in more consistent and higher *cwnd* values.

It is worth noting that BBR was specifically designed to optimize throughput and reduce latency for high-speed. As such, BBR dynamically adjusts its *cwnd* size to maintain a desired sending rate and minimize packet delay. This could explain the fluctuations in *cwnd* values observed in Figure 28, as BBR adjusts its *cwnd* in response to changing network conditions.

However, the fluctuation of *cwnd* values in BBR does not necessarily mean that it is performing poorly. In fact, as shown in previous sections, BBR demonstrated outstanding performance in terms of throughput and latency. The fluctuations in *cwnd* values may be indicative of BBR's ability to dynamically adapt to changing network conditions, which is a key feature of its design.

In comparison, the other TCP schemes may have a more static approach to congestion control, resulting in more consistent *cwnd* values. However, this may come at the expense of throughput optimization and latency reduction. Overall, the choice of TCP congestion control scheme should be based on the specific network requirements and priorities, whether it is stability, throughput, or latency optimization.

4.5 Retransmission Rate of Five Schemes

The retransmission rate was obtained by extracting the "retransmits" parameter from the JSON file in uplink testing and subsequently converted to an Excel format for additional analysis. The average retransmission rate can be found in Table 2.

Table 2. Retransmission rate of five schemes

	Retransmission rate
Reno	4.63
Cubic	3.64
HTCP	2.64
BBR	684.89
Westwood	4

After analyzing Table 2, it appears that BBR has a higher retransmission rate compared to the other schemes. This may indicate that BBR is more sensitive to network congestion and is more likely to experience packet losses, which can negatively impact the performance of a TCP connection. Retransmission of lost packets leads to increased latency and reduced throughput. Therefore, it is important for a congestion control algorithm to balance between optimizing throughput and minimizing packet loss. While BBR prioritizes throughput, it may result in higher retransmission rates. However, it is interesting to note that the latency of BBR testing is still low, indicating that it is compatible with 5G networks.

4.6 Other Wireless Communication Testing of Five Schemes.

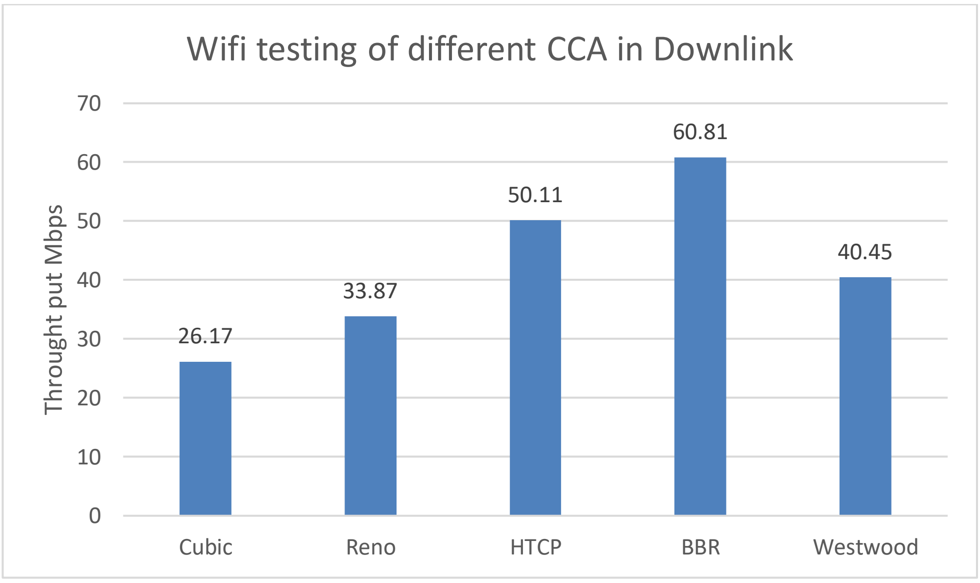


Figure 28. WIFI testing throughput in downlink.

To validate the superior performance of BBR, additional tests were conducted in a different wireless communication environment, specifically using Wi-Fi. In this scenario, a connection was established between the edge server and the same RPI by utilizing a TPLINK Wi-Fi module. The module was connected to the network, which is identical to the edge server, through the WAN interface. Subsequently, the RPI was connected to the module via Wi-Fi.

The results of this test also showed that BBR still outperformed the other congestion control schemes, although it was not as dominant as in the 5G network testing. It is worth noting that different wireless communication environments can have varying characteristics and challenges, and congestion control algorithms may behave differently depending on the network conditions. Despite this, the fact that BBR consistently showed better results in both 5G and WIFI testing suggests that it is a robust and effective congestion control algorithm.

Interestingly, the results also showed that Westwood, which performed poorly in the 5G network testing, demonstrated good results in the WIFI communication environment. This highlights the importance of testing congestion control algorithms in different network conditions to better understand their behavior and performance.

In addition, Figure 29 further supports the better performance of BBR in wireless communication environments. The graph demonstrates that BBR has higher throughput values compared to other congestion control algorithms in a WIFI environment, while the other algorithms exhibit lower throughput values and greater variability in their performance.

5 CONCLUSIONS

This thesis has analyzed five different TCP congestion control algorithms, namely Reno, Cubic, HTCP, BBR, and Westwood over a 5G (NSA/SA) TDD link of 50MHz band. The tests were conducted in both the downlink and uplink cases, with data collected in three different intervals and stored in the JSON format for a subsequent analysis in Excel.

Upon analyzing the results, it can be concluded that BBR outperforms the other algorithms in terms of latency and throughput in downlink. However, BBR does show a higher retransmission rate and more instability in *cwnd* in 5G networks. In contrast, Westwood performed poorly in all aspects covered in this paper but showed good performance in the downlink case. These results suggest that BBR may be better suited for server-based applications, such as edge servers, while HTCP may be more appropriate for client-based applications. Westwood may be a suitable option for achieving network fairness. Finally, the Cubic and Reno algorithms demonstrated results in line with expectations.

In conclusion, for 5G industries and applications, BBR is recommended for server-based use cases such as edge servers, while HTCP is preferable for uplink cases. However, it is important to consider the trade-offs between performance and other factors such as retransmission rates and network stability.

6 REFERENCES

Allman, V. P. (2009, September). *rfc5681*. Retrieved from TCP Congestion Control: https://www.rfc-editor.org/rfc/rfc5681#section-3

Armitage, G., Stewart, L., Welzl, M., & John, H. (2008). An independent H-TCP implementation under FreeBSD 7.0. *ACM SIGCOMM Computer Communication Review*, ACM (Association for Computing Machinery).

Cardwell, N., Cheng, Y., Gunn, C. S., Yeganeh, S. H., & Jacobson, V. (2017). BBR: congestion-based congestion control. *Communications of the ACM*, 58-66.

Cerf, V. G., & Kahn, R. E. (1974). A Protocol for Packet Network Intercommunication. *IEEE Transactions on Communications*, 637-648.

Floyd, S. (2001). A report on recent developments in TCP congestion control. *IEEE Communications Magazine*.

Ha, S., Rhee, I., & Lisong, X. (2008b). CUBIC: A New TCP-Friendly High-Speed TCP Variant.

Ha, S., Rhee, I., & Lixia, X. (2008a). *CUBIC.* New York: ACM SIGOPS Operating Systems Review.

Jason, M. (2021, Aug 20). *jasonmurray*. Retrieved from TCP Congestion Control Algorithms in Linux: https://jasonmurray.org/posts/2021/tcpcca/

Kozierok, C. M. (2005). *The TCP/IP Guide: A Comprehensive, Illustrated Internet Protocols Reference.* San Francisco: No Starch Press.

Loshin, P. (2003). *TCP/IP Clearly Explained.* San Francisco: Morgan Kaufmann Publishers.

Mascolo, S., Casetti, C., Gerla, M., Sanadidi, M. Y., & Wang, R. (2001). TCP Westwood: Bandwidth Estimation for Enhanced Transport over Wireless Links. *ACM Mobicom 2001.* Rome.

Pasi, S., & Kuznetsov, A. (2002). *USENIX 2002 Annual Technical Conference, Freenix Track - Pape*. Retrieved from Congestion Control in Linux TCP: https://www.usenix.org/legacy/event/usenix02/tech/freenix/full_papers/sarolahti/sarolahti_html/

Rappaport, T. S. (2013). Millimeter wave mobile communications for 5G cellular: It will work! *IEEE Access*, 335-349.

Ross, Kurose, J., & Keith. (2012). *Computer Networking: A Top-Down Approach.* Pearson.

Sadeeq, M., Aljahmany, R., Zebari, A. A., & Rizgar. (2020). *Evolution of Mobile Wireless Communication to 5G Revolution.* ResearchGate (self-published).

Sarolahti, P., Koponen, T., Kompella, K., & Ylianttila, M. (2002). Design and Implementation of OpenBSD's PF Packet Filter. *USENIX Association.* Monterey, CA.

Stevens, W. R. (1994). *TCP/IP Illustrated, Volume 1: The Protocols.* Addison-Wesley.

Sun, T. (2020). Research on 5G Wireless Network Technology Based on Millimeter Wave. IEEE.

Waveshare. (2023). Retrieved from SIM8200EA-M2 5G HAT: https://www.waveshare.com/wiki/SIM8200EA-M2_5G_HAT#Overview

Yi, C. (2022). Delivery Rate Estimation.