****

**MASTER THESIS**

The Influence of selected TCP parameters on transmission

performance

**Nathan Djesse Mwamba**

**Student identification number <nd304723>**

**Program: Informatics**

**Specialization:** **Informatics**

**SUPERVISOR**

**Dr. Agnieszka Piotrowska**

**DEPARTMENT**

**Faculty of Automatic Control, Electronics and Computer Science**

**GLIWICE 2023**

# Abstract:

This research delves into the intricate relationship between specific TCP parameters and their effects on transmission performance. Through a series of controlled experiments involving manipulation of segment size and initial congestion window (cwnd), the study delves into the behavior of TCP Cubic and TCP New Reno, two widely used TCP congestion control algorithms.

The experiments encompass comprehensive evaluations of critical performance metrics, including fairness, throughput, delivery ratio, and drop ratio. By systematically altering the parameters and meticulously observing the outcomes, the study offers valuable insights into how these factors interplay to shape the overall performance of data transmission.

The findings of this investigation contribute to a deeper comprehension of the multifaceted dynamics within TCP algorithms and their interactions with network conditions. This understanding is pivotal for network administrators, engineers, and researchers aiming to optimize transmission performance without favoring any specific algorithm. The insights presented in this study pave the way for informed decisions regarding TCP parameter configurations, ultimately enhancing the reliability and efficiency of data transmission across diverse network environments.

* **Keywords:**

TCP Cubic, TCP New Reno, Segment Size, Initial Congestion Window.

# Streszczenie:

Niniejsza praca miała na celu analizę wpływu wybranych parametrów konfiguracyjnych protokołu TCP na efektywność transmisji. Aby umożliwić ocenę wpływu wybranych parametrów, zrealizowano zaproponowane scenariusze badań symulacyjnych. Zmieniając rozmiar segmentu i początkową wielkość okna przeciążenia badano zachowanie protokołów TCP Cubic i TCP New Reno, dwóch szeroko stosowanych algorytmów kontroli przeciążenia TCP.

Eksperymenty objęły ocenę wybranych metryk wydajności, tj. sprawiedliwości, przepustowości, współczynnika dostarczania i współczynnika odrzuconych pakietów. Zmieniając wartości wybranych parametrów konfiguracyjnych i zbierając wspomniane wartości metrykprzedstawiono wpływ tych parametrów na efektywność transmisji danych.

Przedstawione wyniki umożliwiajągłębsze zrozumienie wieloaspektowej dynamiki protokołów TCP. Zrozumienie to ma kluczowe znaczenie dla administratorów sieci, inżynierów i badaczy dążących do optymalizacji wydajności transmisji bez faworyzowania żadnego konkretnego algorytmu. Spostrzeżenia przedstawione w tej pracy torują drogę do świadomych decyzji dotyczących konfiguracji parametrów TCP, ostatecznie zwiększając niezawodność i wydajność transmisji danych w różnych środowiskach sieciowych.

* **Słowa kluczowe:**

TCP Cubic, TCP New Reno, Segment Size, Initial Congestion Window

# Introduction

In today's dynamic landscape of digital communication networks, the efficient transmission of data has become a cornerstone in upholding uninterrupted connectivity. This research delves deeply into a critical aspect: the influence of selected TCP (Transmission Control Protocol) parameters on the performance of data transmission. This domain is of paramount importance, as it directly impacts the reliability and effectiveness of data delivery, a fundamental concern in today's digital world.

TCP, being a foundational protocol governing data transmission, operates under a defined set of parameters that guide its behavior. Surprisingly, the intricate interplay between these parameters and their direct influence on transmission performance has received relatively limited attention compared to other facets of network optimization. This, in turn, has led to a notable gap in our understanding, one that this study aims to address.

The core objective of this thesis is to unravel the complexities that underlie the relationship between specific TCP parameters and their impact on transmission performance. By conducting systematic empirical evaluations utilizing two prominent TCP variants—TCP New Reno and TCP Cubic—the overarching goal is to provide profound insights into how parameter choices intricately shape network efficiency. This study seeks to comprehensively evaluate the effects of window size on transmission performance, delve into the influence of congestion window size, and dissect the impact of maximum segment size on overall transmission performance.

By meticulously conducting these evaluations, the study aims to propose optimal parameter values that can significantly enhance transmission performance. The ultimate outcome of this research is to furnish network practitioners with actionable insights for making informed decisions when configuring TCP parameters, thereby contributing to the achievement of enhanced data transmission efficiency in modern communication networks.

The subsequent chapters are organized to facilitate a coherent exploration of this topic. The introduction establishes the problem's significance within the evolving network landscape, outlining the objective to comprehensively assess the effects of critical TCP parameters. The scope encompasses experimental evaluation using TCP New Reno and TCP Cubic, with a focus on segment size and initial congestion window settings.

Chapter 2 delves into TCP and its variants, analyzing their parameter choices and resultant behaviors. The author's contribution lies in dissecting and comparing these variants, elucidating their historical evolution. Chapter 3 involves rigorous experimental evaluation, employing TCP New Reno and TCP Cubic to explore the influence of parameter modifications on transmission performance. The author's role is to meticulously design and execute experiments and analyze results to uncover the impact on performance metrics. The final chapter, Chapter 4, encapsulates the research journey, presenting synthesis, implications, and future directions. The author's contribution lies in distilling findings and highlighting their broader significance. This study's importance rests on its provision of insights for informed TCP parameter configuration decisions, equipping network practitioners with tools for enhancing data transmission efficiency.

# TCP and analysis of different TCP variants

## Introduction to TCP

Transmission Control Protocol (TCP) is a foundational protocol of the Internet Protocol Suite (TCP/IP). It operates at the transport layer and provides reliable, connection-oriented communication between devices over an IP network. TCP ensures data integrity and sequencing by establishing a virtual circuit between the sender and receiver, which involves a three-way handshake for connection setup [1].

TCP works by breaking data into smaller segments, encapsulating them in TCP headers, and transmitting them to the destination. It provides reliable delivery by using acknowledgments (ACK) and sequence numbers to ensure data is received and reconstructed in the correct order. If any segments are lost or damaged during transmission, TCP's mechanisms trigger retransmission to guarantee data integrity.

### Structure of the TCP Header

The TCP header contains crucial information, including source and destination port numbers, sequence and acknowledgment numbers, control flags (such as SYN, ACK, FIN), and window size. The window size indicates the amount of data that can be sent before receiving an acknowledgment.

A diagram of a number

Description automatically generated with medium confidence

Figure 2.1 *Structure of the TCP Header* [1]

Figure 2.2 *Sliding Window*Figure 2.3 *Structure of the TCP Header* [1]

The TCP packet format consists of various key fields:

* Source Port and Destination Port: indicate the application's sending and receiving port numbers.
* Sequence Number and Acknowledgment Number: represent the byte positions in the data stream and the next expected byte, respectively.
* Data Offset, Reserved, and Control Bits: control aspects of the TCP communication, including flags for SYN, FIN, and urgent data.
* Window: indicates the current capacity of the receiving host for data acceptance.
* Checksum: provides error checking for the entire TCP segment.
* Urgent Pointer: Identifies segments with urgent data when the URG flag is set.
* Options: contains additional information like maximum segment size, timestamps, and selective acknowledgment.
* Padding: ensures the TCP header ends on a 32-bit boundary.

The TCP packet format is designed to enable efficient and reliable data transmission over IP networks, incorporating robust error checking mechanisms [4].

### Flow Control Mechanisms in TCP

Flow control mechanisms are crucial in Transmission Control Protocol (TCP) to prevent a fast sender from overwhelming a slow receiver, leading to packet loss, congestion, and performance degradation. TCP flow control mechanisms adjust the rate of data transmission based on the receiver's ability to receive and process data. In this section, we will provide an overview of the flow control mechanisms used in TCP and explain some of the popular algorithms used to implement these mechanisms [2].

Flow control is implemented in different ways, depending on how the sender and receiver handle messages and track data frames. One of the basic approaches to flow control is sliding window, with Sliding Window Protocol. This protocol allows for the efficient management of data flow between a sender and a receiver. It permits the sender to transmit multiple packets without waiting for each acknowledgment, based on the receiver's buffer capacity. The sliding window protocol dynamically adjusts the window size to control the flow of data, ensuring that the sender does not overwhelm the receiver with an excessive amount of data. By maintaining a balance between data transmission and processing capacity, the sliding window protocol helps prevent data loss and ensures a reliable and efficient data transfer process in TCP [3].

A graph with arrows and text

Description automatically generated

Figure 2.4 *Sliding Window*

### Congestion window size

The window size is a TCP parameter that specifies the maximum amount of data that can be sent before an acknowledgement is received. A larger window size allows for more data to be sent in a single transmission, which can improve performance. However, a larger window size also increases the risk of congestion and packet loss [1].

TCP uses “windowing” which means that a sender will send one or more data segments and the receiver will acknowledge one or all segments. When we start a TCP connection, the hosts will use a receive buffer where we temporarily store data before the application can process it [4].

When the receiver sends an acknowledgment, it will tell the sender how much data it can transmit before the receiver sends an acknowledgment. We call this the window size. Basically, the window size indicates the size of the receive buffer [5].

Typically, the TCP connection will start with a small window size and every time there is a successful acknowledgement, the window size will increase. Here’s an example:

A graph paper with black text

Description automatically generated

Figure 2.5 *segment 1*

Figure 2.6 *Segment 2*Figure 2.7 *segment 1*

Figure 2.8 *Segment 2*

Figure 2.9 *segment4*Figure 2.10 *Segment 2*Figure 2.11 *segment 1*

Figure 2.12 *Segment 2*Figure 2.13 *segment 1*

The Figure 2.8 show two hosts, the host on the left side will send one segment and the host on the right side will send an acknowledgment in return. Since the acknowledgement was successful, the windows size will increase [5]

A graph with black text

Description automatically generated

Figure 2.14 *Segment 2*

The host on the left side is now sending two segments and the host on the right side will return a single acknowledgment. Figure 2.4. Everything is working fine so the window size will increase even further [5].

A graph with arrows and text

Description automatically generated

Figure 2.15 *segment4*

The host is now sending four segments and the host on the right side responds with a single acknowledgment.

In the example Figure 2.16, the window size keeps increasing as long as the receiver sends acknowledgments for all our segments or when the window size hits a certain maximum limit. When the receiver doesn’t send an acknowledgment within a certain time period (called the round-trip time) then the window size will be reduced [5].

When an interface has congestion then it’s possible that IP packets are dropped. To deal with this, TCP has a number of algorithms that deal with congestion control. One of them is called *slow start*.

### Receive Window (rwnd):

A screenshot of a video game

Description automatically generatedThe receive window is used by the receiver to specify the amount of data it can accept at a given time. It helps in controlling the flow of data from the sender, preventing the receiver from being overwhelmed by an excessive amount of incoming data [6].

Figure 2.16 *Receive Window* [6]

Upon establishing a TCP session, the Window size field within the TCP header is utilized to convey the receiving buffer's capacity, dictating the quantity of data that can be both received and processed. Each endpoint retains a local receive window (RWND) to signify the utmost data capacity the receiver can handle for buffering and processing. This RWND value is incorporated into the TCP header. Utilizing RWND as a reference, the sender adjusts the Sliding window size, permitting the transmission of TCP segments to the peer, up to the size specified in the window, without the necessity to wait for an acknowledgment after each transmission [6].

In data transmission, the sender chooses the minimum value between the congestion window and the receiver's window to determine the amount of data that can be sent without overwhelming the network. This approach helps maintain an efficient and stable flow of data, preventing congestion and ensuring reliable communication between the sender and the receiver [6].

## TCP Connection

TCP allows for transmission of information in both directions. This means that computer systems that communicate over TCP can send and receive data at the same time, similar to a telephone conversation. The protocol uses segments (packets) as the basic units of data transmission. In addition to the payload, segments can also contain control information. The TCP software in the network protocol stack of the operating system is responsible for establishing and terminating the end-to-end connections as well as transferring data [7].

The TCP software is controlled by various network applications, such as web browsers or servers, via specific interfaces. Each connection must always be identified by two clearly defined endpoints (client and server). It doesn’t matter which side assumes the client role and which assumes the server role. All that matters is that the TCP software is provided with a unique, ordered pair consisting of IP address and port (also referred to as "2-tuple" or "socket") for each endpoint [7].

The three-way handshake: how a TCP connection is established in detail.

Prerequisites for establishing a valid TCP connection: both endpoints must already have a unique IP address (IPv4 or IPv6) and have assigned and enabled the desired port for data transfer. The IP address serves as an identifier, whereas the port allows the operating system to assign connections to the specific client and server applications [1].

Process for establishing a connection with the TCP protocol is as follows:

First, the requesting client sends the server a SYN packet or segment (SYN stands for synchronize) with a unique, random number. This number ensures full transmission in the correct order (without duplicates).

If the server has received the segment, it agrees to the connection by returning a SYN-ACK packet (ACK stands for acknowledgment) including the client's sequence number plus 1. It also transmits its own sequence number to the client.

Finally, the client acknowledges the receipt of the SYN-ACK segment by sending its own ACK packet, which in this case contains the server's sequence number plus 1. At the same time, the client can already begin transferring data to the server.

Timeline

Description automatically generated

Figure2.17 *TCP Connection Establishment (Three-way handshake)*

Data transfer: Following the establishment of a connection, data is transmitted in segments, each carrying a sequence number and an acknowledgment number. The sequence number denotes the position of the segment in the data stream, while the acknowledgment number signifies the next anticipated segment [8].

Sending and receiving management: TCP employs a sliding window mechanism to effectively handle the sending and receiving of segments. This mechanism entails a range of sequence numbers that are eligible for transmission or reception [8].

Connection termination: Upon the completion of data transfer, the connection is closed through a process known as the "four-way handshake." This involves a series of steps: the client sends a FIN (finish) request to signal the end of data transmission, the server responds with an ACK to confirm, then the server issues a FIN request to indicate its conclusion of data transmission, and finally, the client responds with an ACK to confirm the closure of the connection [8].

**Text

Description automatically generated with medium confidence**

Figure 2.28 *TCP Connection termination (TCP Teardown*)

## TCP Congestion Control

A diagram of a flowchart

Description automatically generatedCongestion control is vital for maintaining dependable data transmission across computer networks in the Transmission Control Protocol (TCP). It employs key strategies to handle and respond to network congestion efficiently. These strategies encompass three main phases [9]:

Figure 2.35 *TCP Congestion Control* [9]

* **Slow Start**

This phase starts cautiously with a limited initial congestion window (CWND), gradually expanding it exponentially as acknowledgments are received. The aim is to explore network capacity without overwhelming it, finding an optimal data transmission rate for the current conditions.

* **Congestion Avoidance**

Following slow start, congestion avoidance maintains a stable transmission rate by increasing CWND linearly rather than exponentially. It monitors the network for congestion signs and adjusts sending rates to strike a balance between efficiency and congestion avoidance [9].

* **Congestion Detection**

This phase involves identifying congestion signals, including packet loss, explicit congestion notifications from routers, and monitoring round-trip times. TCP adapts by reducing CWND to alleviate congestion, with specific responses varying based on the congestion control algorithm used.

These phases collectively enable TCP to adapt to changing network conditions, maximize throughput, and minimize congestion risks, ensuring effective data transmission. Different TCP versions, such as Reno, New Reno, Vegas, and Cubic, have been developed to enhance congestion control strategies, each employing distinct approaches to adjust sender transmission rates based on network conditions and packet loss feedback, below we explain those algorithms and how there are works [9].

### TCP Tahoe

TCP Tahoe, developed for the Transmission Control Protocol (TCP) in computer networks, is among the earliest congestion control algorithms. Created in 1988 by Van Jacobson, a computer scientist at Lawrence Berkeley National Laboratory, it derived its name from the region of Lake Tahoe in California, where Jacobson worked on its development. TCP Tahoe's central objective was to establish a dependable and effective method for managing congestion in computer networks. Its functions included the detection of network congestion and the application of measures to curtail data transmission rates, thereby preventing packet loss [10].

* **Features and Benefits of TCP Tahoe**

TCP Tahoe incorporates various functions that contribute to effective network congestion control. For instance, it employs a slow start feature, enabling new connections to commence with a conservative sending rate that gradually increases as successful data transmission occurs.

Moreover, it utilizes the fast retransmit capability, which identifies lost packets through the observation of duplicate acknowledgments from the receiver. This allows for prompt retransmission of the lost packets, thereby preventing timeouts [10].

These attributes of TCP Tahoe ensure dependable data transmission and optimal utilization of the available network bandwidth. Its notable advantages include its user-friendly design, strong reliability, efficiency, and adaptability. Its straightforward implementation makes it readily accessible for most systems without significant modifications or reliance on other protocols [10].

**Advantages and Disadvantages of Using TCP Tahoe**

A notable benefit of employing TCP Tahoe is its rapid detection of network congestion and subsequent adjustments, ensuring minimal packet loss and preserving performance quality. It also maintains equitable treatment among various connections sharing a common link. Nonetheless, a drawback arises when confronted with high congestion levels, particularly in scenarios where multiple connections vie for network resources simultaneously. This drawback may lead to diminished throughput and increased latency due to a rise in dropped packets caused by insufficient resources. Comprehending TCP Tahoe's definition, origins, characteristics, advantages, and limitations aids in recognizing its continued relevance in contemporary networks and provides insights into its comparison with other congestion control algorithms such as TCP Reno and New Reno [10].

### TCP Reno

TCP Reno builds upon the foundational concepts of TCP Tahoe, retaining features such as slow starts and the coarse-grained re-transmit timer. However, it introduces enhanced capabilities, facilitating earlier detection of lost packets and preventing unnecessary pipeline depletion after packet losses. Reno operates based on the principle that the receipt of immediate acknowledgments is crucial, especially when handling duplicate acknowledgments. Duplicate acknowledgments suggest the probable loss of a segment, prompting the use of the 'Fast Re-Transmit' algorithm.

Upon receiving three duplicate ACKs, TCP Reno interprets this as a signal of segment loss, triggering an immediate re-transmission of the segment and initiating Fast Recovery. It sets the slow start threshold (SSthresh) to half the current window size and aligns the congestion window (CWND) accordingly. With each duplicate ACK received, CWND is incremented, allowing the transmission of a new segment if the increased CWND exceeds the data volume in the pipeline. Reno ensures that half a window of data is acknowledged before the transmission of a new segment following the loss of one segment. After re-transmission, a fresh acknowledgment is awaited, leading to the reduction of CWND to SSthresh. This process prevents the pipeline from emptying, facilitating a controlled flow without depleting network resources, followed by the continuation of the congestion avoidance phase [11].

* **Problems:**

Reno Reno's performance excels in scenarios involving minimal packet losses, but it demonstrates limitations when faced with multiple packet losses within a single window. Under conditions of substantial packet loss, Reno's efficacy diminishes, resembling the performance of Tahoe. The algorithm's capability to identify only a single packet loss poses a significant challenge. In cases of multiple packet losses, information about the subsequent packet losses is delayed until the receipt of duplicate acknowledgments after a round-trip time (RTT) [12].

Moreover, Reno might experience a reduction in the congestion window (CWND) twice for packet losses occurring within a single window. For instance, if packets 1 and 2 are lost from a sequence of packets 1 to 9, the reduction in CWND occurs first when packet 1 is re-transmitted, followed by another reduction when packet 2 is re-transmitted. This results in a reduction of the window size twice for packet losses within the same window.

Additionally, if the window size is small when a loss occurs, Reno may not receive sufficient duplicate acknowledgments for a fast re-transmit, leading to a coarse-grained timeout. Consequently, Reno's ability to effectively detect multiple packet losses is limited under such circumstances [12]

### TCP Vegas

Vegas, an altered version of Reno, is a TCP implementation that prioritizes proactive strategies for managing congestion, recognizing their superior efficiency compared to reactive measures. It addresses the challenge of coarse-grained timeouts by proposing an algorithm that efficiently schedules timeout checks. Moreover, it mitigates the reliance on a sufficient number of duplicate acknowledgments for detecting packet loss. Additionally, Vegas introduces a modified slow start algorithm that prevents network congestion.

Unlike its predecessors, Vegas does not solely rely on packet loss as an indicator of congestion. Instead, it detects congestion before packet losses occur, significantly enhancing its congestion control capabilities. Despite these enhancements, Vegas retains the fundamental mechanisms of Reno and Tahoe, with packet loss still being detectable through the coarse-grained timeout if the other mechanisms fail. [13].

The three major changes induced by Vegas are:

* **New Re-Transmission Mechanism:**

Vegas builds upon Reno's re-transmission mechanism by introducing a sophisticated tracking system for segment transmission and real-time calculations of the estimated round-trip time (RTT) based on acknowledgment timings. Upon the receipt of a duplicate acknowledgment, Vegas swiftly evaluates whether the current time minus the segment's transmission time exceeds the estimated RTT. If this condition is met, Vegas promptly re-transmits the segment, bypassing the need for three duplicate acknowledgments or a coarse timeout. This approach effectively resolves Reno's challenge of detecting lost packets within a small window when an insufficient number of duplicate ACKs is received.

To capture any previously lost segments prior to re-transmission, Vegas monitors the timeout values when a non-duplicate acknowledgment is received, especially the first or second one after a fresh acknowledgment. If the time since the segment's transmission exceeds the timeout value, Vegas initiates a re-transmission without awaiting a duplicate acknowledgment. Consequently, Vegas exhibits the ability to detect multiple packet losses, overcoming one of Reno's limitations. Moreover, Vegas adjusts its window size only if the re-transmitted segment was sent subsequent to the last reduction, addressing Reno's drawback of multiple reductions in the congestion window when multiple packets are lost [13].

* **Congestion avoidance:**

TCP Vegas stands apart from other implementations due to its distinctive approach to congestion avoidance. Unlike other protocols, it does not interpret segment loss as a clear signal of congestion. Instead, it identifies congestion based on a decline in the sending rate compared to the anticipated rate, reflecting the accumulation of substantial queues within routers. It leverages a modified version of the Wang and Crowcroft Tri-S scheme to facilitate this process, as detailed in the specific reference [13].

Consequently, when the calculated rate significantly deviates from the expected rate, Vegas increases transmissions to utilize the available bandwidth fully. Conversely, when the calculated rate closely aligns with the anticipated value, it reduces transmissions to prevent over-saturation of the bandwidth. This unique approach enables Vegas to effectively manage congestion without inefficiently transmitting data at excessively high rates, which could lead to congestion and subsequent reduction, a pattern observed in other congestion control algorithms [13].

* **Modified Slow start:**

TCP Vegas distinguishes itself from other algorithms particularly during its slow-start phase. This modification arises from the recognition that a new connection lacks knowledge of the available bandwidth, potentially leading to a substantial overshoot of the bandwidth during exponential increase and inducing congestion. As a solution, Vegas incorporates a modified approach, enabling exponential increases only every other round-trip time (RTT). In the interim, it continuously calculates the actual sending throughput relative to the expected throughput. Once the difference surpasses a predetermined threshold, Vegas transitions from the slow-start phase to the congestion avoidance phase [13].

### TCP New Reno

TCP New Reno, an extension of TCP Reno, introduces refinements to the Fast Recovery mechanism. This enhancement allows the protocol to more efficiently manage scenarios where multiple packet losses occur within a single window. By addressing this limitation, TCP New Reno improves the overall robustness and reliability of the congestion control process [14].

One of the key modifications implemented in TCP New Reno is the ability to detect and respond to multiple packet losses within the same window more effectively. This refinement ensures that the protocol can swiftly recover from such instances without significant disruptions to the data transmission process. By enhancing the handling of these scenarios, TCP New Reno aims to provide a more stable and equitable experience for network users, optimizing the utilization of available network resources and promoting fair bandwidth allocation among different connections [14]

* **Slow Start and Congestion Avoidance:**

TCP New Reno starts with the slow start phase, gradually increasing the congestion window size until congestion is detected. Once congestion is detected, it enters the congestion avoidance phase, where the congestion window size is increased linearly by one segment per RTT.

* **Fast Retransmit:**

When TCP New Reno receives three duplicate acknowledgments, indicating the loss of a packet, it assumes that the next segment has been lost and performs a fast retransmit. Instead of waiting for a timeout, it immediately retransmits the presumed lost segment [15].

* **Fast Recovery:**

Upon performing a fast retransmit, TCP New Reno enters the Fast Recovery phase. It reduces the congestion window size by half (to avoid overwhelming the network) and continues to send new packets. For each additional duplicate acknowledgment received, it increases the congestion window size by one segment, similar to TCP Reno. This allows TCP New Reno to recover from multiple packet losses without waiting for a timeout [14].

* **Advantages of TCP New Reno:**

Enhanced Management of Multiple Losses: TCP New Reno's improved Fast Recovery mechanism enables it to efficiently recover from multiple packet losses. Through its ability to address partial acknowledgments, it swiftly detects and retransmits missing segments, mitigating any adverse effects on the overall throughput.

Improved Equitable Behavior: TCP New Reno ensures fairness within the network by adjusting its sending rate in response to congestion signals. This approach guarantees fair distribution of network resources among various TCP flows, promoting an equitable allocation of the available bandwidth.

* **Inconveniences of TCP New Reno:**

Heightened Complexity: TCP New Reno introduces increased intricacy, particularly in managing partial acknowledgments, compared to TCP Reno. This heightened complexity may pose challenges in the implementation and comprehension of the algorithm.

Limitations in High-Latency Networks: Similar to TCP Reno, TCP New Reno's reliance on timeouts and duplicate acknowledgments for congestion detection can result in inefficiencies in networks with high latency. Longer round-trip times (RTTs) and heightened delays may impact the algorithm's performance in such network conditions [14].

While TCP New Reno presents enhancements over TCP Reno in terms of handling multiple packet losses and ensuring fairness, its augmented complexity and potential limitations in specific network environments should be taken into account when selecting an appropriate congestion control algorithm [14].

### TCP CUBIC

TCP Cubic (Congestion Control for TCP) is a TCP congestion control algorithm designed to improve the performance of TCP in high-speed and long-distance networks. It was proposed by Hyong-Soon Kim, Sang-Il Choi, and Chong-Kwon Kim in 2008 [16].

The main idea behind TCP Cubic is to use a cubic function to calculate the congestion window size. The congestion window size determines how many packets can be sent without receiving an acknowledgement from the receiver. TCP Cubic also uses a concept called "TCP Friendly Rate Control" (TFRC) to limit the sending rate of TCP flows in a fair manner.

This algorithm aims to achieve high network utilization and fairness in network resource allocation. The cubic function adjusts the window size based on the elapsed time since the last congestion event and the current congestion window size [17].

TCP Cubic is a congestion control algorithm that builds upon the traditional TCP congestion control mechanisms, such as slow start, congestion avoidance, and Fast Recovery. Its main goal is to improve the performance of TCP in high-speed and long-distance networks, where the conventional TCP algorithms may exhibit limitations.

* **Slow Start:**

TCP Cubic starts with a conservative initial congestion window size, typically one or a few segments. It begins sending data at a slow rate and doubles the congestion window size for every round-trip time (RTT) until congestion is detected [16].

* **Congestion Avoidance:**

After the slow start phase, TCP Cubic enters the congestion avoidance phase. It uses a cubic function to determine the growth of the congestion window size. The cubic function allows for a more gradual and scalable increase in the sending rate, taking into account the past history of congestion and network conditions.

* **TCP Vegas Component:**

TCP Cubic incorporates a component from TCP Vegas, a delay-based congestion control algorithm. It monitors the round-trip time (RTT) of the connection and uses it as an indicator of network congestion. If the RTT increases beyond a certain threshold, TCP Cubic reduces the congestion window size to alleviate congestion. This component helps TCP Cubic to be more responsive to changes in network conditions.

* **Fast Recovery:**

TCP Cubic also includes a Fast Recovery mechanism similar to TCP Reno and TCP New Reno. When it receives three duplicate acknowledgments, indicating a packet loss, it performs a fast retransmit by retransmitting the presumed lost packet without waiting for a timeout. This helps to quickly recover from single packet losses and maintain a reasonable sending rate [16].

* **CUBIC function**

CUBIC, a congestion control algorithm, utilizes the concave and convex characteristics of a cubic function to manage window increase. The algorithm establishes WMAX as the window size at the occurrence of a loss event, subsequently reducing the congestion window multiplicatively by a predefined factor, β. Following this, it employs the standard Fast Recovery and retransmission procedures of TCP [16].

During the transition from Fast Recovery to congestion avoidance, CUBIC initiates window growth based on the concave profile of the cubic function. This growth continues until the window size reaches WMAX, where the cubic function's profile shifts to a convex shape, leading to convex window growth. This dual approach to window adjustment, incorporating both concave and convex profiles, contributes to enhanced protocol and network stability while ensuring consistently high network utilization. The establishment of a plateau around WMAX facilitates optimal network usage and fosters stability within the protocol during steady-state operations [16].

* **Advantages of TCP CUBIC:**

**Scalability in High-Speed Networks:** TCP CUBIC's cubic function for congestion control allows it to scale well in high-speed networks. It provides a more efficient allocation of bandwidth and achieves higher throughput compared to traditional algorithms like TCP Reno [16].

* + **Fairness:**

TCP Cubic aims to provide fair sharing of network resources among competing flows. It adjusts its congestion window size based on network conditions, ensuring that all flows have an equitable opportunity to transmit data [16].

* + **Responsiveness to Network Conditions:**

TCP Cubic integration of the TCP Vegas component allows it to be more responsive to changes in network congestion. By monitoring the RTT and adapting the congestion window size accordingly, it can better adapt to varying network conditions.

* **Inconveniences of TCP CUBIC:**
  + **Increased Complexity:**

TCP Cubic is more complex than traditional TCP congestion control algorithms, such as TCP Reno. Its implementation and understanding require additional effort and expertise.

* + **Potential Unfairness in Certain Scenarios**:

While TCP Cubic aims for fairness, it may exhibit unfairness in scenarios where different flows have significantly different round-trip times. Flows with shorter RTTs might have an advantage, leading to uneven resource allocation.

Overall, TCP Cubic provides advantages in scalability, fairness, and responsiveness in high-speed and long-distance networks. However, its increased complexity and potential fairness issues in specific scenarios should be considered when deploying the algorithm [16].

### TCP BBR

Bottleneck Bandwidth and Round-trip propagation time (BBR), a TCP congestion control algorithm developed by Google in 2016, represents a departure from the conventional loss-based approach. With the increasing bandwidth and overall reliability of the Internet, issues like buffer bloat have become prominent, impacting latency. BBR addresses these challenges by prioritizing latency over lost packets, resulting in improved throughput and reduced latency [18].

Implementing a fundamental rewrite of congestion control, BBR focuses on leveraging latency as a key factor for determining the sending rate. Notably, its efficacy is observed in long-haul paths, such as Transatlantic file transfers, where even minor packet loss can significantly impact throughput. Furthermore, on the last mile path, where Bufferbloat-related latency issues prevail, BBR's approach to avoiding buffer filling results in improved latency.

Notably, BBR has been integrated into the Linux kernel from version 4.9 onwards and can be easily enabled using a simple sysctl command. In practical tests using two Ubuntu machines with 2.5Gbps NICs in the same data center, a single TCP flow between the servers yielded a throughput of 2.35Gb/s, showcasing the promising potential of BBR for network optimization [18].

## TCP optimization

TCP configuration parameters refer to the settings that can be adjusted to fine-tune the behavior and performance of the TCP protocol. These parameters can be modified in the operating system's TCP stack to optimize TCP's operation based on specific network conditions. TCP performance can be optimized through various configuration parameters, including [3]:

### Maximum Segment Size (MSS):

MSS is a parameter that influences the size of data packets in a TCP connection. It specifies the maximum amount of data (in bytes) that can be sent in a single TCP segment. MSS limits the size of packets, or small chunks of data, which travel across a network, such as the Internet [19].

Network data is divided into packets, each containing headers and a payload. MSS, which determines the size of the data part (payload) of a packet, plays a crucial role in the efficient transmission of data. It represents the maximum size of the payload that a device can accept, excluding any attached headers. This value, measured in bytes, helps regulate the fragmentation and reassembly of packets, minimizing unnecessary overhead and enhancing overall efficiency [19].

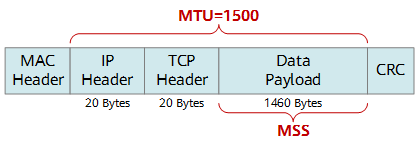


Figure 2.36 *MTU vs MSS* [19]

### Initial Congestion Window (CWND):

CWND, an integral aspect of TCP's congestion control, governs the number of segments a sender can transmit during the initial stages of establishing a connection. Initially set at a conservative size, CWND prevents network overload by cautiously regulating the volume of data transmitted. While the default value is typically around 10 MSS, it can be adjusted based on the perceived congestion level of the network [11].

Any alterations to the default CWND value should be approached with caution and a comprehensive understanding of the specific network conditions. Modifying it without careful consideration may result in inefficient network usage and potential congestion issues. A moderate increase in CWND might prove beneficial if the network can accommodate larger data volumes without congestion. However, any adjustments should be implemented with a thorough comprehension of the network's capacity and congestion patterns [11].

Setting CWND too high can lead to network congestion, subsequently causing packet loss and retransmission, ultimately degrading network performance, increasing latency, and reducing overall throughput. Moreover, an excessively high CWND, implemented without considering network conditions, can result in unfair bandwidth distribution, negatively impacting other users sharing the same network resources. Thus, it is crucial to ensure that any changes to the CWND are made based on a comprehensive understanding of the network's capabilities and limitations [11].

### Maximum Receive Window Size:

In TCP, the Maximum Receive Window Size designates the largest data capacity that the receiving end of a connection can accommodate and store in its buffer before acknowledging the sender. It functions as a crucial flow control mechanism, ensuring that the receiver is not inundated with an excessive data load. During the TCP handshake, the value is negotiated, and it can dynamically change throughout the connection's duration, adapting to factors like available buffer space, network congestion, and the receiver's data processing capabilities. A larger receiver window size facilitates more effective data transmission between the sender and receiver, particularly in high-bandwidth networks [20].

### Maximum Connection Backlog:

The Maximum Receive Window Size serves as a receiver-defined threshold that limits the amount of data that can be buffered, preventing potential segment drops and congestion at the receiver's end. A larger window size enhances data throughput by allowing the sender to transmit more data before awaiting acknowledgments. On the other hand, Maximum Connection Backlog, mainly applicable to server applications, specifies the maximum number of pending connection requests that a server can handle concurrently. When this limit is reached, the server may reject or queue incoming connection attempts. Effective management of the connection backlog is essential for maintaining server resources and ensuring orderly processing of incoming connections, mitigating the risk of resource depletion [20].

### Maximum Retransmission Timeout (RTO):

The Maximum Retransmission Timeout is a critical TCP parameter that governs the retransmission of unacknowledged data segments within a designated time interval. TCP sets a timer upon segment transmission and, if an acknowledgment isn't received within the specified timeout, the segment is deemed lost or delayed, prompting TCP to trigger a retransmission. This maximum timeout serves as an upper limit for the timer. Adjusting this parameter significantly influences the protocol's effectiveness in managing delayed or lost segments. A shorter timeout can lead to more frequent retransmissions, potentially causing congestion and network inefficiencies, while a longer timeout may delay the detection of lost segments, resulting in slower data transfer and increased latency. Striking the right balance in configuring this parameter is crucial for ensuring reliable data transfer while minimizing unnecessary retransmissions [3].

## Classification of TCP Protocols

TCP protocols are differentiated from each other’s on the basis of their congestion control strategy and are classified as shown below:

Table 2.1 *Classification of TCP Protocols*

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| TCP Protocol | | | | | |
| Packet Loss Probability | | | | Queuing Delay based | |
| TCP Tahoe | TCP Reno | TCP New Reno | TCP Cubic | TCP BBR | TCP Vegas |

### Probability-Based Packet Loss:

TCP utilizes various congestion control algorithms, such as TCP New Reno and TCP Cubic, to respond to packet loss caused by network congestion. These algorithms adjust the sending rate based on feedback received from the network, including packet loss indications. By reducing the sending rate when packet loss occurs, TCP aims to alleviate congestion and improve overall network performance [21].

The main flaw of this group of algorithms, including TCP New Reno and TCP Cubic, lies in their reactive nature. These algorithms primarily respond to packet loss as an indicator of network congestion. While they effectively reduce the sending rate upon detecting packet loss, they may take some time to react to changes in network conditions. As a result, there can be a delay in congestion alleviation, leading to temporary network instability and decreased throughput efficiency.

Moreover, these algorithms might not be able to differentiate between different causes of packet loss, such as network congestion, transmission errors, or other network anomalies. Consequently, they may trigger unnecessary reductions in the sending rate, impacting the overall throughput and efficiency of data transfer. Additionally, they might not fully utilize the network capacity when congestion is not the main issue, leading to underutilization of available bandwidth in certain scenarios.

The reactive nature of these algorithms can potentially lead to suboptimal performance, especially in modern network environments where factors like latency and available bandwidth play significant roles in determining the overall network congestion. As a result, the main flaw of these algorithms is their relatively limited adaptability and responsiveness to the intricacies of complex network conditions, potentially hindering their ability to achieve optimal data transfer rates and network performance [3].

### Queueing Delay-Based Packet Loss:

TCP also addresses packet loss caused by queuing delays in buffer overflows. When the queuing delay exceeds a certain threshold, it suggests that the buffer is congested, and packets may be dropped to prevent further congestion. TCP's congestion control mechanisms, monitor the queuing delay and proactively adjust the sending rate to avoid buffer overflow and excessive packet loss [22].

The main flaw of queueing delay-based packet loss algorithms, including TCP BBR, and TCP Vegas, lies in the potential challenge of accurately assessing the causes of queuing delays. While these algorithms attempt to proactively adjust the sending rate based on the measured queuing delay, accurately identifying the specific reasons for the delay can be complex and challenging. Factors such as varying network conditions, diverse network topologies, and the presence of transient or intermittent congestion can make it difficult for these algorithms to precisely attribute queuing delays to buffer overflows or network congestion.

Additionally, in dynamic network environments where queuing delays can be influenced by various factors other than network congestion, such as routing changes or temporary traffic spikes, there is a risk of these algorithms triggering unnecessary adjustments in the sending rate. This can potentially lead to underutilization of available network resources and reduced overall throughput, impacting the efficiency of data transmission.

Furthermore, while these algorithms aim to proactively adjust the sending rate to prevent buffer overflow and excessive packet loss, they may encounter difficulties in distinguishing between different types of queuing delays, potentially leading to suboptimal performance in certain network conditions. As a result, the main challenge for these algorithms lies in accurately interpreting queuing delays and effectively differentiating between normal fluctuations and congestion-related issues, in order to optimize the sending rate and ensure efficient data transmission [11].

## Literature review and state of the art

Numerous studies have been conducted on the impact of TCP parameters on transmission performance. Jiang and Zhang (2018) studied the effect of window size and maximum segment size on TCP performance and proposed a hybrid congestion control algorithm that combines the benefits of both parameters. Xu et al. (2016) investigated the impact of congestion window size on TCP performance and proposed a dynamic congestion control algorithm that adapts to changing network conditions. Ren et al. (2019) analyzed the effect of round-trip time on TCP performance and proposed a round-trip time-based congestion control algorithm.

## Conclusion of chapter

In this chapter, we examined and analyzed several TCP variants, including TCP Tahoe, TCP Reno, TCP Vegas, TCP New Reno, TCP Cubic, and TCP BBR. Each TCP variant showcased different approaches to handle packet loss and manage congestion in the network.

TCP Tahoe and TCP Reno, being probability-based protocols, respond to packet loss by reducing the sending rate, thereby controlling congestion. These variants are well-established and widely used in various network environments. TCP New Reno, an extension of TCP Reno, introduced some enhancements in its Fast Recovery algorithm.

On the other hand, TCP Vegas is a queueing delay-based protocol that focuses on minimizing queuing delay rather than relying solely on packet loss as an indicator of congestion. It offers the potential for better performance in scenarios where queuing delay is a significant concern.

TCP Cubic, designed for high-speed and long-distance networks, utilizes a cubic function to adjust the congestion window. It aims to achieve high network utilization and fairness, particularly in challenging network conditions.

TCP BBR, a modern TCP congestion control algorithm, considers both bandwidth and Round-Trip Time (RTT) to optimize performance. It provides efficient utilization of available bandwidth, low queuing delay, and reduced packet loss.

The analysis of these TCP variants highlighted the importance of selecting the appropriate TCP protocol based on the network characteristics, requirements, and objectives. Different TCP variants offer distinct advantages and trade-offs in terms of throughput, fairness, latency, and congestion control.

By understanding the characteristics and behaviors of these TCP variants, network administrators, researchers, and engineers can make informed decisions regarding protocol selection to optimize transmission performance based on specific network conditions.

In the next chapter, we will conduct experiments using TCP New Reno and TCP Cubic as our selected TCP algorithms. By varying the aforementioned TCP parameters individually and in combination, we will evaluate their impact on transmission performance metrics such as throughput, latency, fairness, and network utilization. These experiments will provide insights into how specific TCP parameters influence the performance of TCP New Reno and TCP Cubic in different network scenarios

# Evaluation of TCP Parameters on Transmission Performance

## Introduction

The performance of TCP (Transmission Control Protocol) is crucial for ensuring efficient and reliable data transmission in computer networks. The selection and configuration of TCP parameters significantly impact the overall performance of TCP connections. Therefore, it is essential to understand how specific TCP parameters influence transmission performance to optimize network performance and user experience.

This chapter focuses on the experimental evaluation of TCP parameters on transmission performance using two widely used TCP variants: TCP New Reno and TCP Cubic. These variants were chosen due to their popularity and distinct characteristics in handling congestion and adapting to network conditions.

The objective of this chapter is to investigate the effects of selected TCP parameters on key performance metrics, such as throughput, drop ratio, and fairness. By systematically varying and analyzing these parameters, we aim to gain insights into their impact on the overall performance of TCP connections.

## Methodology

To conduct the experimental evaluation, we will adopt a systematic methodology that ensures accurate and reliable results. The following steps will be followed:

We will set up a controlled network environment, consisting of simulation tools (NS3), to create reproducible test scenarios. This environment will allow us to manipulate TCP parameters and observe their effects on transmission performance.

Selection of TCP Parameters: we will carefully select a set of TCP parameters that have a significant influence on congestion control, flow control, and overall performance. These parameters may include the initial congestion window size, segment size, congestion control algorithms, retransmission timeout values, among others. For how experiment we will use two parameters : segment size and initial congestion window.

Experimental design: we will design a series of experiments where TCP New Reno and TCP Cubic connections will be established. The selected TCP parameters will be systematically varied within the experiments to observe their impact on performance metrics.

Performance metrics: we will measure and analyze key performance metrics, including throughput, drop ratio, and fairness. These metrics will provide insights into the behavior and efficiency of TCP connections under different parameter configurations.

Data collection and Analysis: experimental data will be collected and analyzed to identify patterns, trends, and correlations between TCP parameters and transmission performance. Statistical analysis techniques may be applied to validate the significance of the results.

Comparison and Evaluation: we will compare the performance of TCP New Reno and TCP Cubic under various parameter configurations. This comparison will allow us to evaluate the strengths and weaknesses of each variant and assess their suitability for different network scenarios.

The outcomes of this experimental evaluation will contribute to a deeper understanding of the influence of TCP parameters on transmission performance. The findings will guide network administrators, researchers, and engineers in making informed decisions when configuring TCP parameters to optimize network performance and enhance the user experience.

## NS-3 Network Simulation 3(NS3)

Network simulation is pivotal in comprehending the behaviors and dynamics of diverse networking technologies and protocols. NS-3, also known as Network Simulator 3, is a prevalent open-source framework for simulating discrete-event networks, specifically tailored for research and educational purposes. Serving as an evolution of the well-known NS-2 simulator, NS-3 was developed to surpass the constraints of its predecessor and introduce a host of advanced functionalities. Its comprehensive library encompasses an array of pre-implemented network models and protocols, ranging from fundamental point-to-point and CSMA/CD models to sophisticated ones like Wi-Fi, WiMAX, and LTE, facilitating intricate investigations into various networking scenarios. By enabling detailed and accurate simulations of both wired and wireless networks, NS-3 empowers researchers, developers, and educators to evaluate the performance of different communication protocols and design intricate network systems effectively.

## Network Topology

We construct a dumbbell topology, where routers are located at the bottleneck link between two end points, and we use three senders on router1 and three receivers on router2.

A diagram of a router

Description automatically generated

Figure 3.1 *Design the Network Topology*

The topology Figure 3.1 has Six nodes, labeled as n0 through n5, are strategically positioned to fulfill distinct roles within the simulated network. Specifically, nodes n0, n1, and n2 serve as senders, generating data packets for transmission. Conversely, nodes n5, n3, and n4 function as receivers, tasked with capturing and processing the incoming data. This differentiation in roles allows us to accurately capture the dynamics of data exchange and evaluate the impact of TCP parameters on transmission performance.

Two routers, denoted as r0 and r1, act as pivotal points of connectivity within the topology. These routers play a crucial role in forwarding data between senders and receivers, simulating the intricate routing mechanisms present in real networks. The routers are interconnected through point-to-point links, ensuring the establishment of a network path for data transmission. The data rate for each node and routers are set to 10Mbps, the delay for all of them are set to 2ms, the maximum queue size we set it at 200packets.

### Connections

We plan to carry out two experiments, each concentrating on distinct aspects of the network simulation. The first experiment centers on the variation in segment sizes and involves a total of nine connections. All connections will be initiated simultaneously, and the experiment will employ a standard file size across all connections. In the second experiment, we aim to analyze the impact of different initial congestion window sizes on the network's behavior. This experiment comprises three connections, with the file size set to 10Mbps. The connections will be initiated at staggered intervals, with the first connection start at 0 seconds, the second at 11 seconds, and the final one at 21 seconds.

### Configurations

The MTU (Maximum Transmission Unit) represents the largest data size that can be transmitted across a network without fragmentation. It establishes the maximum packet size allowable for a single network layer transaction. In contrast, the segment size denotes the size of data chunks or segments being transmitted over the network. Our experiment includes testing various segment sizes, such as 50 bytes, 100 bytes, 500 bytes, 1000 bytes, and 1440 bytes, to observe the impact on network performance.

In terms of the application utilized, the BulkSendApplication in NS-3 is deployed for simulating bulk data transfer within the network. This application generates a continuous flow of bulk data from a source node to a destination node within the simulated network environment.

During the simulation, we have configured the data rate to 50Mbps, indicating the maximum data transfer speed within the network. The delay is set to 2 milliseconds, representing the estimated time taken for data packets to travel from the source to the destination. To manage the queue, we employ the First-In-First-Out (FIFO) policy, ensuring that data packets are processed in the order of their arrival. The queue size is set to 200 packets, signifying the maximum number of data packets the network can accommodate in the queue before additional packets may be subjected to alternate handling procedures or dropped. Furthermore, we have designated a transmission time of 100 seconds, providing us with an extensive observation period to assess the network's behavior comprehensively. This setting allows for a detailed analysis of how various network configurations and conditions influence the overall data transmission process. We have designated a transmission time of 100 seconds, which serves as a crucial parameter determining the duration over which data is transmitted from the source to the destination within the simulated network environment. This specific setting enables us to closely observe and analyze the network's behavior over an extended period, allowing for comprehensive evaluations of how various network configurations and conditions influence the overall data transmission process.

# The simulation results and analysis

This chapter is dedicated to presenting the results of the simulation and analyzing their implications. We conducted two experiments, one focusing on segment size and the other on the initial congestion window. For the segment size experiment, we examined how varying segment sizes impact network parameters such as throughput, goodput, fairness, and drop ratio. In the case of the initial congestion window experiment, we observed how different initial congestion window values affect the time taken for transmitted packets in each flow.

## segment size experiment

Let's begin by delving into the findings from the segment size experiment, the segment size in a TCP communication refers to the maximum amount of data that can be transmitted within a single TCP segment. This experiment aims to investigate the impact of varying segment sizes on the following network parameters within the NS-3 simulation environment:

* Throughput
* Goodput
* Fairness
* Drop Ratio

We will investigate how Segment size influence transmission performance for those parameters above by using TCP Cubic and TCP New Reno.

### Segment Size vs. Throughput

We delve into the relationship between segment size and throughput within the context of our study. This analysis focuses on segment sizes of 50 bytes, 100 bytes, 500 bytes, 1000 bytes, and 1440 bytes, aiming to discern the impact of varying segment dimensions on the resulting data throughput measured in megabits per second (Mbps). Throughput quantifies the rate at which data is effectively transmitted over a network connection.

The Table 3.1 illustrates the outcomes of our experimental investigation, wherein we systematically varied the segment size and assessed its consequences on throughput:

Table 3.1 Influence of Segment Size[B] on Throughput[Mbps]

|  |  |  |
| --- | --- | --- |
| SEGMENT SIZE | CUBIC | NEW RENO |
| 50 | 9.35 | 9.35 |
| 100 | 9.41 | 9.41 |
| 500 | 9.51 | 9.50 |
| 1000 | 9.53 | 9.53 |
| 1440 | 9.54 | 9.54 |

* **Observations and Analysis**

Figure 3.2 *Throughput vs segment size*

As the segment size increases incrementally from 50 to 1440 bytes, a discernible pattern emerges in the throughput results for both the CUBIC and NEW RENO congestion control algorithms. The obtained data consistently demonstrates an upward trajectory in throughput with the augmentation of segment size.

At the smallest segment size tested (50 bytes), the throughput for both algorithms are approximately 9.35 Mbps. With a segment size of 100 bytes, the throughput marginally improves to 9.41 Mbps.

However, as we transition to larger segment sizes of 500, 1000, and 1440 bytes, the throughput continues to rise steadily. At the largest segment size tested (1440 bytes), the throughput reaches approximately 9.55 Mbps for both TCP Cubic and TCP New Reno.

The experiment demonstrates a clear trend in throughput increase with larger segment sizes for both the TCP Cubic and TCP New Reno congestion control algorithms. While the difference in throughput between smaller and larger segment sizes appears moderate, the impact becomes notable due to the sheer volume of packets. With smaller segments, the necessity to transmit more packets places a greater load on routers, especially those with limited resources, potentially leading to congestion. This elucidates why the overall throughput is notably affected by the segment size, emphasizing the importance of segment size optimization for efficient data transmission.

### Segment Size vs. Goodput

Goodput, also known as effective throughput or application-level throughput, is a network performance metric that measures the rate at which useful data is transmitted over a network, excluding any protocol overhead, retransmissions. It represents the actual data payload delivered successfully from the source to the destination.

Let's analyze the data to understand the influence of segment size on goodput for both the TCP Cubic and TCP New Reno congestion control algorithms:

Table 3.2 *Segment Size[B] vs. Goodput[Mbps]*

|  |  |  |
| --- | --- | --- |
| SEGMENT SIZE | GOODPUT CUBIC | GOODPUT NEWRENO |
| 50 | 3.85 | 3.85 |
| 100 | 5.69 | 5.69 |
| 500 | 8.47 | 8.47 |
| 1000 | 8.98 | 8.98 |
| 1440 | 9.16 | 9.16 |

Increasing Segment Size Enhances Goodput: similar to the trend observed with throughput, there is a consistent pattern in the data regarding the influence of segment size on goodput for both TCP Cubic and New Reno congestion control algorithms. As the segment size increases from 50 bytes to 1440 bytes, both algorithms show a steady increase in goodput.

the augmentation in segment size notably amplifies the goodput, following a consistent pattern across both the TCP Cubic and New Reno congestion control algorithms. Unlike the observations made for throughput, the variations in goodput with varying segment sizes exhibit a significantly pronounced impact, emphasizing the critical role of segment size in enhancing data transfer efficiency.

For example, with a segment size of 50 bytes, the goodput for both algorithms are approximately 3 Mbps. However, as the segment size increases, the goodput gradually improves.

At the largest segment size tested (1440 bytes), the goodput for both algorithms reach approximately 9 Mbps.

Similar Performance between TCP Cubic and New Reno: similar to the throughput results, it's notable that for each segment size, the goodput values for both TCP Cubic and New Reno are almost identical. This suggests that the choice of congestion control algorithm has little to no impact on the observed trend of increasing goodput with larger segment sizes in this specific experiment.

Figure 3.3 *Segment Size vs. Goodput*

The observed relationship between segment size and goodput can be explained by the same underlying principles as in the throughput analysis. Larger segment sizes allow more data to be transmitted in each packet, reducing overhead and transmission delays, which leads to higher goodput.

Goodput specifically measures the rate of successfully delivered application data, excluding protocol overhead and retransmissions. Therefore, as segment size increases and more data can be packed into each packet, a larger proportion of the transmitted data becomes application payload, resulting in higher goodput.

### Segment Size vs Drop Ratio

Continuing our exploration, we now shift our attention to the examination of "Segment Size and drop ratio." This analysis aims to provide insights into the interplay between different segment sizes and the occurrence of dropped segments in the network. By investigating drop ratios across varying segment dimensions, we strive to better understand how the choice of segment size influences the network's responsiveness and stability under different conditions.

let's analyze the data to understand how segment size influences drop ratio for both TCP Cubic and TCP New Reno congestion control algorithms:

Table 3.3 *Segment Size[B] vs Drop Ratio[%]*

|  |  |  |
| --- | --- | --- |
| SEGMENT SIZE | DROP RATIO CUBIC | DROP RATIO NEW RENO |
| 50 | 0.06% | 0.06% |
| 100 | 0.10% | 0.08% |
| 500 | 0.27% | 0.21% |
| 1000 | 0.49% | 0.38% |
| 1440 | 0.73% | 0.53% |

As the segment size increases from 50 bytes to 1440 bytes, both TCP Cubic and TCP New Reno congestion control algorithms exhibit an increase in drop ratios.

For example, at the smallest segment size tested (50 bytes), the drop ratio for both algorithms are 0.06% that is negligible. However, as the segment size increases, the drop ratios consistently rise but not too much.

At the largest segment size tested (1440 bytes), the drop ratio increases to 0.73% for TCP Cubic and 0.53% for TCP New Reno.

With the escalation of segment size from 50 bytes to 1440 bytes, both TCP Cubic and TCP New Reno congestion control algorithms demonstrate a progressive elevation in drop ratios. While the initial drop ratios at the smallest segment size (50 bytes) remain minimal at 0.06% for both algorithms, there is a consistent upward trend as the segment size increases. Notably, TCP Cubic consistently exhibits slightly higher drop ratios compared to TCP New Reno throughout the range of segment sizes. Nevertheless, the overall pattern emphasizes the tendency of drop ratios to increase with larger segment sizes for both congestion control algorithms.

Figure 3.4 *Segment Size and drop ratio.*

The relationship between segment size and drop ratio is closely intertwined with the interaction between congestion control algorithms and network conditions. Larger segment sizes, synonymous with higher data volume within each packet, can amplify the likelihood of packet drops during network congestion events. The drop ratio, although generally below 1%, serves as an indicator of network congestion, highlighting the percentage of packets that were discarded to alleviate traffic pressure.

Surprisingly, a higher drop ratio corresponds to higher throughput and goodput, indicative of a more efficient transmission despite the apparent congestion. This can be attributed to the fact that despite the increased drop ratio and congestion, larger segment sizes enable the more efficient transmission of the remaining packets, leading to enhanced overall throughput and goodput. The optimization of packet size seems to play a critical role here, ensuring that the transmission remains efficient even in the face of network congestion.

* **Algorithm Differences**:

The slight difference in drop ratios between TCP Cubic and TCP New Reno may be attributed to variations in how these congestion control algorithms respond to network congestion and how they estimate available bandwidth. TCP Cubic tends to be more aggressive in probing for available bandwidth, which may lead to slightly higher drop ratios in this scenario Figure 3.4.

### segment size vs fairness

As we delve further into the intricate dynamics of network performance, the exploration shifts to the relationship between segment size and fairness. In this section, we investigate how different segment sizes influence the fairness between multiple flows or connections within the network. By analyzing the fairness metrics across varying segment dimensions, we aim to uncover insights into the equitable distribution of network resources and the potential impact of segment size on this fundamental aspect of network behavior.

|  |  |  |
| --- | --- | --- |
| SEGMENT SIZE | TCP Cubic | TCP New Reno |
| 50 | 0.98 | 0.97 |
| 100 | 0.97 | 0.97 |
| 500 | 0.98 | 0.95 |
| 1000 | 0.97 | 0.98 |
| 1440 | 0.96 | 0.94 |

Table 3.4 *segment size[B] vs fairness*

Varied Fairness with Changing Segment Size: The data demonstrates that fairness, as measured between TCP Cubic and TCP New Reno congestion control algorithms, varies with different segment sizes.

For instance, with a segment size of 50 bytes, TCP Cubic exhibits a fairness index of approximately 0.988, while TCP New Reno has a fairness index of about 0.972.

As the segment size changes, the fairness indices also change, indicating different levels of fairness between the two algorithms.

Segment Size Impact on Fairness: There is no consistent trend regarding how changes in segment size affect fairness. In some cases, increasing the segment size leads to higher fairness (e.g., from 50 to 1000 bytes), while in other cases, it results in lower fairness (e.g., from 1000 to 1440 bytes).

Figure 3.5 *segment size vs fairness*

Fairness in network communications refers to how equitably network resources are allocated among competing connections or flows. A fairness index of 1.0 indicates perfect fairness, where all flows receive an equal share of the available bandwidth.

Upon comprehensive examination of the influence of segment size on network performance, it becomes apparent that larger segment sizes consistently improve both throughput and goodput in various experiments. This improvement is attributed to the reduction of packet header overhead and the efficient use of available bandwidth by transmitting more data per packet. However, this advantage is counterbalanced by higher drop ratios during congestion, particularly for longer packets that are more susceptible to being discarded.

Furthermore, while segment size impacts the fairness between competing flows, no definitive trend emerges from the current analysis. Understanding the relationship between segment size and fairness would require additional simulations, particularly with varying numbers of flows, to draw more conclusive insights. The consistent trends across TCP Cubic and TCP New Reno congestion control algorithms underscore the critical role of segment size in shaping overall network performance, surpassing the influence of the specific congestion control mechanism employed. Selecting an optimal segment size that aligns with specific network requirements and balances efficiency, latency, and fairness remains crucial in network design and management.

## Initial Congestion Window experiment

Continuing our investigation, we now delve into an experiment that examines the implications of different initial congestion window (cwnd) values on various network parameters, encompassing the time of transmitted packets for each flow, and the associated Drop Ratio. In this experiment, we will be utilizing three network flows. By considering initial cwnd settings of 1, 3, 10, 20, and 50, we endeavor to uncover the intricate relationships between these diverse starting points and their effects on network performance metrics.

### Initial congestion window vs time transmitted of packet.

For various flows and two TCP congestion control algorithms, TCP Cubic and TCP New Reno, under different Initial Congestion Window (ICWND) settings. The file size used for these experiments is 50 MB, data rate is 10MB and delay 2ms. Let's break down the results for each ICWND setting with file size 50MB:

* **Initial Congestion Window (ICWND) 1:**

Table 3.5 *ICWND 1*

|  |  |  |
| --- | --- | --- |
| FLOW ID | CUBIC | NEW RENO |
| 1 | 8639 | 8602 |
| 2 | 8637 | 8591 |
| 3 | 8637 | 8591 |

TCP Cubic: with an ICWND of 1, the initial congestion window is very conservative, allowing only a minimal number of packets to be sent initially. Consequently, all three flows experience relatively high delays in packet transmission. Flow 1 takes 8,639 milliseconds, Flow 2 takes 8,637 milliseconds, and Flow 3 takes 8,637 milliseconds.

TCP New Reno: similarly, in the New Reno algorithm with ICWND = 1, delays remain relatively high for all flows. Flow 1 takes 8,602 milliseconds, Flow 2 takes 8,591 milliseconds, and Flow 3 takes 8,591 milliseconds. resulting in relatively high delays for all flows in both the TCP Cubic and TCP New Reno algorithms.

Figure 3.6 *Icwnd1*

* **Initial Congestion Window (ICWND) 3:**

Table 3.6 *ICWND 3*

|  |  |  |
| --- | --- | --- |
| FLOW ID | CUBIC | NEW RENO |
| 1 | 8618 | 8581 |
| 2 | 8617 | 8570 |
| 3 | 8617 | 8570 |

TCP Cubic: increasing ICWND to 3 allows for a larger initial congestion window, which results in reduced delays compared to ICWND = 1. In the Cubic algorithm, Flow 1 takes 8,618 milliseconds, Flow 2 takes 8,617 milliseconds, and Flow 3 takes 8,617 milliseconds.

TCP New Reno: similarly, in the New Reno algorithm with ICWND = 3, delays are reduced compared to ICWND = 1. Flow 1 takes 8,581 milliseconds, Flow 2 takes 8,570 milliseconds, and Flow 3 takes 8,570 milliseconds.

Figure 3.7 *Icwnd 3*

* **Initial Congestion Window (ICWND) 10:**

Table 3.7 *ICWND 10*

|  |  |  |
| --- | --- | --- |
| FLOW ID | CUBIC | NEW RENO |
| 1 | 8598 | 8561 |
| 2 | 8596 | 8550 |
| 3 | 8590 | 8550 |

TCP Cubic: further increasing ICWND to 10 allows for more aggressive initial transmissions, resulting in shorter delays for all flows. In the Cubic algorithm, Flow 1 takes 8,598 milliseconds, Flow 2 takes 8,596 milliseconds, and Flow 3 takes 8,590 milliseconds.

TCP New Reno: in the New Reno algorithm with ICWND = 10, similar reductions in delays are observed. Flow 1 takes 8,561 milliseconds, Flow 2 takes 8,550 milliseconds, and Flow 3 takes 8,550 milliseconds.

Figure 3.8 *Icwnd 10*

* **Initial Congestion Window (ICWND) 20:**

|  |  |  |
| --- | --- | --- |
| FLOW ID | CUBIC | NEW RENO |
| 1 | 8591 | 8554 |
| 2 | 8589 | 8543 |
| 3 | 8589 | 8543 |

Table 3.8 *ICWND 20*

TCP Cubic: ICWND = 20 continues the trend of larger initial congestion windows, which further reduces delays. In the Cubic algorithm, Flow 1 takes 8,591 milliseconds, Flow 2 takes 8,589 milliseconds, and Flow 3 takes 8,589 milliseconds.

TCP New Reno: Similarly, in the New Reno algorithm with ICWND = 20, delays remain reduced. Flow 1 takes 8,554 milliseconds, Flow 2 takes 8,543 milliseconds, and Flow 3 takes 8,543 milliseconds.

Figure 3.9 *Icwnd 20*

* **Initial Congestion Window (ICWND) 50:**

Table 3.9 *ICWND 50*

|  |  |  |
| --- | --- | --- |
| FLOW ID | CUBIC | NEW |
| 1 | 8591 | 8554 |
| 2 | 8589 | 8543 |
| 3 | 8589 | 8543 |

TCP Cubic: with the largest ICWND of 50, delays remain relatively constant compared to ICWND = 20. In the Cubic algorithm, Flow 1 takes 8,591 milliseconds, Flow 2 takes 8,589 milliseconds, and Flow 3 takes 8,589 milliseconds.

Figure 3.10 *Icwnd 50*

TCP New Reno: in the New Reno algorithm with ICWND = 50, similar delays are observed. Flow 1 takes 8,554 milliseconds, Flow 2 takes 8,543 milliseconds, and Flow 3 takes 8,543 milliseconds.

Increasing the ICWND generally leads to a reduction in packet transmission times for the Cubic and New Reno algorithms. This is because a higher ICWND allows more packets to be sent initially, which saturates the available bandwidth more quickly and reduces the time taken to transmit the data. However, beyond a certain point (around ICWND = 20), further increases in ICWND do not lead to significant reductions in delay, suggesting that other factors or network conditions may come into play. However, by adding the bandwidth from 10MB to 50MB, we found that in a congested or high bandwidth network, a larger ICWND can lead to faster transmissions. In a congested or low bandwidth network, a larger ICWND can lead to more congestion and longer delays.

### Initial congestion window with drop ratio

In this section, we will focus on the relationship between ICWND (Initial Congestion Window) and drop ratio, examining how increasing ICWND influences the drop ratio. Let's begin with ICWND set to 1:

* **Initial Congestion Window (ICWND) 1:**

In this section, we will focus on the relationship between ICWND (Initial Congestion Window) and drop ratio, examining how increasing ICWND influences the drop ratio. Let's begin with ICWND set to 1:

|  |  |  |
| --- | --- | --- |
| FLOW ID | CUBIC | NEW RENO |
| 1 | 0% | 0% |
| 2 | 0.36% | 0.19% |
| 3 | 0.71% | 1.29% |

Table 3.10 *Icwnd 1 vs drop ratio.*

With ICWND (Initial Congestion Window) set to 1, both TCP Cubic and TCP New Reno start with a very conservative initial window size.

TCP Cubic: Successfully maintains a 0% drop ratio for all three flows. Even with the smallest ICWND, it exhibits effective congestion control without any packet losses.

TCP New Reno: Also performs well, achieving a 0% drop ratio for Flows 1 and 2. For Flow 3, there is a minimal increase in the drop ratio to 1.29%, but it remains relatively low.

Figure 3.11 *Icwnd 1 vs drop ratio.*

* **Initial Congestion Window (ICWND) 3:**

Table 3.11 *ICWND 3 VS DROP RATIO*

|  |  |  |
| --- | --- | --- |
| FLOW ID | CUBIC | NEW RENO |
| 1 | 0% | 0% |
| 2 | 0.36% | 0.19% |
| 3 | 0.71% | 1.34% |

As ICWND is increased to 3, flows have a slightly larger initial window to begin with.

TCP Cubic: Maintains a 0% drop ratio for all flows, demonstrating its ability to adapt to the increased ICWND effectively.

TCP New Reno: Remains strong with a 0% drop ratio for Flows 1 and 2. For Flow 3, there is a slight increase in the drop ratio to 1.34%, which is still relatively low.

Figure 3.12 *Icwnd 1 vs drop ratio.*

* **Initial Congestion Window (ICWND) 10:**

Table 3.12 *ICWND 10 VS DROP RATIO*

|  |  |  |
| --- | --- | --- |
| FLOW ID | CUBIC | NEW RENO |
| 1 | 0% | 0% |
| 2 | 0.36% | 0.19% |
| 3 | 0.71% | 1.31% |

ICWND is further increased to 10, providing a more generous initial window size.

TCP Cubic: Continues to excel with a 0% drop ratio for all flows, even with the larger ICWND.

TCP New Reno: Maintains a 0% drop ratio for Flows 1 and 2. For Flow 3, the drop ratio increases slightly to 1.31%, which remains relatively low.

Figure 3.13 *ICWND 10 VS DROP RATIO*

* **Initial Congestion Window (ICWND)20:**

Table 3.13 *ICWND 20 VS DROP RATIO*

|  |  |  |
| --- | --- | --- |
| FLOW ID | CUBIC | NEW RENO |
| 1 | 0% | 0% |
| 2 | 0.36% | 0.19% |
| 3 | 0.71% | 1.32% |

ICWND is set to 20, offering a larger initial window.

TCP Cubic: Still achieves a 0% drop ratio for all flows, showcasing its ability to handle larger ICWND values effectively.

TCP New Reno: Remains strong with a 0% drop ratio for Flows 1 and 2. For Flow 3, the drop ratio increases slightly to 1.32%, which is still relatively low.

Figure 3.14 *ICWND 20 VS DROP RATIO*

* **Initial Congestion Window (ICWND) 50:**

Table 3.14 *ICWND 50 VS DROP RATIO*

|  |  |  |
| --- | --- | --- |
| FLOW ID | CUBIC | NEW RENO |
| 1 | 0% | 0% |
| 2 | 0.36% | 0.19% |
| 3 | 0.71% | 1.30% |

With ICWND set to 50, flows start with a significantly larger initial window. TCP Cubic continues to maintain a 0% drop ratio for all flows, even with the larger ICWND and the TCP New Reno remains effective with a 0% drop ratio for Flows 1 and 2. For Flow 3, the drop ratio increases slightly to 1.30%, which is still relatively low.

Figure 3.15 *ICWND 50 VS DROP RATIO*

The data demonstrates that both TCP Cubic and TCP New Reno exhibit robust congestion control capabilities across a range of ICWND values. Even with larger initial congestion window sizes, both algorithms effectively manage congestion and keep the drop ratios at relatively low levels, showcasing their adaptability and reliability in various network conditions.

## Limitations of works

The limitations of this study are crucial to consider when interpreting its findings. Firstly, the choice of a simplified network topology involving only 8 devices, including 6 nodes and 2 routers, may not adequately capture the intricacies of larger and more diverse real-world networks with numerous devices, various network elements, and intricate configurations.

Secondly, the reliance on fixed network parameters, such as a constant data rate, a 2ms delay, and a maximum queue size of 200 packets, may not accurately represent the dynamic and variable conditions typically encountered in real-world networks. Network conditions can vary widely due to factors like traffic load, link congestion, and network congestion.

Thirdly, the study primarily focuses on two specific TCP variants, CUBIC and NEW RENO, which are only a subset of the available TCP congestion control algorithms. The exclusion of other variants limits the generalizability of the study's findings, as different TCP algorithms may interact differently with the studied parameters.

Additionally, the use of static segment sizes in the experiments may not fully account for the dynamic nature of segment size adjustments in real-world TCP implementations. Real networks often adapt segment sizes based on changing network conditions to optimize performance.

Furthermore, the experiments predominantly concentrate on a single bottleneck link, overlooking the potential complexities introduced by multiple bottlenecks in more intricate network topologies. In reality, networks can have multiple points of congestion, each with its characteristics and interactions with TCP parameters.

Lastly, the results are based on simulations conducted in the NS3 simulator, which involves simplifications and assumptions about network behavior. While simulations are valuable for controlled experiments, these assumptions may not perfectly mirror the intricate and dynamic nature of real-world networks.

## Conclusion of chapter

In this chapter, we explored the impact of two critical TCP parameters, segment size and Initial Congestion Window (ICWND), on the performance of TCP Cubic and TCP New Reno.

**Segment Size**: our experiments revealed that segment size significantly influences performance metrics. Larger segment sizes tended to increase throughput but also resulted in a slight uptick in packet drop ratios. Both TCP Cubic and TCP New Reno consistently exhibited high fairness across varying segment sizes, underscoring their ability to ensure equitable resource allocation in diverse network conditions.

**ICWND:** increasing the ICWND parameter enables faster initial data transmission without significantly increasing the risk of congestion-related packet drops. This is particularly beneficial for applications that prioritize low latency and require rapid data transfer for a seamless user experience. By adjusting ICWND strategically, network performance can be optimized to meet the specific needs of different applications.

# Summary

This research delves into the intricate relationship between selected TCP (Transmission Control Protocol) parameters, namely segment size and initial congestion window (ICWND), and their profound influence on transmission performance. Conducted using the NS3 simulator within a dumbbell topology involving 8 network devices comprising 6 nodes and 2 routers, the study explores these parameters' impact comprehensively.

In the realm of segment size, the research embarks on a series of meticulous experiments, varying segment sizes from 50 to 1440 bytes. These experiments scrutinize metrics such as average throughput, goodput, packet drop ratios, and fairness, serving as vital indicators of network performance for two TCP variants: CUBIC and NEW RENO. The findings emphasize that larger segment sizes can significantly boost data transfer rates, yet this advantage comes at the cost of increased susceptibility to packet drops due to network congestion. Hence, the choice of segment size should be a judicious one, carefully tailored to align with specific application requirements and network capacity.

Furthermore, the research extends its exploration to the ICWND parameter, varying its values from 1 to 50 in dedicated experiments. The results reveal that higher ICWND values facilitate faster data transmission without disproportionately elevating packet drop rates, making ICWND a strategic tool for optimizing data transfer speed, particularly in applications prioritizing low latency.

In addition to these conclusions, the research identifies intriguing avenues for future exploration. These include the investigation of dynamic segment size adaptation mechanisms, a broader examination of TCP variants, an exploration of the impact of segment size and ICWND in the context of Multipath TCP (MPTCP), the integration of machine learning techniques to craft adaptive algorithms for parameter selection, and the essential transition from simulations to real-world testing for practical validation and network efficiency enhancements. Overall, this research not only sheds light on the significance of parameter configurations in network efficiency but also serves as a foundation for further developments in network optimization and adaptability.

# Bibliography

[1] R. 793, "TRANSMISSION CONTROL PROTOCOL," September 1981. [Online]. Available: https://datatracker.ietf.org/doc/html/rfc793.

[2] J. F. &. R. Kurose, "top-down approach," in Computer networking, 2017.

[3] M. P. V. &. S. Allman, TCP congestion control, IETF, 1999.

[4] R. 793, "TRANSMISSION CONTROL PROTOCOL," September 1981. [Online]. Available: https://datatracker.ietf.org/doc/html/rfc793.

[5] Chris.m.chavez, "NetworkLessons," 2013-2023. [Online]. Available: https://networklessons.com/cisco/ccie-routing-switching-written/tcp-window-size-scaling.

[6] N. K. Nainar, Network Analysis using Wireshark, 2. C. Edition, Ed., packt publishing , 2020.

[7] K. Academy, "Transmission Control Protocol (TCP)," Khan Academy, 2023. [Online]. Available: https://pl.khanacademy.org/computing/ap-computer-science-principles/the-internet/x2d2f703b37b450a3:transporting-packets/a/transmission-control-protocol--tcp.

[8] "IONOS," DIGITAL GUIDE, 03 02 2020. [Online]. Available: https://www.ionos.com/digitalguide/server/know-how/introduction-to-tcp/.

[9] Mansi, "TCP Congestion Control," pp. 1-3, 12 september 2022.

[10] Satish Kumar, "TCP Tahoe and TCP Reno," Ed Tech company, 11 july 2023. [Online]. Available: https://www.tutorialspoint.com/tcp-tahoe-and-tcp-reno.

[11] V. Jacobson, "Congestion Avoidance and Control," Computer Communication Review, Vols. vol. 18, no., no. ACM SIGCOMM, pp. 4, pp. 314-329, 1988.

[12] Jacobson, "Modification TCP Congstion Control and Avoidanc Algorithms," Technical, vol. V, p. 30, 1990.

[13] A.-H. E-.Altman, "Analysis of TCP Reno and TCP Vegas".

[14] S. H. Lisong Xu, "CUBIC: A New TCP-Friendly High-Speed TCP Variant," Raleigh, NC 27695.

[15] D. S. Ullah, "Modeling TCP NewReno Slow Start and Congestion-Avoidance using Simulation Approach," January , vol. 11 , pp. 118-119, 2011.

[16] H. S. C. S. I. &. K. K. Kim, "ACM SIGOPS Operating Systems Review," A New TCP-Friendly High-Speed TCP Variant, vol. 42, no. 5, 2008.

[17] S. R. I. &. X. L. Ha, "ACM SIGOPS Operating Systems Review," new TCP-friendly high-speed TCP variant, pp. 42(5), 64-74., 2008.

[18] A. Toonk, "TCP BBR - Exploring TCP congestion control," 2020.

[19] G. Yangyang, 30 09 2021. [Online]. Available: https://info.support.huawei.com/info-finder/encyclopedia/en/MTU.html.

[20] j. Wang, "Network and Computer Aplications," vol. 53, pp. 103-114, 2015.

# Technical documentation

# List of abbreviations and symbols

*CWND* Congestion Window

ICWD Initial Congestion Window

*RTT* Round-Trip Time

*TCP* Transmission Control Protocol

*IP* Internet Protocol

*UDP* user Datagram Protocol

*Seg* Segment

*NS3* Network simulator 3

*SACK* selective Acknowledgment

*ACK* Acknowledgment

*RTO* Transmission Timeout

*MRTO* Maximum Transmission Timeout

*MSS* Maximum Segment Size

*MTU* Maximum Tansmission Unit

*Wmax*: window Maximum

*MPTCP* Multipath Transmission Control Protocol

*Rwnd* The receive window

# List of additional files in electronic submission

Additional files upload to the system include:

* Source codes : TCP-Validation in NS3
* Excel file for expriment

# List of Figures

[Figure 2.1 *Structure of the TCP Header* [1] 4](https://polslpl-my.sharepoint.com/personal/nd304723_student_polsl_pl/Documents/NDM%20thesis%20update%20final53.docx#_Toc149172405)

[Figure 2.2 *Sliding Window*Figure 2.3 *Structure of the TCP Header* [1] 4](https://polslpl-my.sharepoint.com/personal/nd304723_student_polsl_pl/Documents/NDM%20thesis%20update%20final53.docx#_Toc149172406)

[Figure 2.4 *Sliding Window* 5](#_Toc149172407)

[Figure 2.5 *segment 1* 6](https://polslpl-my.sharepoint.com/personal/nd304723_student_polsl_pl/Documents/NDM%20thesis%20update%20final53.docx#_Toc149172408)

[Figure 2.6 *Segment 2*Figure 2.7 *segment 1* 6](https://polslpl-my.sharepoint.com/personal/nd304723_student_polsl_pl/Documents/NDM%20thesis%20update%20final53.docx#_Toc149172409)

[Figure 2.8 *Segment 2* 6](https://polslpl-my.sharepoint.com/personal/nd304723_student_polsl_pl/Documents/NDM%20thesis%20update%20final53.docx#_Toc149172410)

[Figure 2.9 *segment4*Figure 2.10 *Segment 2*Figure 2.11 *segment 1* 6](https://polslpl-my.sharepoint.com/personal/nd304723_student_polsl_pl/Documents/NDM%20thesis%20update%20final53.docx#_Toc149172411)

[Figure 2.12 *Segment 2*Figure 2.13 *segment 1* 6](https://polslpl-my.sharepoint.com/personal/nd304723_student_polsl_pl/Documents/NDM%20thesis%20update%20final53.docx#_Toc149172412)

[Figure 2.14 *Segment 2* 6](#_Toc149172413)

[Figure 2.15 *segment4* 7](https://polslpl-my.sharepoint.com/personal/nd304723_student_polsl_pl/Documents/NDM%20thesis%20update%20final53.docx#_Toc149172414)

[Figure 2.16 *Receive Window* [6] 8](https://polslpl-my.sharepoint.com/personal/nd304723_student_polsl_pl/Documents/NDM%20thesis%20update%20final53.docx#_Toc149172415)

[Figure2.17 *TCP Connection Establishment (Three-way handshake)* 10](https://polslpl-my.sharepoint.com/personal/nd304723_student_polsl_pl/Documents/NDM%20thesis%20update%20final53.docx#_Toc149172416)

[Figure 2.28 *TCP Connection termination (TCP Teardown*) 11](https://polslpl-my.sharepoint.com/personal/nd304723_student_polsl_pl/Documents/NDM%20thesis%20update%20final53.docx#_Toc149172417)

[Figure 2.35 *TCP Congestion Control* [9] 11](https://polslpl-my.sharepoint.com/personal/nd304723_student_polsl_pl/Documents/NDM%20thesis%20update%20final53.docx#_Toc149172418)

[Figure 2.36 *MTU vs MSS* [19] 23](#_Toc149172419)

[Figure 3.1 *Design the Network Topology* 31](https://polslpl-my.sharepoint.com/personal/nd304723_student_polsl_pl/Documents/NDM%20thesis%20update%20final53.docx#_Toc149172420)

[Figure 3.2 *Throughput vs segment size* 35](https://polslpl-my.sharepoint.com/personal/nd304723_student_polsl_pl/Documents/NDM%20thesis%20update%20final53.docx#_Toc149172421)

[Figure 3.3 *Segment Size vs. Goodput* 37](https://polslpl-my.sharepoint.com/personal/nd304723_student_polsl_pl/Documents/NDM%20thesis%20update%20final53.docx#_Toc149172422)

[Figure 3.4 *Segment Size and drop ratio.* 39](https://polslpl-my.sharepoint.com/personal/nd304723_student_polsl_pl/Documents/NDM%20thesis%20update%20final53.docx#_Toc149172423)

[Figure 3.5 *segment size vs fairness* 41](https://polslpl-my.sharepoint.com/personal/nd304723_student_polsl_pl/Documents/NDM%20thesis%20update%20final53.docx#_Toc149172424)

[Figure 3.6 *Icwnd1* 43](https://polslpl-my.sharepoint.com/personal/nd304723_student_polsl_pl/Documents/NDM%20thesis%20update%20final53.docx#_Toc149172425)

[Figure 3.7 *Icwnd 3* 43](https://polslpl-my.sharepoint.com/personal/nd304723_student_polsl_pl/Documents/NDM%20thesis%20update%20final53.docx#_Toc149172426)

[Figure 3.8 *Icwnd 10* 44](https://polslpl-my.sharepoint.com/personal/nd304723_student_polsl_pl/Documents/NDM%20thesis%20update%20final53.docx#_Toc149172427)

[Figure 3.9 *Icwnd 20* 45](https://polslpl-my.sharepoint.com/personal/nd304723_student_polsl_pl/Documents/NDM%20thesis%20update%20final53.docx#_Toc149172428)

[Figure 3.10 *Icwnd 50* 46](https://polslpl-my.sharepoint.com/personal/nd304723_student_polsl_pl/Documents/NDM%20thesis%20update%20final53.docx#_Toc149172429)

[Figure 3.11 *Icwnd 1 vs drop ratio.* 48](https://polslpl-my.sharepoint.com/personal/nd304723_student_polsl_pl/Documents/NDM%20thesis%20update%20final53.docx#_Toc149172430)

[Figure 3.12 *Icwnd 1 vs drop ratio.* 49](https://polslpl-my.sharepoint.com/personal/nd304723_student_polsl_pl/Documents/NDM%20thesis%20update%20final53.docx#_Toc149172431)

[Figure 3.13 *ICWND 10 VS DROP RATIO* 49](https://polslpl-my.sharepoint.com/personal/nd304723_student_polsl_pl/Documents/NDM%20thesis%20update%20final53.docx#_Toc149172432)

[Figure 3.14 *ICWND 20 VS DROP RATIO* 50](#_Toc149172433)

[Figure 3.15 *ICWND 50 VS DROP RATIO* 51](https://polslpl-my.sharepoint.com/personal/nd304723_student_polsl_pl/Documents/NDM%20thesis%20update%20final53.docx#_Toc149172434)

# List of Tables

[Table 2.1 *Classification of TCP Protocols* 25](#_Toc149167943)

[Table 3.1 Influence of Segment Size[B] on Throughput[Mbps] 35](#_Toc149167944)

[Table 3.2 *Segment Size[B] vs. Goodput[Mbps]* 36](#_Toc149167945)

[Table 3.3 *Segment Size[B] vs Drop Ratio[%]* 38](#_Toc149167946)

[Table 3.4 *segment size[B] vs fairness* 40](#_Toc149167947)

[Table 3.5 *ICWND 1* 42](#_Toc149167948)

[Table 3.6 *ICWND 3* 43](#_Toc149167949)

[Table 3.7 *ICWND 10* 44](#_Toc149167950)

[Table 3.8 *ICWND 20* 45](#_Toc149167951)

[Table 3.9 *ICWND 50* 46](#_Toc149167952)

[Table 3.10 *Icwnd 1 vs drop ratio.* 47](#_Toc149167953)

[Table 3.11 *ICWND 3 VS DROP RATIO* 48](#_Toc149167954)

[Table 3.12 *ICWND 10 VS DROP RATIO* 49](#_Toc149167955)

[Table 3.13 *ICWND 20 VS DROP RATIO* 50](#_Toc149167956)

[Table 3.14 *ICWND 50 VS DROP RATIO* 51](#_Toc149167957)