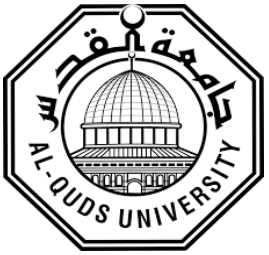
Deanship of Graduate Studies

AL-QUDS University



Robust Dynamic Congestion Control Protocol for Mobile Networks (TCP DCM+)

Derar Sameeh Abdel-Aziz Khader

MSc. Thesis

Jerusalem – Palestine

1441 - 2020

Robust Dynamic Congestion Control Protocol for Mobile Networks (TCP DCM+)

Prepared by:

Derar Sameeh Abdel-Aziz Khader

BSc. Computer Engineering – Philadelphia University – Jordan, 2005

Supervisor: Dr. Rushdi Hamamreh

A thesis submitted to Faculty of Engineering, Al-Quds University in Partial Fulfillment of the requirements for the degree of Master of Electronic and Computer Engineering.

1441 - 2020

AL-QUDS University

Deanship of Graduate Studies

Electronic and Computer Engineering

Thesis Approval

Robust Dynamic Congestion Control Protocol for Mobile Networks

(TCP DCM+)

Prepared by: Derar Sameeh Abdel-Aziz Khader

Registration No.: 21311880

Supervisor: Dr. Rushdi Hamamreh

Master Thesis submitted and accepted. Date: 21/12/2019

The names and signatures of the examining committee members are as follows:

1. Head of Committee: Dr. Rushdi Hamamreh. Signature:
2. Internal Examiner: Dr. Ali Jamoos Signature:
3. External Examiner: Dr. Mousa Farajallah Signature:

Jerusalem – Palestine

1441 - 2019

Dedication

This thesis is dedicated to my beloved parents, who have raised me to be the person I am today.

A special thanks to my best friend and brother (Salah),

To all other brothers and sisters (Iman, Rajaa and Inas), who always encouraged me to continue my way to achieve this degree.

To my lovely wife 'Kholoud', who always stands to my side in all good and hard times.

To my sons, Barhoom and Abood, who are the pleasure of my life.

Thank you all.

Declaration

I certify that this thesis submitted for the degree of Master, is the result of my own research, except where otherwise acknowledged, and that this thesis has not been submitted for higher degree to any other university or institution.

Signed: .....

Derar Sameeh Abdel-Aziz Khader

Date: 21/12/2019

Acknowledgements

At first, all the thank is to Allah, the only god, who gave me the time and health to finish this thesis.

Also, I am very thankful to the best teacher and supervisor I got ever, Dr. Rushdi Hamamreh, who believed in me and gave me the time to discuss and to try new ideas. Without his encouragements, this work never comes to the end.

Also, I am deeply thankful to all my teachers at AL-QUDS university, who always do their best.

Abstract

Data networks are considered as a critical corner of data transmission between the different hosts wherever they exist. In the last few years, the wireless and mobile networks become more important for daily use and are their spread is increasing for personal and commercial use. The main difference between wired and wireless networks is the large number of lost packets during the data transmission. The packet losses are a result of errors on the data transmission channel. These errors are due to external noise, interference and mobility of the wireless devices that results in deep fading. The mentioned problems earlier are the reasons that the throughput of wireless, mobile and mobile adhoc networks is less than wired networks, which does not suffer such problems.

Old traditional transmission control protocols like (Standard TCP) behave extremely hard when they detect any data packet losses. They drop the congestion window to the half though the transmission channel capacity is not exhausted. This high drop results in low throughput, hence longer time to finish the transmission.

Most traditional TCP protocols lack the use of appropriate techniques to estimate the available channel capacity, which are known as bandwidth estimation (BWE) techniques. In 2004, TCP Westwood+ protocol proposed a technique for estimating the available channel capacity. It uses a first-order low-pass filter to find the available bandwidth. TCP Westwood+ has largely improved the throughput of TCP connections, however, the problem of window drops is still existing, which makes it less appropriate for use in networks, that include mobility, i.e. MANETs. Hence, it is desired to modify the TCP protocol behavior to eliminate these drops, which are the results of congestion events or

channel problems. If the congestion events are eliminated, then we can detect the times at which the transmission channel problems occur.

The proposed approach in this thesis is called TCP DCM+. It is the abbreviation for “Dynamic Congestion Control for Wireless and Mobile Networks”. The transfer of data with different sizes has been simulated with different packet error rates, which should simulate the existence of wireless channel for large packet error rates (1e-3 to 5e-2).

We executed hundreds of simulations for cases with different parameters like error rates, MTU sizes, bandwidth of both bottleneck (link) and destination (access), protocol type and the size of sent data. We found that DCM+ performs better than the other approaches, especially if the error rates are large. We used the usual performance metrics like throughput, average delay and packet losses to measure how well our approach performs. Additionally, we introduced two new metrics to measure the total time needed to finish the transmission, and also to measure the robustness and stability of the transmission. Our conclusion is, that DCM+ is minimizing congestion events, hence, transmits data much faster, shows stable behavior and is highly robust compared with other approaches.

بروتوكول التحكم الديناميكي في نقل البيانات في الشبكات اللاسلكية والمتنقلة

اعداد الطالب: ضرار سميح عبد العزيز خضر
اشراف: د. رشدي حمامرة

الملخص

تعتبر شبكات المعلومات ركنا أساسيا في نقل البيانات، وأصبحت الشبكات اللاسلكية (Wireless Networks) في السنوات الأخيرة تستحوذ على حيز كبير في إدارة الأعمال والاستعمال الشخصي. غير أن هنالك فرق مهم بين الشبكات السلكية واللاسلكية وهو أن الشبكات اللاسلكية معرضة بشكل أكبر لحدوث أخطاء على البيانات المرسلة، وهذه تحصل على قناة الارسال (Transmission Channel) بسبب التشويش الخارجي (External Noise)، أو بسبب الحركة (Mobility) والتنقل للأجهزة، وقد تكون بسبب وجود عوائق فيزيائية (Physical Barriers) تمنع الاتصال المباشر بين طرفي القناة. جميع ما ذكر يتسبب في انخفاض سرعة نقل البيانات في الشبكات اللاسلكية أو المتنقلة أو الشبكات المخصصة لأغراض معينة (MANET Networks) وهو ما لا تعاني منه الشبكات السلكية.

في هذه الأطروحة، تم اقتراح وتصميم واختبار تقنية جديدة تسمى "TCP DCM+". هذه الطريقة تقوم على اعتماد تقنية التحديد المسبق للسعة المتاحة للإرسال (Bandwidth Estimation)، وذلك من خلال اعتماد الخوارزمية الموجودة في بروتوكول "TCP Westwood+"، ثم دمجها مع برمجية أُخرى لقياس حجم الفراغ المتاح في ذاكرة المُوجِّه (Router Buffer) بين طرفي الارسال. قياس الحجم المتاح يتم من خلال مقارنة زمن الذهاب والإياب (Round Trip Times) لعملتي ارسال متتاليتين، فاذا كانت نسبة الزمن السابق أكبر من الزمن الحالي فأنه يعني أن ذاكرة الراوتر تفرغ بسرعه، والا فأنها تمتلئ. اعتمادا على هذه القيمة يكون الارسال بشكل سريع جدا أو بطيء وبحذر، وهو ما يعتمد على الحجم السابق لنافذة الارسال (Window Size). فإذا كان حجم نافذة الارسال السابقة التي وصلت للمستقبل بنجاح أقل من عتبة البدء البطيء (slow- start threshold) فعندها يتم ارسال كمية كبيرة جدا من البيانات دفعة واحدة (Burst) بحجم المكان الفارغ في الراوتر، والا يتم الارسال بشكل بطيء

يتلاءم مع حجم الفراغ في الراوتر، وبحيث لا يحدث تصادمات مع مرسل آخر على نفس ممر البيانات (Data Path).

من خلال المحاكاة لمئات الحالات المختلفة، والتي استخدمت فيها متغيرات متعددة مثل نسبة ضياع الحزم، حجم الحزمة، سعة قناة الارسال، البروتوكول المستخدم وحجم البيانات المرسلة، فان النتيجة كانت أن الطريقة المقترحة من طرفنا هي الأسرع، وأنها ذات قدرة عالية جدا على التأقلم (Robustness) مع الظروف القاسية للإرسال، كما انها لا تسبب أي اختناقات على قناة الارسال بسبب تصميمها الذي اعتمد على معرفة زمن الذهاب والإياب للحزم، وإذا ما كان هناك امكانية للإرسال بسرعة كبيرة أم قليلة.

Table of Contents

Table of Figures

List of Tables

List of Algorithms

List of Equations

Acronyms and Abbreviations

AAF	Anti-aliasing filter
ACK	Acknowledgement
AIADD	Additive Increase-ADaptive Decrease
AIMD	Additive Increase-Multiplicative Decrease
AQM	Active Queue Management
CA Phase	Congestion Avoidance Phase
CDMA	Code-division multiple access
CSMA/CA	Carrier-Sense Multiple Access/ Collision Avoidance
CSMA/CD	Carrier-Sense Multiple Access/ Collision Detection
CTT	Complete Transmission Time

CWND	Congestion Windows
DCM	Dynamic Congestion Model
DCM+	Dynamic Congestion Control for Wireless and Mobile Systems
DUPACK	Duplicate Acknowledgement
LAN	Local Area Network
MAN	Metropolitan Area Network
MANET	Mobile Adhoc Network
MSS	Maximum Segment Size
MTU	Maximum Transmission Unit
NAI	Normalized Advancing Index
PDR	Packet Delivery Ratio
RED	Random Early (Discard / Drop)
RTO	Retransmission Timeout
RTT	Round Trip Time
SS	Slow Start
SSTHRESH	Slow-Strat Threshold
TCP	Transmission Control Protocol
UMTS	Universal Mobile Telecommunications Service
WAN	Wide Area Network

Chapter One: Introduction

Content

Introduction

1.1 Overview

1.1.1 Classification of Computer Networks

Computer network is a system of interconnected nodes such as PCs, laptops, servers, mobile phones and peripherals such as printers. Information sharing is then enabled among them during the interconnection. We may connect the different nodes via wired or wireless media [1]. We can classify the computer networks according to the following factors:

1- Area
2- Inter-connectivity
3- Administration
4- Architecture

1- Area

From the perspective of area, a network can belong to any of the following categories:

- It may be spanned across a few meters.
- It may connect the devices in a building,
- It may be spanned across the whole town.
- It may connect many towns or provinces.
- It could also connect the whole planet like the internet.

2- Inter-Connectivity

The components of the network can be connected differently. Connectedness could use a logical or a physical topology, or both approaches.

- Network *mesh*, where every single device can be connected to every other device on network.
- *Bus* structure, where all devices can be connected to a single medium (switch), but geographically disconnected.
- *Linear* structure, where each device is connected to its left and right peers only, like in ring structure.
- *Star*-like structure, where all devices are connected together with a single device.
- *Hybrid* structure, where all devices are connected arbitrarily using all previous ways.

3- Administration

A network can be private or public. Private networks, from an administrator's point of view, belongs a single autonomous system, hence cannot be accessed outside its physical or logical domain. A public network on the other side, can be accessed by all, depending on the access right given by the administrator.

4- Architecture

Depending on their architecture, networks can be differentiated into various types such as client-Server, peer-to-peer or hybrid,

- Client-Server architecture: one or more nodes are acting as server. The clients make requests to the server(s). Servers take and process the request(s), and send responses back to clients. An example is the banking system.

- Point-to-Point architecture: Two systems are connected together. They could be geographically separated, but the communication is restricted to these devices (peers), like in torrent traffic.
- Hybrid architecture, which involves architectures of the above types.

1.1.2 Types of Data Networks

We distinguish networks based on their geographical span. A network can be as small as 50 cm, the distance between the mobile phone and its Bluetooth headphone. Also, it could be as large as the internet itself, covering the whole world. Following types of networks are distinguished:

1- Personal Area Network (PAN)

PAN is the smallest network, which is very personal to the user. This type includes Bluetooth- or infrared-enabled devices. This type has a range up to 10 meters. PAN may include wireless keyboard, mouse, headphones, printers and TV remotes. Piconet is an example for Bluetooth-enabled PAN. It may contain up to 8 devices connected together in a master-slave (client-server) architecture.

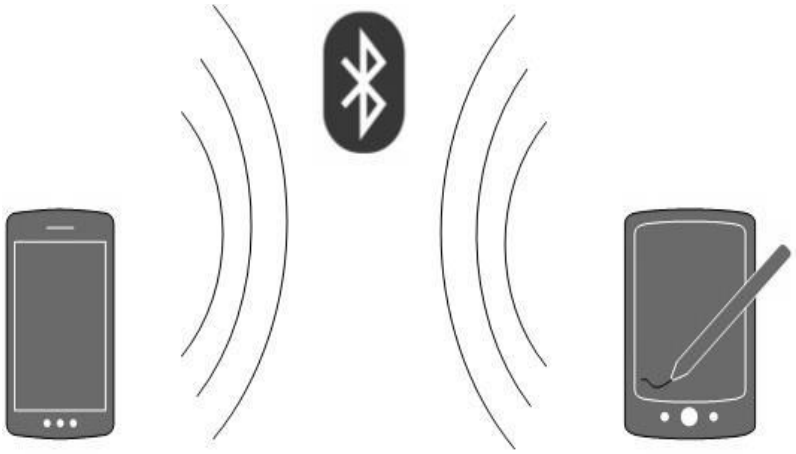


Figure 1-1: Bluetooth

2- Local Area Network (LAN)

If the network is spanned inside a building and operated under a single administrative system, then it is called “Local Area Network”, or short (LAN). LANs cover the offices in a company, schools, colleges or universities. Depending on the class used in the network design, or the number of bits in the network mask, the number of connected nodes in the LAN may vary from as least as two to as much as millions. LANs provide a way for sharing the resources between the end users, i.e. printers, scanners, public data on file servers, etc. Figure 1.2 shows an example for a LAN.

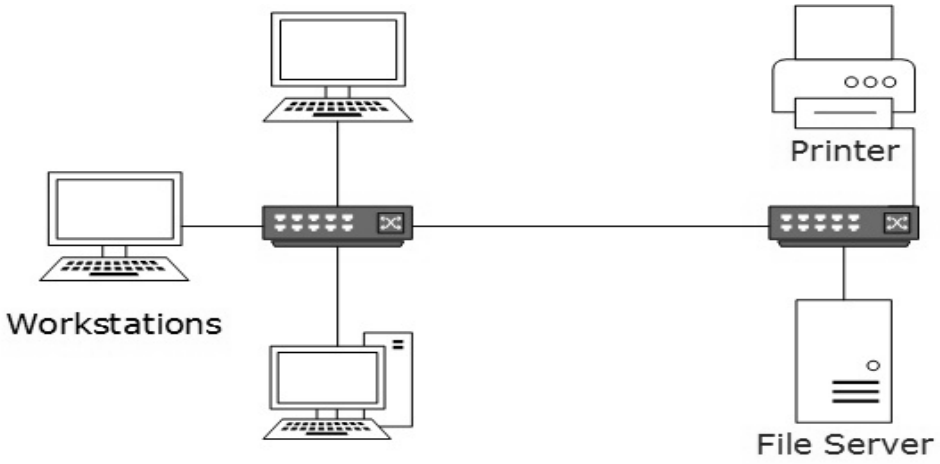


Figure 1-2: Local Area Network (LAN)

3- Metropolitan Area Network (MAN)

The Metropolitan Area Network (MAN) is designed to connect different areas of a city. In Europe and USA, MAN network is used for cable TV services. It can be in one or more of the following forms: Ethernet, Token-ring, ATM, or Fiber Distributed Data Interface (FDDI). MAN is a service, that is mostly provided by ISPs. This service enables users to expand their LAN, i.e., MAN can help the companies to connect all of its offices in a city.

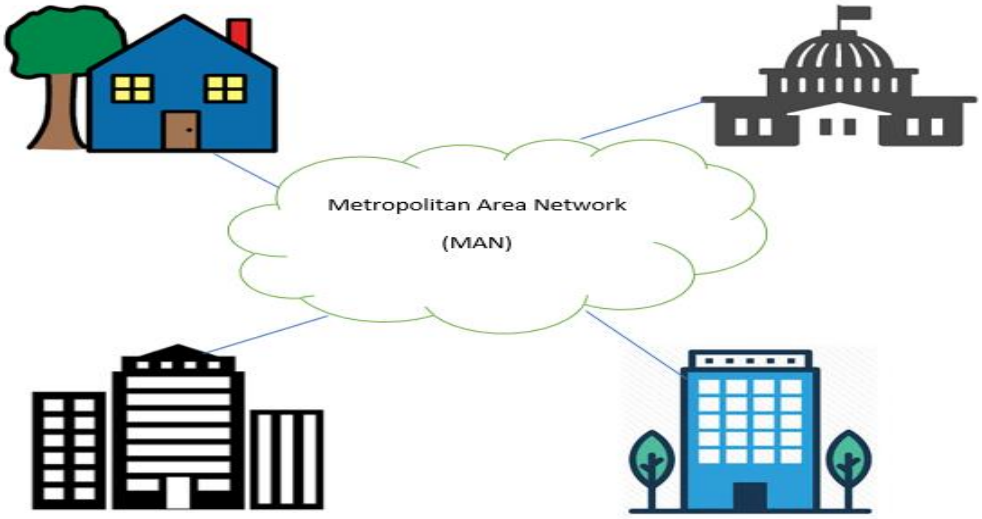


Figure 1-3: Metropolitan Area Network (MAN) [2a]

MANs mostly use fiber cables for their high-capacity and high-speed data traffic. MANs lie between LANs and WANs (Wide Area Networks).

4- Wide Area Network (WAN)

WANs use routers to cover large areas. This coverage may be from small provinces up to a whole country or a continent. Figure 1-4 shows an example of a WAN.

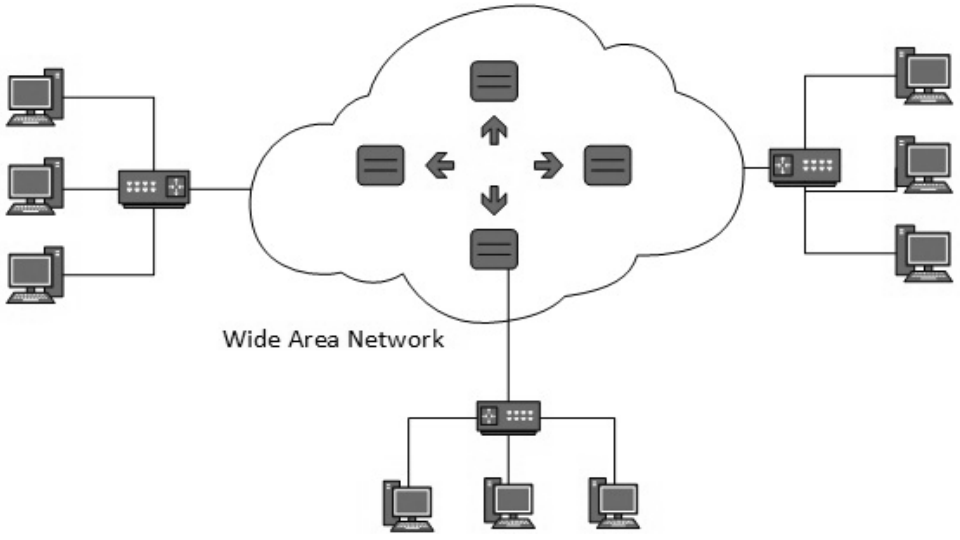


Figure 1-4: Wide Area Network (WAN)

Telecommunication networks are generally WANs, which use routers and satellites for their data transfer. WANs equipment are very expensive, hence WAN services cost more than those of LAN, when using the same speed. The WANs are equipped with high-speed devices to build the final backbone, which connects the remote LANs.

5- Internetwork

Internetwork is a network of networks. It is also called the internet. Internet is the planets' largest network, that could be shown as a living creature. The interruption of internet in one area on the planet may affect the whole planet.

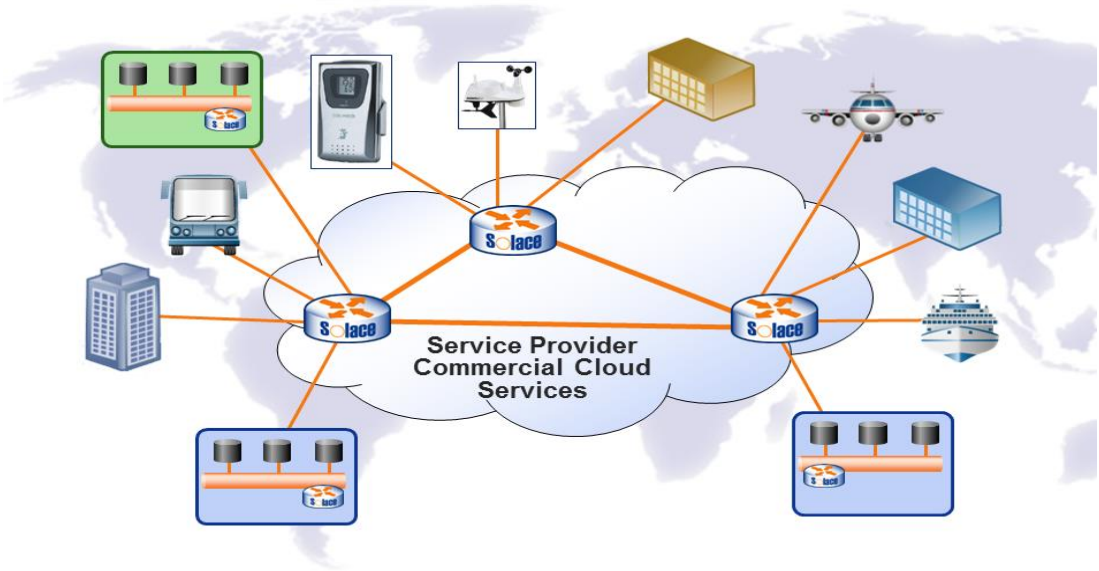


Figure 1-5: Internet [2b]

Figure 1-5 depicts the complexity of internet. The internet is made by a huge number of connections between smaller WANs. Internet mainly uses TCP/IP protocol suite for data transfer between the hosts. The IP protocol is used for addressing and routing. At a huge level, internet can be considered as a

client-server model. Internet backbone is built from very high-speed fiber cables. They connect the various continents via fiber cables, which may be laid on the ground of the sea. They are known as “Submarine Communication Cables”.

1.2 Motivation

Data transmission refers to the movement of data in form of bits between two or more digital devices. It is classified as serial and parallel communication like in figure 1-6. This transfer of data takes place via some form of transmission media (i.e., coaxial cable, fiber optics, wireless channel, etc.). Modern economies depend on data transmission as a business [3]. Telephone and mobile companies make a huge profit from the telecommunication services. These services are enhanced and extended to include emails, SMSs, data sharing, internet browsing, etc., which require large bandwidths.

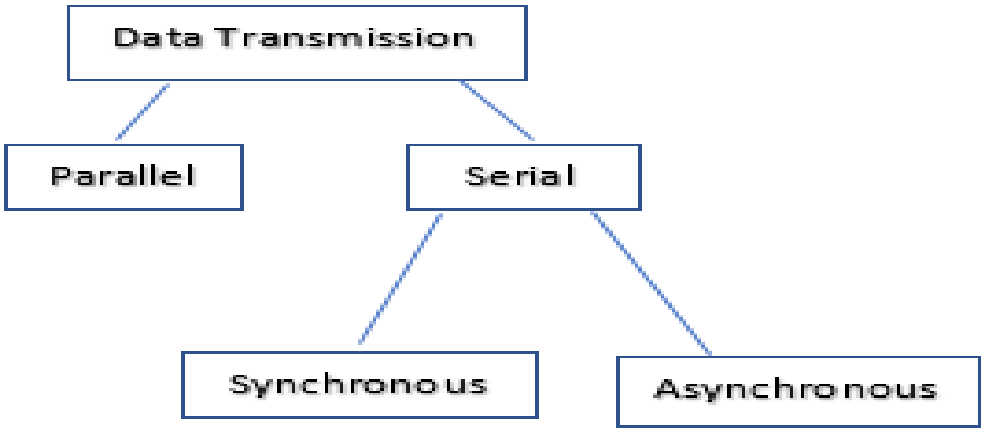


Figure 1-6: Types of Data Transmission

The quick delivery and robustness of the transmission technique is crucial [4]. Old standard TCP techniques (i.e. TCP Reno, TCP NewReno) were invented to control when and how much data can be sent per time interval [5]. These approaches, however, have been found to be less appropriate for wireless

networks. Table 1-1 depicts the main differences between wired and wireless networks.

Wireless	Wired
Physical configuration needed	No physical configuration required
Packet error rate is very high	Packet error rate is very low
Large delays	Small Delays
Security is low	Security is high
Low data rate → low speed	High data rate → High speed

Table 1-1: Wireless vs. Wired Networks

According to the underlying structure of the existing wireless network, we differentiate 2 main types:

- Mobile (cellular) networks,
- Mobile Adhoc Networks (MANETs).

In Table 1-2, we present the main differences between these types.

Cellular Networks	MANETs
Infrastructure required	Infrastructure not existing
Locations of cell sites fixed	No fixed locations
Long planning before launch required	MANETs automatically adapt to network changes
Setup cost too high	Setup cost low
Setup time to build the cellar network high	Less time is needed to form the MANET.

Table 1-2: Cellular Networks vs. MANETs

In wireless networks, that are static (not moving), the channel conditions are varying slightly. On the other side, in MANETs and mobile networks, the channel conditions are varying largely [6a] and quickly, which cause the channel capacity to change continuously [6b]. Hence, it is desired to use a technique, that knows how much channel capacity is available before

transmitting data. Also, it should be dynamic in its speed. The technique should allow the sender to transmit

data in bursts without causing new congestions. TCP DCM+ is a new approach, that is targeting these aims in order to achieve extremely high throughput by the existence of tough channel conditions like high error rates.

1.3 Problem Statements

A decrease in throughput and response speed of a network connectivity is known as “network degradation” [7]. If this kind of deterioration exists within a network, it is important to analyze it and to do a diagnosis. Degradation causes of network may include propagation delays, which involve faulty network devices, congestion, routing problems or transfer of large data files across the IT infrastructure. Insufficient memory and low processing capacity of end nodes may also be other reasons for the delays. There are also other forms of degradation, that may occur as a result of malware or spyware in the network.

While problems with individual hardware devices don’t usually affect the functionality throughout the entire network, other problems can be network-wide. For example, congestion or problems of fragmentation of data packets can affect network performance. To anticipate and handle some kinds of network degradation like congestion or interruption of connection, network engineers may consider fault-tolerant designs, where systems may be designed to operate well even under extreme conditions. This prevents various kinds of natural degradation from causing system failure or interrupting core network services. In this thesis, a new technique will be presented to improve performance and to prevent network congestion [8][9] [10a].

1.4 Objectives of This Thesis

The aim of this thesis is to design a new technique, that will be able to use the accurate estimation of the channel capacity [10b], and to send data accordingly in a dynamic way depending on the channel conditions like packet error rates, bottleneck and destination bandwidths. TCP DCM+ should be fair in sharing the channel capacity with other TCP sources, stable and robust in delivering data to the destination. Also, the throughput and transmission time should be improved. Our simulations show, that we have realized our aims.

1.5 Thesis Contributions

We have designed an end-to-end approach, that extremely improves the TCP transmissions by modifying the behavior in the congestion avoidance phase in the TCP sender. The following modifications have been performed on the sender-side code of the files "rtt-estimator.xx" and "tcp-westwood.xx" and "TCP-Variants-Comparison.cc" to attain the desired behavior:

1- keeping the old value of round-trip-time as (RTT_{old}),
2- adding a new member function called "Congestion Avoidance" to the code, which should emulate the procedure when entering a congestion phase. This behavior overwrites the behavior in NewReno, when entering congestion avoidance phase.
3- A new parameter called ***rateCA*** has been introduced to emulate the free size of the TCP buffer in the intermediate node at the time of ACK segment reception.
4- New metrics have been introduced to measure the stability, robustness and transmission time of TCP DCM+ in depth.

1.6 Thesis Structure

The rest of this thesis is structured as follows:

- **Chapter Two: Congestion Control in Data Networks**

  This chapter explains the reasons of congestion in data networks. Also, some strategies to handle congestion in mobile, wireless and wired networks have been discussed.

- **Chapter Three: TCP Protocols of Mobile and Wireless Networks**

  In this chapter, we explain some techniques for managing network congestion. We discuss and compare the working of the following TCP Protocols: Westwood, Westwood+, NewReno, BIC, Hybla and Ledbat.

- **Chapter Four: Proposed Approach: DCM+**

  We propose here our new protocol (DCM+) as additive-increase/adaptive decrease approach. The building blocks of this protocol is explained in this chapter.

- **Chapter Five: Simulation Results and Analysis**

  We present in this chapter the simulations using different parameters for different cases and show the advantages of this techniques over other approaches.

- **Chapter Six: Conclusion and Future Work**

  We conclude the thesis in this chapter and make suggestions for possible future research using DCM+.

Chapter Two: Congestion Control in Data Networks

Content

2.1 Introduction

According to the queueing theory, congestion in data networks is the reduced quality of service (QoS), that could occur, if a network node or link is carrying more data than it can handle [11a]. Similar to a road congestion, effects like the followings could happen: queueing delay, packet loss or the blocking of new connections. As a consequence of network congestion, an additional increase of the offered load leads either only to a small increase or even to a decrease in network throughput [11b].

Some network protocols use aggressive retransmissions to compensate for packet loss due to congestion, but this can lead to more congestion, and even after the initial load has been reduced to a level that would not normally have caused any congestion. Examples of this type are TCP Reno and TCP NewReno.

After 1986, data networks, that use TCP protocol for their data communication, started to use TCP congestion control algorithms [31b] and congestion avoidance techniques to avoid a throughput collapse [12][13]. This collapse is called “window drop”. When a congestion is detected, the congestion window or (***cwnd***) is dropped to a predetermined value. ***Cwnd*** is an algorithm parameter, that saves the last value of sent data. To detect a network congestion, different techniques use different indicators, such as packet loss or 3

consecutive duplicate acknowledgement (also known as DUPACK) packets. Packet loss may be done intentionally, i.e. by routers to empty its buffer and hence to mitigate the congestion. It also can be lost as a reason of bad media like a noisy wireless channel.

Known techniques to mitigate congestion are: exponential back-off is used in protocols such as ***CSMA/CA*** in the wireless standard IEEE 802.11. The original ethernet uses ***CSMA/CD***. Reduction of the congestion window is used in most TCP protocols. Network appliances like routers and switches use fair-queueing, Random Early Discard (RED) [14] or Active Queue Management (AQM) [15] technique. Other techniques that address congestion include priority schemes, which transmit some packets with higher priority first. Similarly is the explicit allocation of network resources to specific flows through the use of admission control [16].

2.2 Congestion Control in Data Networks

Congestion control techniques are those algorithms implemented in the operating systems in order to control or prevent congestion [14]-[16]. Generally, congestion control techniques can be classified into two groups:

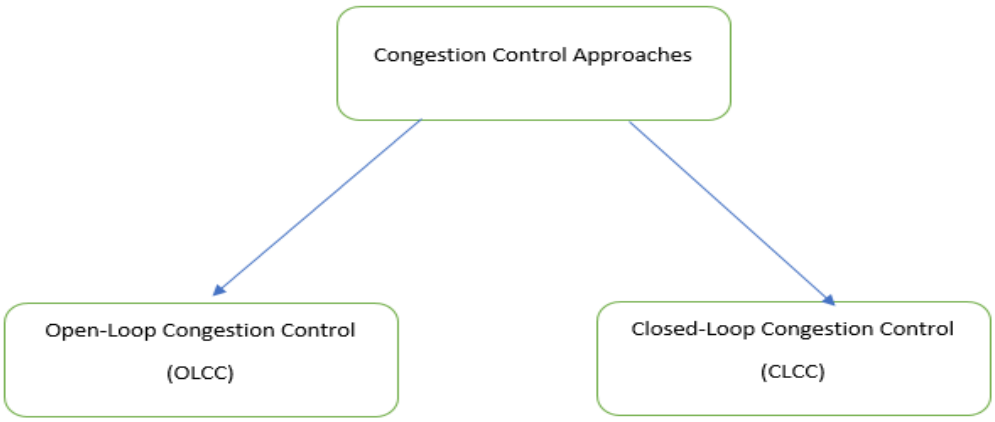


Figure 2-1: TCP Congestion Control Approaches

- **Open Loop Congestion Control (OLCC)**

When OLCC is applied, then congestions are prevented before they happen. This technique can be applied either by the sender or the receiver. This technique make use of either of the following methods: retransmission timers, selective-repeat window, partially discarding of packets, acknowledgment packets and the policy to admit/deny a connection.

- **Closed Loop Congestion Control (CLCC)**

In this case, congestion control algorithms are used to manage or alleviate congestions after they happens. Several techniques exist, and they can be used by different protocols. Techniques of this type are backpressure, choke packet, implicit and explicit Signaling.

Congestion control is a vital process for data networks, especially those, that rely mainly on TCP traffic. It has a central role for achieving high performance and throughput via managing the congestions, which cause drops in the windows size of the transmission. As a result, this prevents the collapse of the global network like the internet [17][18]. Since 1986, many protocols have been proposed and implemented for controlling data transmission between hosts. Old Tahoe [19] is the earliest variant of TCP. It implements two algorithms called *slow start* (SS) and *congestion avoidance* (CA) to update the congestion window (*cwnd*). From the point of view of a sender, the algorithms of old Tahoe are:

- **Slow start (*SS*):**

- at the transmission start, the size of congestion window is 1. This means TCP can send only one packet until it receives an acknowledgement.
- When ACK is received, the congestion window increases to two.
- Upon the arrival of every new ACK, the sender increases its congestion window by one.

The congestion window in this phase increases exponentially. So, on the arrival of a new ACK, the window follows the equation:

$$cwnd \; + = \; MSS; \tag{2.1}$$

- **Congestion Avoidance (*CA*)**

  - continues slow start phase until it reaches a certain threshold, or a packet loss occurs (*congestion indicator*)
  - on *congestion indicator*: TCP enters the *CA* phase: *cwnd* increases from ‘n’ to ‘n+1’ only when it has received ‘n’ new ACKs.

The window grows in this phase linearly. The rate of growth of the window slows down, because this is the stage where TCP is susceptible to packet loss. The equation used here is:

$$cwnd \; += \; (SMSS * SMSS) \; / \; cwnd \tag{2.2}$$

TCP NewReno, on the other side, is a TCP variant of the old days of wired networks [20][21][22]. NewReno though has some drawbacks and limitations, especially in both wireless and mixed (wired/wireless) networks [23]. Another limitation of TCP NewReno is its little support for mobility [24]. We conclude, that NewReno has little chances in mobile and wireless

environments like Wi-Fi networks and Mobile Adhoc Networks (***MANETs***). TCP NewReno has been implemented in the TCP protocol stack of different operating systems.

Recently, newer TCP variants like TCP Westwood+, BIC, CUBIC, HighSpeed, Scalable, Hybla and Ledbat are available in modern operating systems like Linux [25][26][27]. They are better appropriate for wireless networks. TCP Ledbat, for example, is implemented under MS Windows Server 2019, and also in MS Windows 10. Figure 2-2 depicts the network congestion, while figure 2-3 shows the first proposed congestion control algorithm known as “TCP Tahoe”

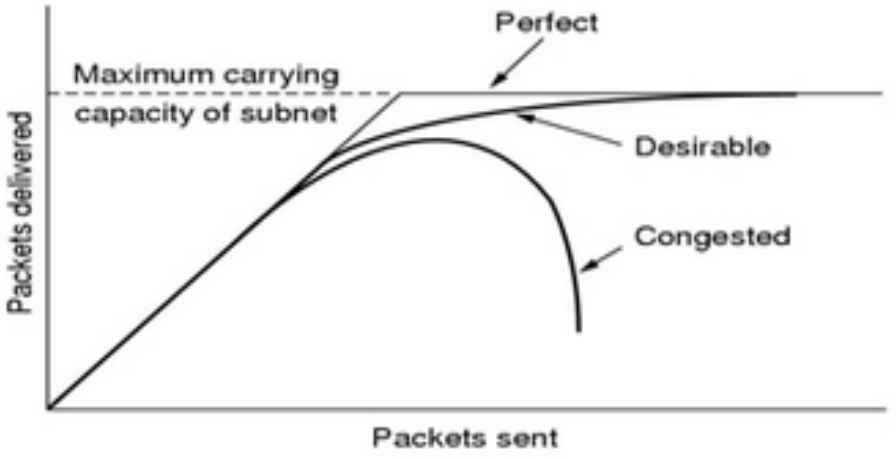


Figure 2-2: Principle of Network Congestion [48 a]

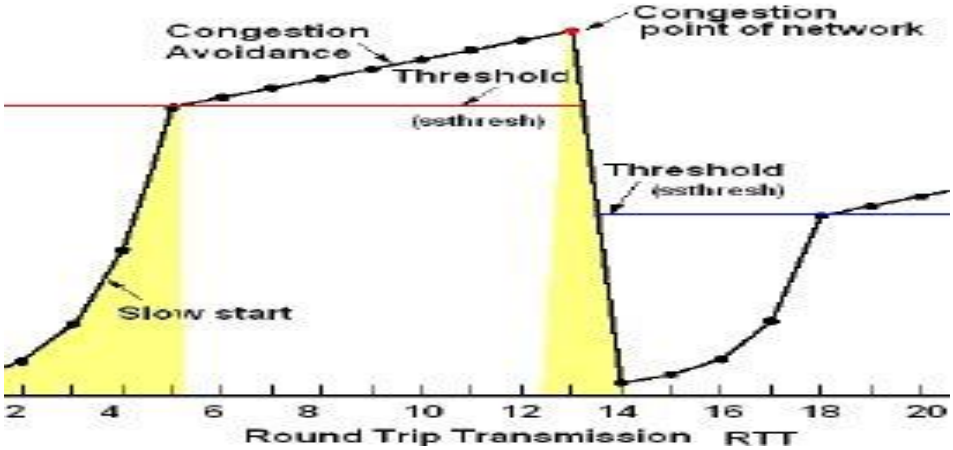


Figure 2-3: Congestion Management in Standard TCP Protocols [48 b]

2.3 TCP Congestion Control Strategies

TCP is a connection-oriented protocol, that is also very reliable. It uses sequence numbers to each byte sent in segment. It also provides the feedback mechanism, also known as "signaling". It means, that when a host receives a packet, it is bound to ACK it, having the next sequence number expected. When a TCP Server crashes in the middle of a communication and restarts its transmission, it sends a broadcast to all its hosts. The hosts can then send the last data segment which was never unacknowledged and carry onwards. According to TCP, congestion occurs if huge amount of data is fed into a network, which is not capable of handling it. In this case, the mechanism used to handle this problem is the "congestion window". The value of congestion window will be increased or decreased depending on the network status. Different algorithms use different procedures to increase/decrease the value of *cwnd*. The main strategies used in TCP are:

- Additive increase, Multiplicative Decrease (AIMD),
- Additive increase, Adaptive Decrease (AIADD)
- Multiplicative increase, Multiplicative Decrease (MIMD)

- **AIMD**

This approach represents a feedback control algorithm, which makes the network to a closed loop system [29][30]. It is the working principle of TCP NewReno. AIMD combines linear growth of *cwnd* with an exponential reduction when a congestion is detected. When multiple flows using this approach share the same link, then they will eventually converge to use

equal amounts of bandwidth, which is known as “fair share”. The algorithm used in AIMD is:

$$w(t+1) = \begin{cases} w(t) + a, & nomal, \\ w(t) * b, & congested, \end{cases} \quad (2.3)$$

where, “a and b” are constants of additive increase and multiplicative decrease, respectively. The variable “t” is the time point of the ACK arrival.

- **AIADD**

The main concept of this technique is to adapt the reduction of the window to the available bandwidth at the time the congestion has occurred [30]. TCP Westwood+, that lies at the heart of our proposed algorithm, DCM+, was the first algorithm of this technique proposed in 2004 [31a] [31b][32].

- **MIMD**

This approach shows instable behavior of congestion window. It does not converge to a fair-share of the network bandwidth. Hence, it is not practical and not used.

2.4 Congestion Control in Mobile and Wireless Networks

Over the years, congestion control in mobile and wireless networks has been investigated [33a]. Many advanced schemes and techniques have been developed, all with the aim of improving the performance in these networks. As the mobile and wireless technologies are rapidly growing and implemented, it is important to solve the problems caused by the congestion [33b].

The protocol (TCP) is the most used protocol in today’s Internet. It supports reliable transport of data by establishing a connection between the

transmitting and receiving nodes. The transmitter starts a timeout mechanism, when it starts sending a packet to a receiver. The transmitter constantly tracks the round-trip times (RTTs) for its packets as a means to determine the appropriate timeout period. At the receiver, each received packet is acknowledged implicitly or explicitly to the transmitter. If the transmitter does not get an ACK packet for a given segment, and the corresponding timeout period has expired, then the packet is deemed to be lost, and subject to retransmission. A congestion window with dynamically adjusted *cwnd* size is used by the TCP protocol to regulate the traffic flow from the transmitter to the receiver [34].

Although TCP was initially designed and optimized for wired networks, the growing popularity of wireless data applications has led mobile wireless networks such as CDMA2000 and UMTS networks to extend TCP to wireless communications as well. It was the main objective of TCP to efficiently use the available bandwidth and to avoid overloading the network, which may result in packet losses. The used strategy aims at appropriately throttle the senders' transmission rates.

2.5 Summary

The congestion in the network is considered to be the main reason for packet losses. Consequently, the performance of TCP connections is often unsatisfactory when wireless networks are used. It requires various improvement techniques. Bad quality of radio links is the key factor for unsatisfactory performance. This quality in wireless networks can fluctuate greatly in time due to channel fading, noise and user mobility.

Chapter Three: TCP Protocols for Mobile and Wireless Networks

Content

3.1 TCP NewReno

TCP NewReno is also known as RFC 6582. It is an improvement on standard TCP Reno. In particular, NewReno modifies Reno's ***Fast-Retransmit*** and ***Fast-Recovery*** algorithms. It improves the performance of handling loss of multiple segments in a single round-trip time (RTT), when no SACK is used.

As NewReno is able to detect multiple packet losses, it is much more efficient than Reno. New-Reno, like Reno, enters the ***fast-retransmit*** phase when it receives multiple duplicate packets (3 DUPACKs). They however differ in, that NewReno doesn't exit ***fast-recovery*** until all outstanding data is acknowledged, because ***fast-recovery*** phase allows for multiple re-transmissions in NewReno.

The ***fast-recovery*** phase proceeds as in Reno, however, when a fresh ACK is received then there are two cases:

1- If the received ACK acknowledges all the segments, which were outstanding when we entered ***fast-recovery,*** then it exits fast recovery and sets ***cwnd*** to ***ssthresh*** and continues congestion avoidance like Tahoe.

2- If the ACK is a partial ACK, then it deduces that the next segment in line was lost, it re-transmits that segment, and sets the number of duplicate ACKs to zero, and it exits ***fast-recovery***. New-Reno has a problem, that it takes one RTT to detect each packet loss. When the ACK for the first retransmitted segment is received, only then we can deduce the other losses [20].

3.2 TCP Westwood/ Westwood+

3.2.1 TCP Westwood

TCP Westwood is end-to-end protocol, that was first proposed in 2001 [33] [35a]. It belongs to the paradigm “additive increase/ adaptive decrease”, or short (***AIADD***). It uses the implicit feedback for the end-to-end measurement of the bandwidth available along a TCP connection. At the TCP sender, low-pass filter is applied to filter the returning rate of acknowledgments in order to estimate the available bandwidth. After a congestion episode, that is after a timeout or 3 duplicate acknowledgments, the estimated bandwidth is used to properly set the congestion window (***cwnd***) and the slow start threshold (***ssthresh***). After a congestion, the new states of congestion window and slow-start threshold are consistent with the real network capacity.

The main principle of TCP westwood is its mechanism of faster recovery. This phase is designed to avoid the large reduction of the congestion window after a congestion, by taking into account the end-to-end estimation of available bandwidth, which enable the sender to recover faster after a loss event. This very appropriate, especially over connections with large round-trip times (RTT), or when running over wireless links where sporadic losses are due to unreliable links rather than congestion [35b]. The proposed modifications follow the end-to-end design principle of TCP. They require only slight modifications at the sender side and are backward-compatible. The feedback is merely end-to-end and does not rely upon explicit information from intermediate nodes or routers at the network level.

When an ACK is received by the sender of TCP westwood, then it conveys the information that an amount of data corresponding to a specific transmitted packet was delivered to the receiver. Averaging the delivered data count over time yields a fair estimation of the bandwidth currently used by the source, in case that, the transmission process was not affected by losses.

When TCP source receives 3 duplicate ACKs (DUPACKs), indicating an out-of-sequence reception, they should also count toward the bandwidth estimate, and a new estimate should be computed right after they are received. As the source is in no position to tell which segment triggered the DUPACK transmission, it is unable to update the data count by the size of that segment.

End-to-End Bandwidth Estimation

Before a congestion event, the used bandwidth is less or equal to the available bandwidth of the network. So, TCP source can still probe the network capacity. Immediately after a congestion episode, the bandwidth used by the connection is exactly equal to the maximum bandwidth available to that connection. A congestion event like packet loss is a clear indicator, that the buffer of intermediate node (router) is fully saturated. It is known, that whenever the low-frequency input traffic rate exceeds the link capacity, then a congestion event occurs. Hence, low-pass filter is needed to calculate low-frequency components of the available bandwidth. The used filter is obtained by discretizing a first-order low-pass filter using the trapezoidal rule (Tustin approximation). The bandwidth estimation is done by the algorithm (3-1) as in the following pseudocode:

Algorithm 3-1 *BWE in TCP Westwood*

```
if (ACK is received)
  sample_BWE[k] = (acked*pkt_size*8) / (now - lastacktime);
  BWE[k] = (19/21) *BWE[k-1] + (1/21) *(sample_BWE[k] + sample_BWE[k-1]);
endif;
```

where:

acked:	number of segments acknowledged,
pkt_size:	segment size in bytes,
now:	current time,
lastacktime:	time the previous ACK was received,
k and (k-1):	current and the previous value of the variable,
BWE[k]:	low-pass filtered measurement of the available bandwidth at sample k.

TCP westwood Algorithm

Here, we describe in the algorithms (3-2) and (3-3) how the bandwidth estimation can be used in TCP Westwood to control network congestions [36].

A. Algorithm after n *duplicate ACKs*

The pseudocode of the algorithm is the following:

Algorithm 3-2 **Window growth of TCP Westwood after n duplicate ACKs**

```
if (n DUPACKs are received)
      if (cwin > ssthresh) /* congestion avoidance */
            ssthresh = f1(BWE*RTTmin);
            cwin = ssthresh;
      endif
      if (cwin < ssthresh) /* slow start */
            ssthresh = f2(BWE*RTTmin);
            if (cwin > ssthresh)
                  cwin = ssthresh;
            endif
      endif
endif
```

B. Algorithm after coarse timeout expiration

The pseudocode of the algorithm is:

Algorithm 3-3 **Window growth of TCP Westwood *after coarse timeout expiration***

```
if (coarse timeout expires) /* RTO expired */
      if (cwin > ssthresh) /* congestion avoid. */
            ssthresh = f3(BWE*RTTmin);
                  if (ssthresh < 2)
                        ssthresh =2;
                        cwin =1;
                  else
                        cwin =f4(BWE*RTTmin);
                  endif
      endif

      if (cwin < ssthresh) /* slow start*/
      ssthresh = f5 (BWE *RTTmin);
            if (ssthresh < 2) ssthresh = 2;
                  cwin =1;
            else
                  cwin =f6(BWE*RTTmin);
            endif
      endif
endif
```

3.2.2 TCP Westwood+

The definition of TCP Westwood+ protocol is given bellow by its founder Saverio Mascolo [32][37]:

“TCP Westwood+ is a sender-side only modification of the TCP Reno/NewReno classic congestion control protocol stack that optimizes the performance of TCP congestion control especially over wireless networks.”

The main difference between TCP Westwood and TCP Westwood+ is, that in the first one each sample is calculated for each single ACK, that arrives

at the sender. This is wrong, because it results in overestimation of the bandwidth. On the other side, TCP Westwood+ calculates the estimation for each round-trip time (RTT). Hence, the estimation in TCP Westwood+ is more accurate, and reflects the precise value of available network capacity. Following figure shows the BW estimation in TCP westwood+ after a congestion event, and how to set the new states of ***ssthresh*** and ***cwnd***.

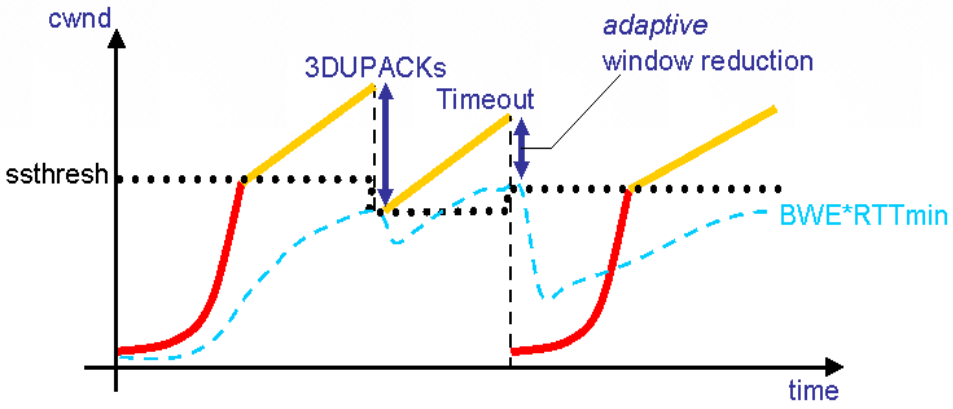


Figure 3-1: Window Dynamics of TCP Westwood+ [47]

As TCP westwood, TCP westwood+ is an end-to-end approach, that makes use of the bandwidth estimation to set *cwnd* and *ssthresh* after a congestion episode, that is, after 3 duplicate acknowledgments (3 DUPACKs), or if the timeout threshold exceeded (RTO expired). Since Linux kernel 2.6, TCP westwood+ is considered as the main congestion control protocol of Linux operating systems. Key idea of TCP westwood+: end-to-end approach, that makes use of the rate of returning ACKs to calculate the available network capacity (bandwidth). To calculate the throughput of TCP Westwood+, we use the following equation [32]:

$$T^{ww+} = \frac{1}{\sqrt{RTT * T_q}} * \sqrt{\frac{1-p}{p}} \qquad (3.1)$$

where:

RTT: round trip time,

T_q: average queueing delay,
p: packet error rate

Figure 3-2 shows the throughput of TCP Westwood+ as a 3D-function of ***RTT*** (x-axis) and $\boldsymbol{T_q}$ (y-axis), while the throughput (***T***) is depicted as (z-axis). We assumed here, that (p) is constant. This plot is a general representation of the growth rate of TCP Westwood+. It is clear, that for high values of ***RTT*** and $\boldsymbol{T_q}$, the throughput is low and changes in very small portions, while it is exponentially increasing for low ***RTT*** and $\boldsymbol{T_q}$.

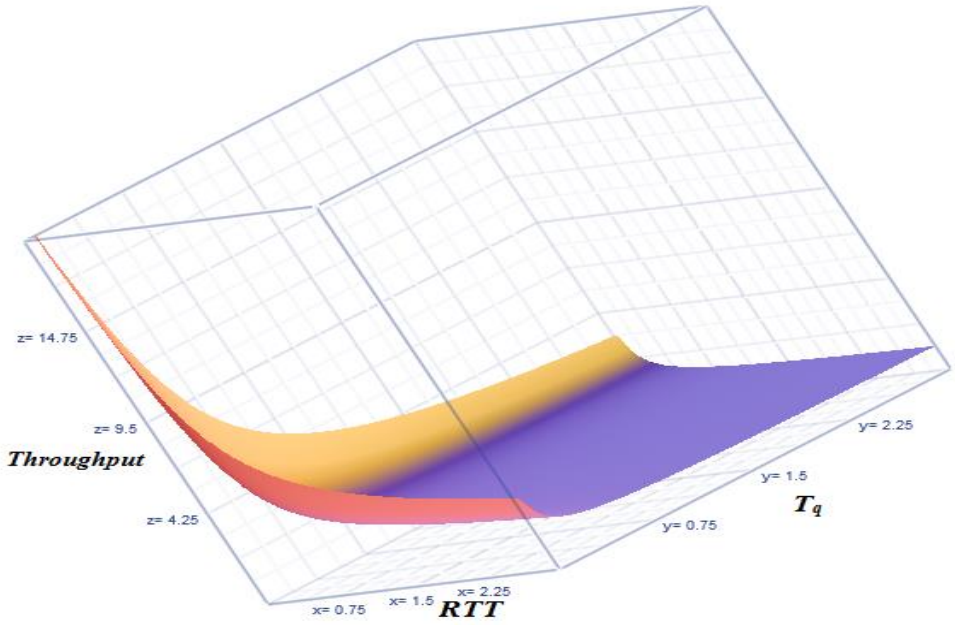


Figure 3-2: Throughput of Westwood+ as 3D plot for RTT and T_q

Like in figure 3-2, we see, that the more realistic plot of throughput in figure 3-3 shows an exponential behavior. In figure 3-3, we assume, that the average queueing delay ($\boldsymbol{T_q}$) is constant, while (p) and ***RTT*** are the variables x and y, respectively.

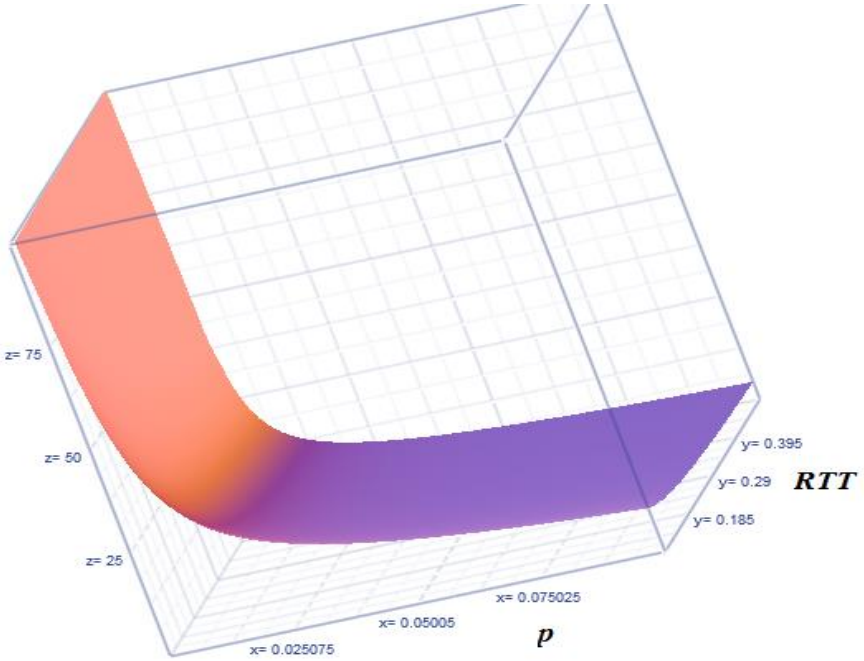


Figure 3-3: Throughput of Westwood+ as 3D plot for *p* and *RTT*

In figure 3-3, the x-axis represents the packet-error-rate or (***p***). It can take values in the range [0,1], while ***RTT*** (y-axis) can take random values. We assumed here, that (***p***) takes values less than 0.1, and ***RTT*** is in the range 0.08 and 0.5. We see in this figure similar results as in the simulations (***chapter*** 5). The throughput is small and not changing for high values of (*p*), even if the value of ***RTT*** is small. On the other hand, we have exponential increase, if (***p***) is low (< 0.025). This behavior is similar to the behavior of TCP DCM+.

When comparing TCP Westwood+ with TCP Reno, as shown in equation (3.2), it is clear, that both throughputs depend on ($\frac{1}{\sqrt{p}}$), that is they are friendly to each other. Also, Reno throughput depends on the value ($\frac{1}{RTT}$), while Westwood+ depends on ($\frac{1}{\sqrt{RTT}}$), which means, Westwood+ increases the fair-sharing of the network capacity between the flows with different ***RTT***s.

$$\boldsymbol{T^{Reno} = \frac{1}{RTT} * \sqrt{\frac{2*(1-p)}{p}}} \qquad (3.2)$$

TCP Westwood+ follows the following algorithm (3-4) [8].

Algorithm 3-4 TCP Westwood+

```
Input = in

Output = out

1. If (in = ACK) then
2. Estimate available bandwidth;
3. out = Increase cwnd according to NewReno;
4. end if

Else

5. If in = (3 DUPACKs) then
6. ssthresh = max (2, (BWE*RTTmin)/Seg_Size);
7. cwnd = ssthresh;
8. out = cwnd;
9. end if

Else

10.    if in = (coarse timeout) then
11.    ssthresh = max (2, (BWE*RTTmin)/Seg_Size);
12.    cwnd = 1;
13.    out = cwnd;
14.    end if
15.    return (out);
```

Algorithm (3-4) of Westwood+ shows the reactions of the TCP sender on new coming ***ACK*** segments. It increases additively as NewReno, if normal ***ACK*** is the input. Otherwise, if 3 duplicate acknowledgements (***DUPACK***s) or a timeout segment (***RTO***) are the input, then ***ssthresh*** and ***cwnd*** are readjusted.

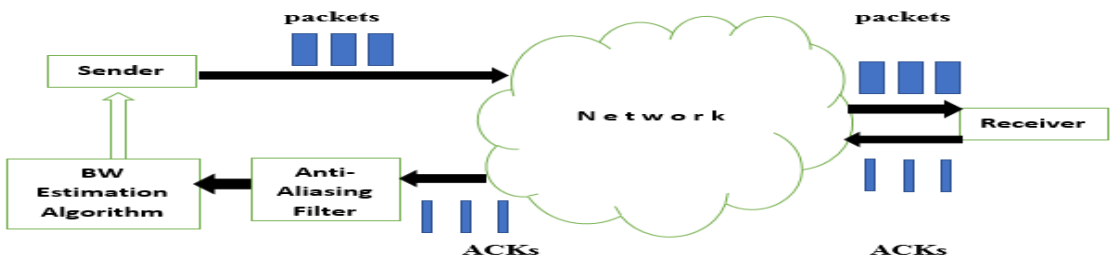


Figure 3-4: Key Working Idea of TCP Westwood/Westwood+

Anti-Aliasing Filter

AAF is a filter used to limit the signal bandwidth in order to satisfy the sampling theorem of Nyquist–Shannon.

$$\boldsymbol{b_i = \frac{d_i}{RTT_i}} \tag{3.3}$$

where:

b_i : anti-aliasing BW sample (W at the filter output)

d_i : data successfully acknowledged from receiver within last RTT (*i*)

RTT_i : last RTT (i)

The filter's equations are given below:

$$\boldsymbol{\hat{b}_k = \alpha \cdot \hat{b}_{k-1} + (1 - \alpha) \cdot b_k} \tag{3.4}$$

$$\boldsymbol{b_k = \frac{d_k}{RTT_k}} \tag{3.5}$$

where:

b_k : bandwidth sampled measured at time point k,

$\hat{b}_k$: bandwidth estimation from the filter at time point k,

$\hat{b}_{k-1}$: bandwidth estimation from the filter at time point (k-1),

α : filter constant (19/21)

3.3 TCP Hybla

TCP Hybla [38] has been designed to solve the problems of heterogenous networks that exhibit large round-trip times (RTTs) in their TCP connections. Terrestrial or satellite links are such networks that are disadvantaged because of their very long RTTs. TCP Hybla has emerged as an analytical model, as depicted below, which stems from studying the dynamics of congestion window in standard TCP variants (Tahoe, Reno, NewReno). This model suggests some

necessary modifications to remove the dependence of TCP performance on RTT.

As mentioned in its proposal, TCP Hybla reduces the penalization suffered by the wireless links, i.e. the satellites connections. Also, Hybla does not infringe the end-to-end semantics of TCP. Hence, it is compatible with the standard TCP. The equation below describes the growth rate of the congestion window $W(t)$ of normal TCP as a function of RTT. As seen, it is RTT-dependent.

$$W(t) = \begin{cases} 2^{t/RTT}, & 0 \le t < t_\gamma \quad , SS \\ \frac{t - t_\gamma}{RTT} + \gamma, & t \ge t_\gamma, \quad CA \end{cases} \tag{3.6}$$

where:

$W(t)$: the congestion window expressed in segments,

γ : the slow-start threshold (***ssthresh***), which is defined as

$$\gamma = 2^{t_\gamma / RTT} \tag{3.7}$$

t_γ : the time at which ***ssthresh*** is reached. It results from the above equation:

$$t_\gamma = RTT * log_2 \gamma \tag{3.8}$$

The next figure shows the growth rate of normal TCP, which depends on RTT. It is apparent that slower connections are penalized by a longer time to reach the required ***ssthresh*** value.

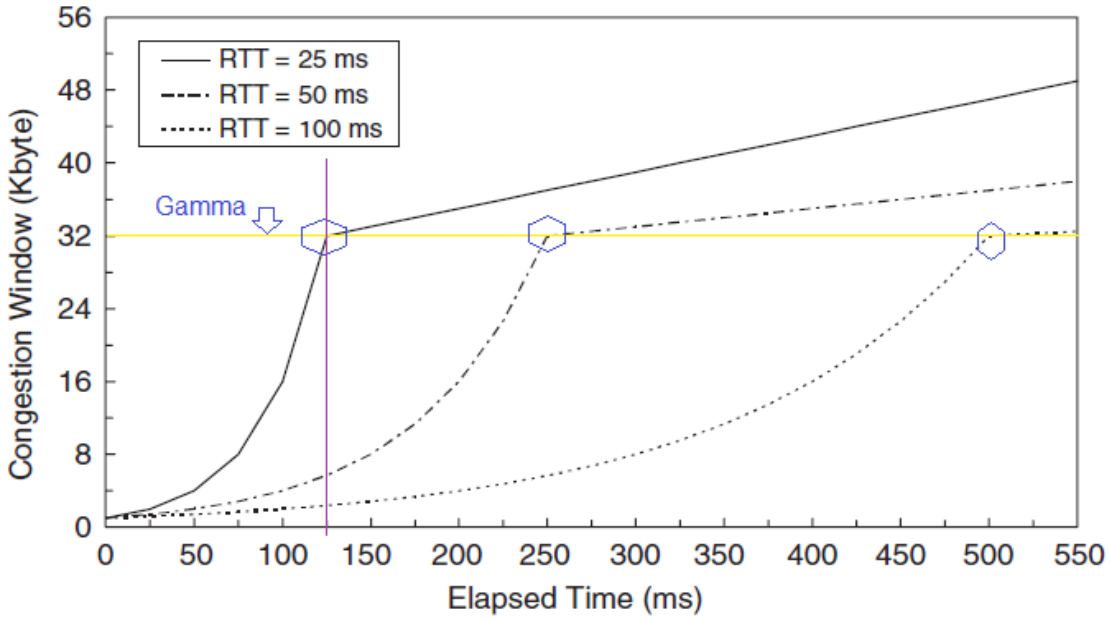


Figure 3-5: Window Dynamics of Standard TCP [38]

It is clear that with larger RTTs, the growth rate of the congestion window becomes slower, hence, t_γ is higher. Now, we define the segments transmission rate as:

$$\boldsymbol{B(t) = \frac{W(t)}{RTT}} \tag{3.9}$$

After modifying the congestion window to be RTT-independent, as suggested by TCP Hybla, we get:

$$\boldsymbol{W^H(t) = \begin{cases} \rho * 2^{\frac{\rho * t}{RTT}}, & 0 \le t < t_{\gamma,0}, SS \\ \rho * [\rho * \frac{t - t_{\gamma,0}}{RTT} + \gamma], & t \ge t_{\gamma,0}, CA \end{cases}}, \tag{3.10}$$

$$\boldsymbol{\rho = \frac{RTT}{RTT_0}} \tag{3.11}$$

where:

$W^H(t)$: the congestion window of TCP Hybla expressed in segments,

ρ : the normalized round-trip time,

RTT_0 : the round-trip time of the reference connection.

So, according to TCP Hybla, we have a segments transmission rate $B^H(t)$, which is independent of RTT, and defined as:

$$\boldsymbol{B^H(t) = \frac{W^H(t)}{RTT}} \qquad (3.12)$$

Following figure shows the congestion window of TCP Hybla for different RTTs.

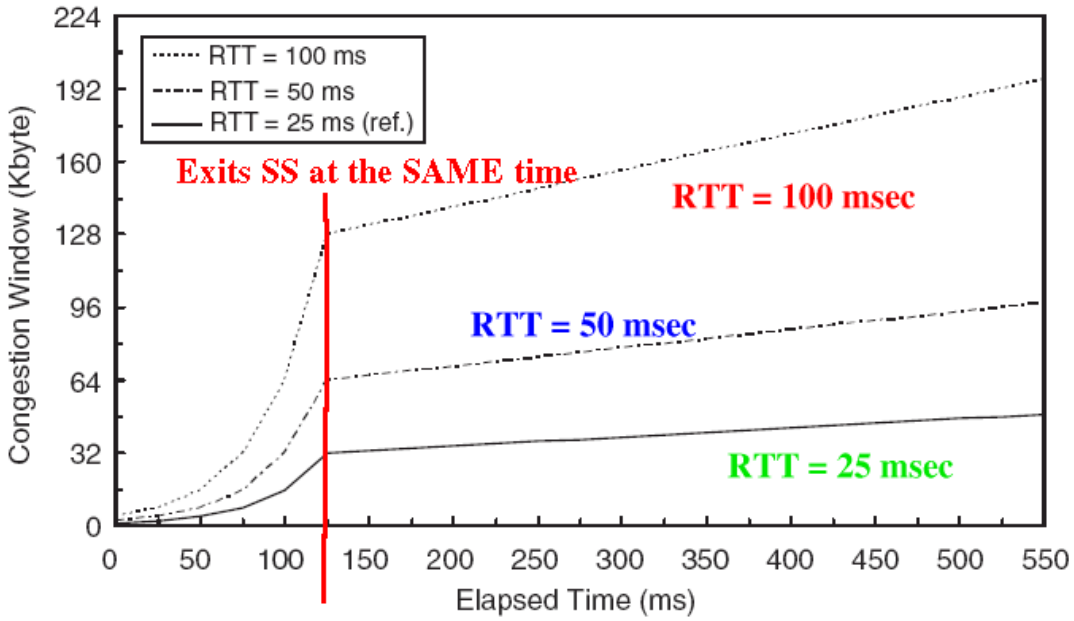


Figure 3-6: Window Dynamics of TCP Hybla [38]

It is visible that for any possible RTT, the congestion window does not depend on RTT, and the time needed to reach ***ssthresh*** is the same for all RTTs. The following equation shows the segments transmission rate $B^H(t)$ as a function of RTT_0 only. For fast connections, ($RTT \leq RTT_0$), Hybla behaves as the standard TCP.

$$\boldsymbol{B^H(t) = \begin{cases} \frac{1}{RTT0} * [\,2^{\frac{t}{RTT0}}\,], & 0 \leq t < t_{\gamma,0} \quad , SS \\ \frac{1}{RTT0} * [\,\frac{t - t_{\gamma,0}}{RTT0} + \gamma\,], & t \geq t_{\gamma,0}\,, \quad CA \end{cases}} \qquad (3.13)$$

3.4 TCP BIC (Binary Increase Congestion Control)

TCP BIC [39] consists of the following main parts as shown in the next figure:

- *binary search increase*,
- *additive increase*, and
- *probing*

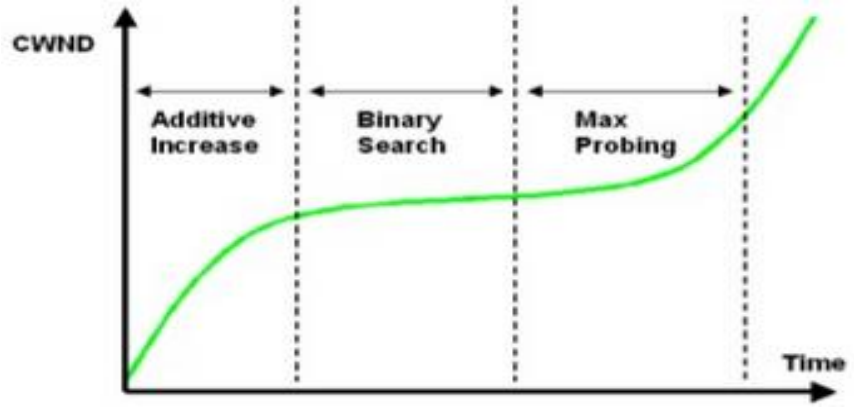


Figure 3-7: Window Phases of TCP BIC [40]

In the phase of *binary search increase*, the BIC congestion control is viewed as a search problem. It gives "yes/no" feedback through packet loss as to whether the current window size is larger than the network capacity. The search problem uses 2 starting points: *Wmin* and *Wmax*, which are the minimum and maximum window sizes, respectively. *Wmax* is defined as the window size just before the last packet loss occurred (also called fast recovery). *Wmin* is the window size just after the last packet loss.

The algorithm of this phase repeatedly computes a new value for the midpoint between *Wmax* and *Wmin*. Then, it sets the current window size to this value. Thereafter, it checks for feedback, in the form of packet losses. Based on this feedback, the midpoint is taken as the new *Wmax* if there is a packet loss, and as the new *Wmin* if not. The above process continues until the difference

between *Wmax* and *Wmin* is smaller than a preset threshold, called *the minimum increment* (*Smin*). On the other hand, *Smax* is the maximum increment. This phase is needed to probe the available bandwidth. It is aggressive, when the difference between the current window size and the target window size is large, but becomes less aggressive as the current window size gets closer to the target window size.

The second part of BIC algorithm is the *additive increase*. It shows a linear behavior as shown in the figure below. When combined with the strategy "*binary search increase"*, the strategy "*additive increase*" ensures faster convergence and RTT-fairness. This combination of *binary search increase* and *additive increase* is called *binary increase*. Combined with a multiplicative decrease strategy, *binary increase* becomes close to pure additive increase under large windows. This is because a larger window results in a larger reduction (large decrease factor: β) in multiplicative decrease. Therefore, a longer additive increase period. When the window size is small, it becomes close to pure binary search increase – a shorter additive increase period [40]. Figure (3-8) shows the behavior of TCP BIC during its phases.

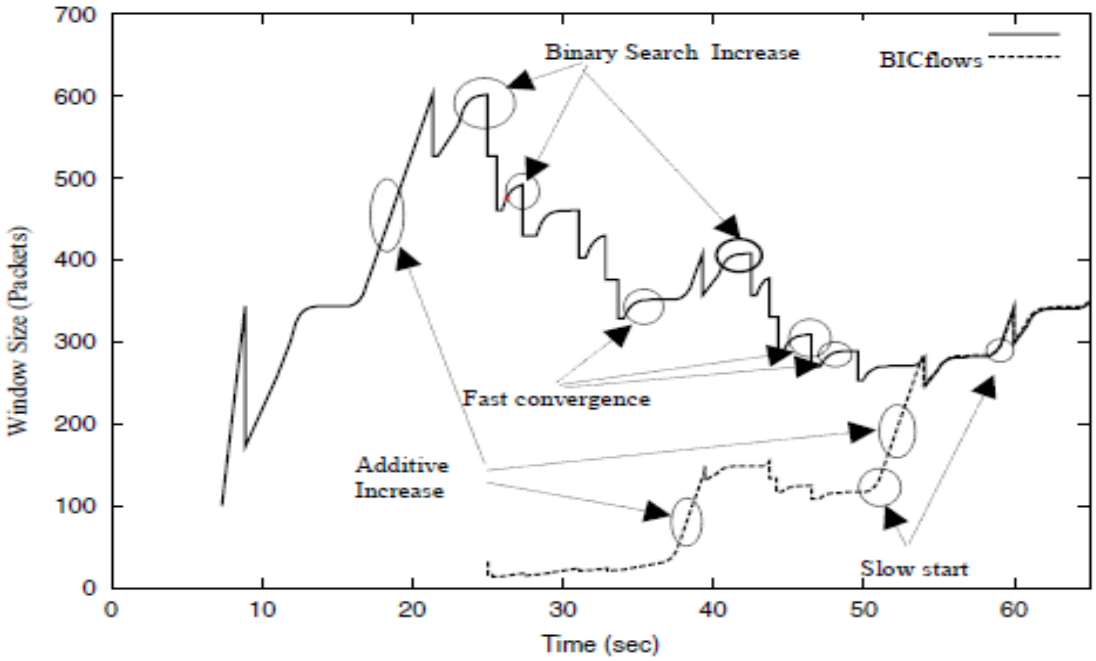


Figure 3-8: Behavior of TCP BIC During its Phases [39]

The pseudo-code below describes the principle of TCP BIC

Algorithm 3-5 TCP BIC

```
while (cwnd < Wmax) {
     if ( (midpoint - Wmin) > Smax )
       cwnd = cwnd + Smax
      else
       if ((midpoint - Wmin) < Smin)
            cwnd = Wmax
       else
            cwnd = midpoint
     if (no packet loss)
       Wmin = cwnd
     else
       Wmin = β*cwnd
       Wmax = cwnd
     midpoint = (Wmax + Wmin)/2
 }
```

The bandwidth probing is executed according to the following algorithm:

Algorithm 3-6 BW Probing in TCP BIC

```
while (cwnd >= W_max) {
   if (cwnd < W_max + S_max)
        cwnd = cwnd + S_min
   else
        cwnd = cwnd + S_max

   if (packet loss)
        W_min = β*cwnd
        W_max = cwnd
}
```

The throughput of TCP BIC for very large window size can be given as:

$$R \approx \frac{1}{RTT}\sqrt{\frac{S_{max}}{2}\frac{2-\beta}{\beta}\frac{1}{p}} \tag{3.14}$$

But for very small window sizes, the throughput is:

$$R \approx \frac{W_{max}}{RTT} \tag{3.15}$$

where:

$$W_{max} \approx \frac{1}{\left(\log_2\left(\frac{W_{max}\,\beta}{S_{min}}\right)+2\,(1-\beta)\right)p} \tag{3.16}$$

3.5 TCP Ledbat

LEDBAT [41] is an abbreviation for **L**ow **E**xtra **D**elay **Ba**ckground **T**ransport, which is described in the RFC 6817. TCP LEADBAT is meanly used with point-to-point (P2P) applications like BitTorrent and for non-interactive streaming applications. It is an experimental congestion control protocol, that is based on one-way delay. When a queue is building up, then the one-way delay is increasing, which means a congestion may happen. It utilizes the available bandwidth on an end-to-end path while limiting the consequent increase in queueing delay on that path. Following figure shows the working principle of LEDBAT in the presence of TCP traffic.

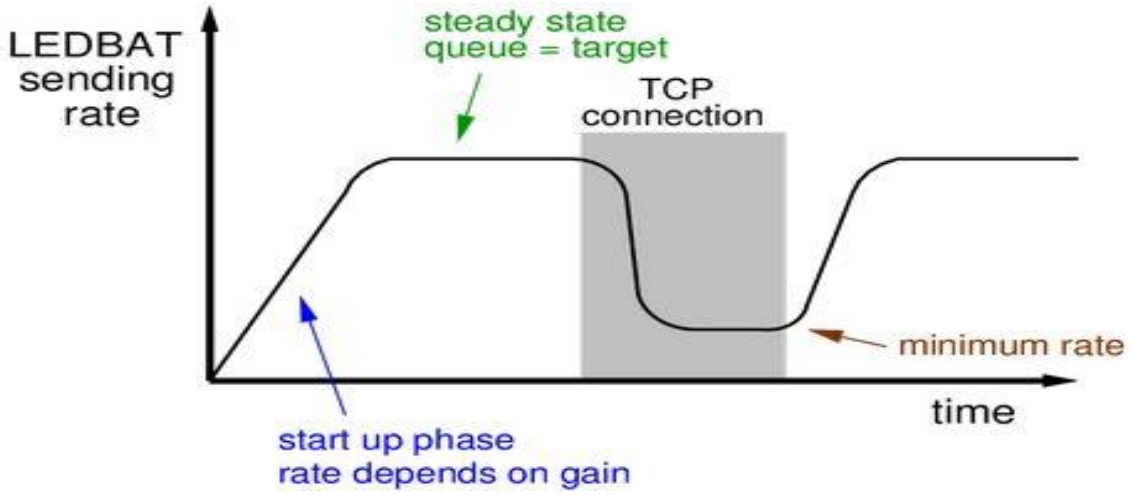


Figure 3-9: Window Dynamics of TCP Ledbat [25][41]

Figure (3-9) above explains the design goal of LEDBAT, and why it belongs to the class of transport protocols known as "Lower-Than-Best-Effort-Protocols". TARGET here stands for the value of queueing delay for which the congestion window ***cwnd*** is maximal without causing packet losses. At TAGET, LEDBAT is causing the maximum allowable queueing delay, which is

introduced in the network. Next figure shows this idea and a comparison with the standard TCP variants like TCP Reno and TCP NewReno.

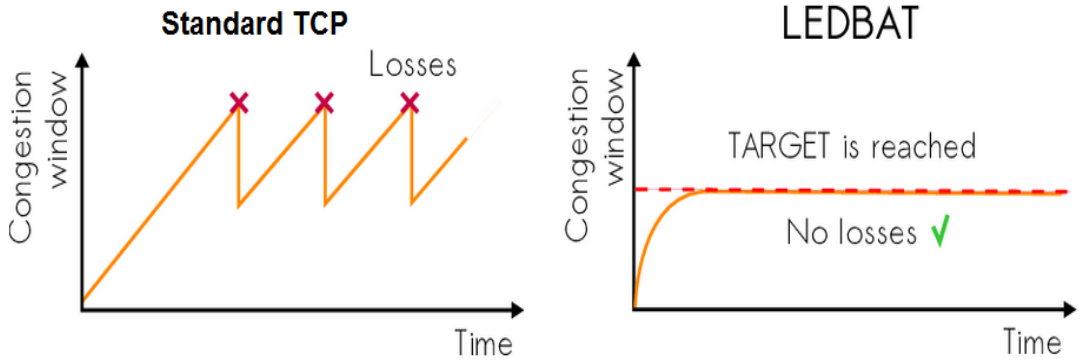


Figure 3-10: Congestion Management in TCP Ledbat vs. Standard TCP

Another parameter called gain (G) is used to determine the rate at which the ***cwnd*** responds to changes in the queueing delay. The ***cwnd*** increase or decrease of LEDBAT depends on the difference between the current measurement of queueing delay ($D_{current}$) and the predetermined delay (TARGET). This difference is called ***offtarget***. Following equations describe the dynamic behavior of LEDBAT.

$$\boldsymbol{offtarget = TARGET - D_{Current}} \tag{3.17}$$

$$\boldsymbol{cwnd = h\,(offtarget) = w(t) + \frac{G*offset}{w(t)}} \tag{3.18}$$

where:

$D_{current}$: the current queueing delay measured in the network.

$h\,(\,.\,)$: a function describing the behavior of ***cwnd*** depending on ***offtarget***.

The time evolution of LEDBAT source congestion window is calculated according to:

$$w(t+1) = \begin{cases} \frac{1}{2}\, w(t) & \textit{if packet loss} \\ w_{min} & \textit{if } cwnd \leq w_{min} \\ cwnd & \textit{otherwise} \end{cases} \quad (3.19)$$

According to RFC paper of Ledbat, following pseudo-code shows the algorithm and the operations of sender and receiver of LEDBAT.

Algorithm 3-7 TCP Ledabt

```
on data_packet @ RX:
   remote_timestamp = data_packet.timestamp
   acknowledgement.delay =
       local_timestamp() - remote_timestamp

on acknowledgement @ TX:
   current_delay = acknowledgement.delay
   base_delay = min(base_delay, current_delay)
   queuing_delay = current_delay - base_delay
   off_target = TARGET - queuing_delay
   cwnd += GAIN * off_target / cwnd
```

3.6 Summary

We tested some of the known TCP protocols dedicated for congestion control in wired and wireless networks and we compared them versus Westwood+. As visible in the next figures, we see that TCP Westwood+ is preferred over all other protocols as it has the highest throughput (***T***), the best link utilization (***U***) and the shortest time to complete transmission (***CTT***). We assume that the data file has the size 100 MB. The bottleneck bandwidth is 100 Mbps and the time needed for transmitting the file is ***Tx,*** which depends on the packet error rate, the segment size (MTU) and the protocol type. Based on these values, we get the following equations:

1. $$U = \frac{Databits \times 100}{Bandwidth \times Interval} = \frac{100 * 10^6 * 8\ bit}{100 * 10^6 * \frac{bit}{sec} * Tx} * 100\ (\%) = \frac{800}{(\frac{Tx}{sec})}\ (\%) \quad (3.20)$$

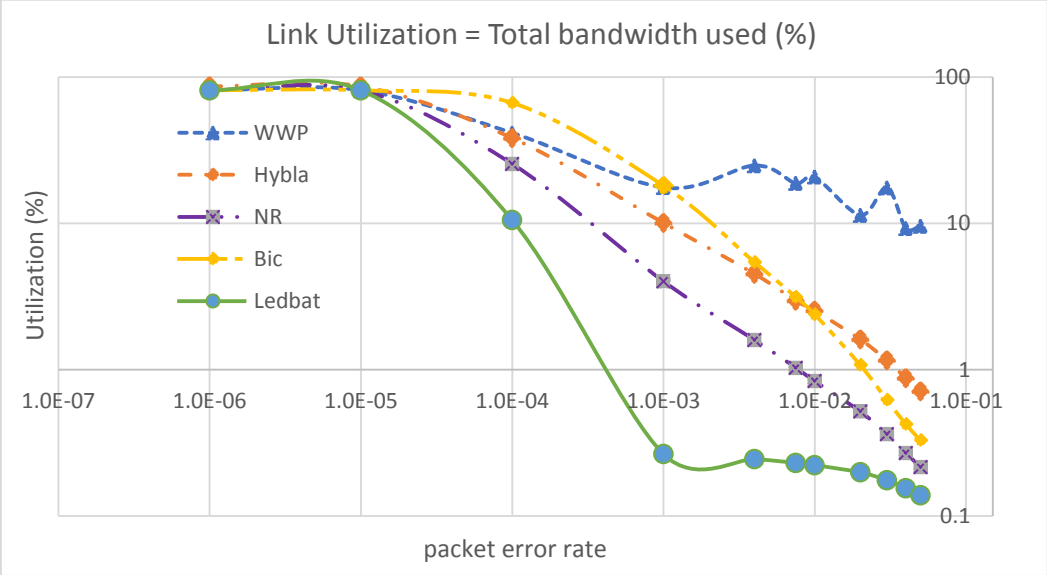


Figure 3-11: Network Utilization for Different TCP Approaches

Figure 3-11 shows that Westwood+ has the best utilization when the packet error rate is higher than 1e-3.

2. $$T = \frac{Total\ Number\ of\ Bytes\ Transmitted}{Time\ Interval}$$ (Throughput Kbps) (3.21)

Figure 3-12 shows that Westwood+ has the best throughput when the error rate is higher than 1e-3.

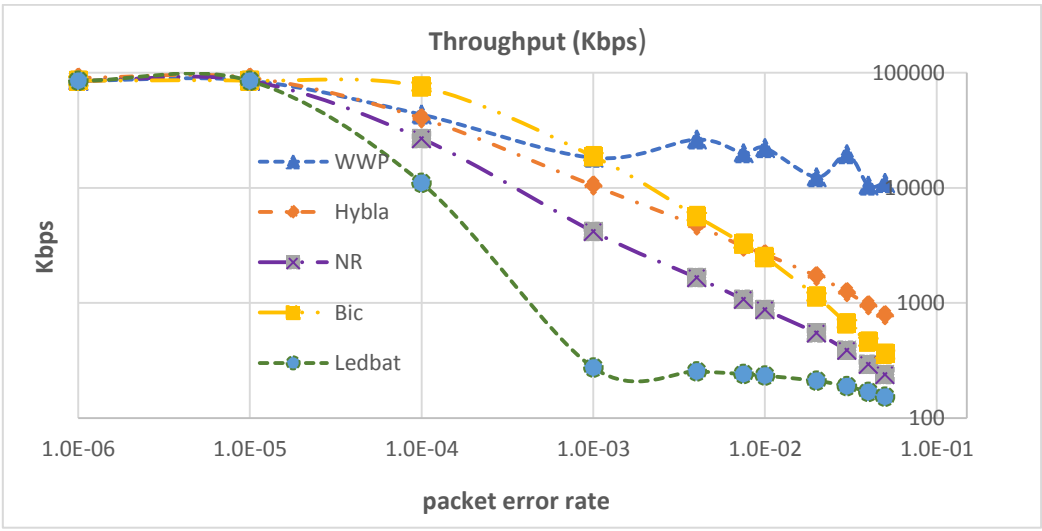


Figure 3-12: Network Throughput for Different TCP Approaches

3. ***CTT = Last_Ack_Time - First_Ack_Time*** (3.22)

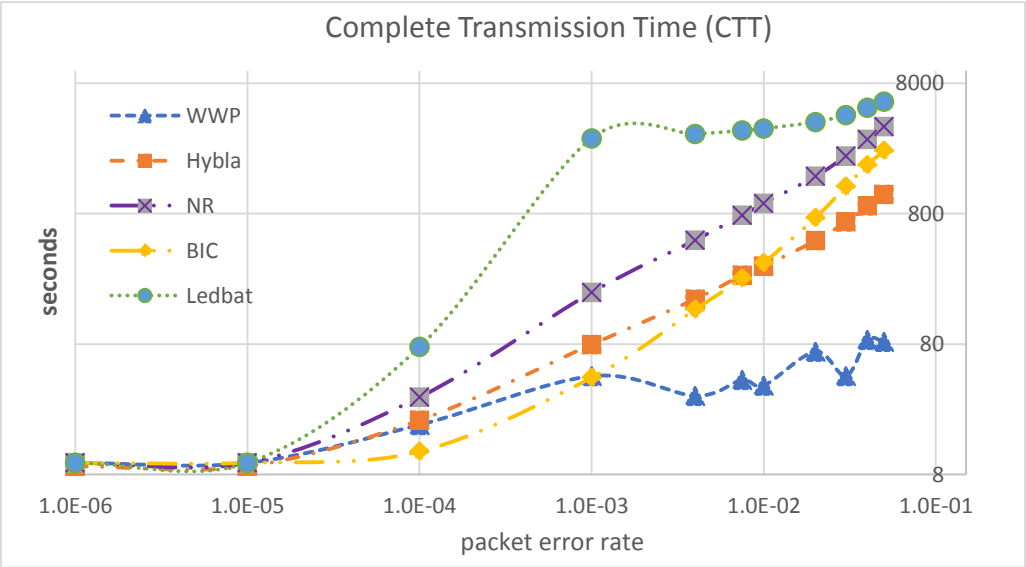


Figure 3-13: CTT for Different TCP Approaches

Figure 3-13 shows that Westwood+ has the shortest transmission time when the error rate is higher than 1e-3.

Chapter Four: Proposed approach - DCM+

Content

4.1 Introduction

TCP DCM+ is a new approach, that we have proposed in [8], and has been explained in more depth in [9]. It stands for ***D***ynamic ***C***ongestion control for wireless and ***M***obile systems. It uses the bandwidth estimation (***BWE***) algorithm of TCP Westwood+, and hence comes the (+) sign. Please refer to chapter 3 to see the procedure of ***BWE*** in Westwood+. Figure 4-1 shows the origins of TCP DCM+. The transition lines between the different algorithms show the time evolution of these approaches. We see that DCM+ has its origins in many different techniques, i.e. DCM [42], which aims at improving the performance in MANETs using TCP Westwood.

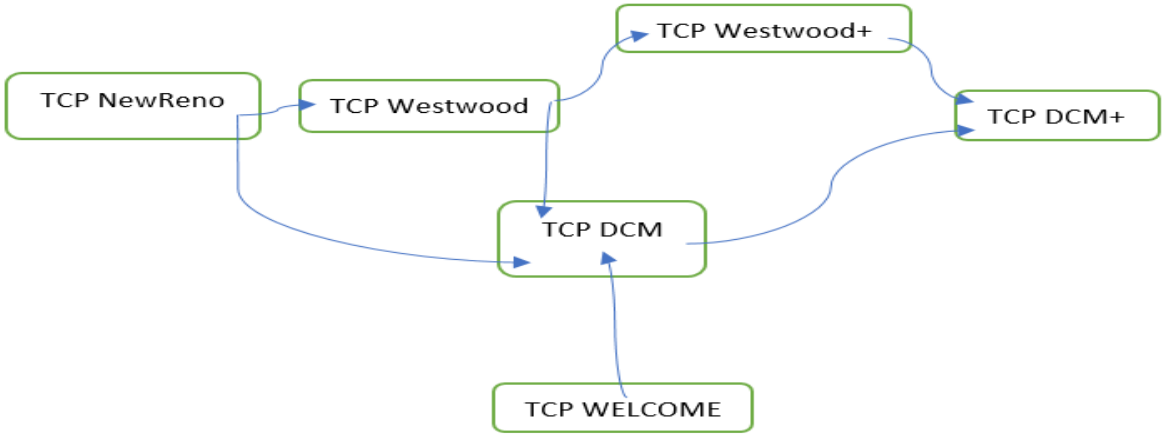


Figure 4-1: Origin of TCP DCM+

TCP DCM+ has been designed to avoid the congestion problems occurring in wireless and mobile networks. It is an L4-only protocol, which can be used in different network types like wired, wireless and MANETS. It shows its tremendous performance improvement when the packet error rate is between (0.001 and 0.04).

In chapter 5, we compare DCM+ versus other protocols and show that it is superior to these approaches. Our comparison uses TCP NewReno, TCP Hybla, TCP Ledbat, TCP BIC, TCP Vegas, TCP HighSpeed and TCP Westwood+. The properties of DCM+ like stability and robustness are eminent. For packet error rates less than 0.05, it shows very similar results like TCP Westwood+, however it has the advantage that it does not suffer any window drops because of congestion. The only drops it shows are those caused by wireless channel losses.

4.2 Window Dynamics of TCP DCM+

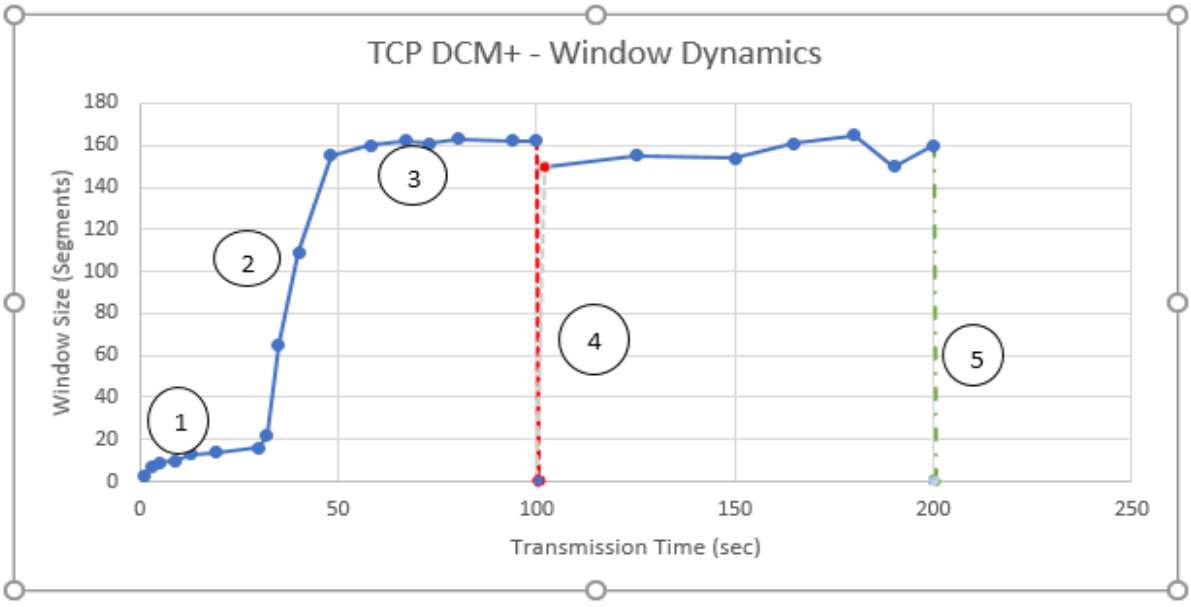


Figure 4-2: Phases of TCP DCM+

Applying TCP DCM+ for any connection between end devices may consist of the following 5 phases like in figure 4-2:

1- *Initialization* (Probing) phase (IP), where rate of growth (***cwnd***) is small, though channel capacity is not exhausted. In this phase, no danger of congestion is expected. Packet losses may happen depending on the channel conditions. This phase is a core phase, that always happen at the beginning of each DCM+ transmission. Small amounts of data are sent within this period; hence, their throughput may be less or equal than other approaches.

2- *Advancing* phase (AP), where the transmission increases quickly by large steps (exponentially). This is a core phase for mid to large files, which is the reason for high performance of DCM+.

3- *Near-Channel-Capacity* phase (NCCP), where TCP DCM+ uses the whole available channel capacity to transmit data at a near-constant rate. This is a core phase for large files. It can be reached after an AP- or an LP-phase like in phase 4.

4- *Losses* phase (LP), where a lost packet is detected and retransmitted. The channel capacity after sending the lost packets is nearly equal the available channel capacity before the losses has been detected. This phase may occur in any of the previous phases, but it is more convenient to happen within the phases 2 and 3, because the number of sent segment in phases 2 and 3 is the highest. The number of losses in all phases depend on TCP buffer size and the channel conditions like packet error rate, bottleneck BW, segment size, etc. This phase is random and not a core phase.

5- *End* phase (EP), where the transmission is completed successfully. Drops may happen randomly at the end of this phase. See figure 5-7.

4.3 TCP DCM+ algorithm

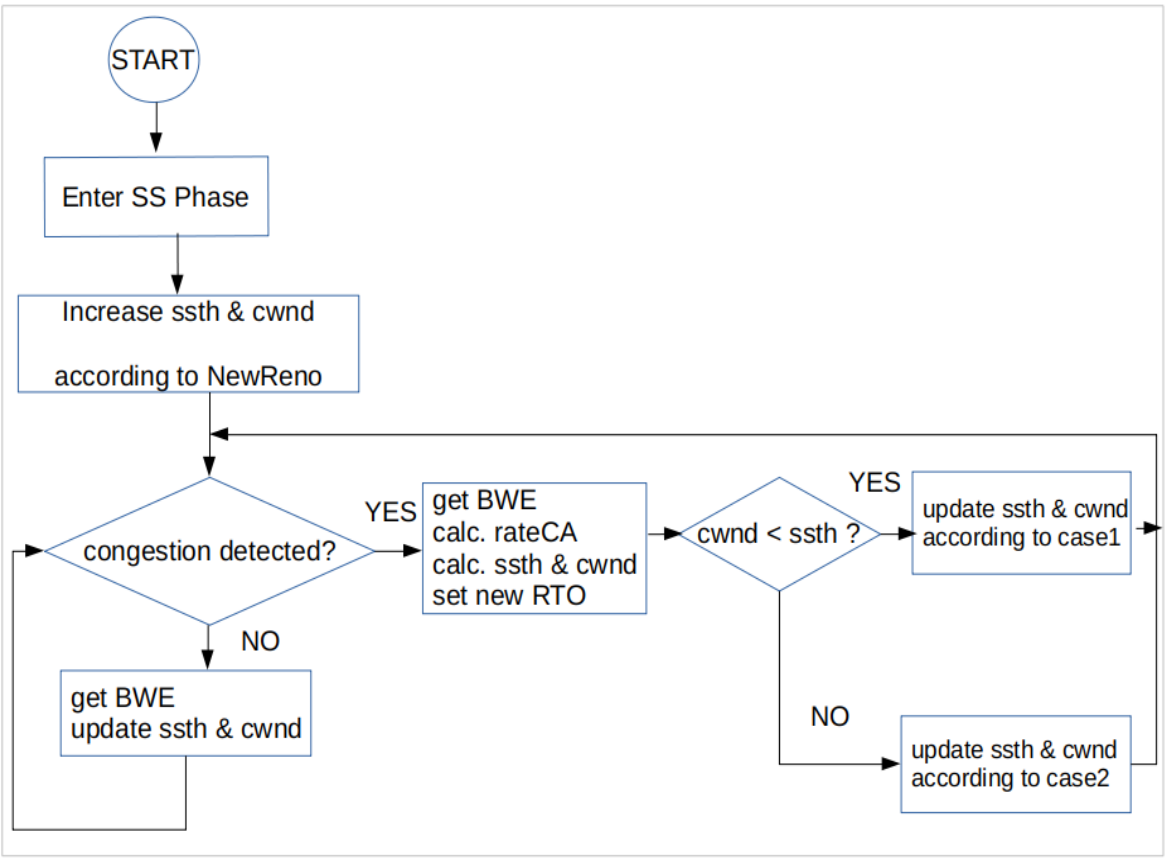
As mentioned previously, DCM+ is an end-to-end (***E2E***) technique, which maintains the standard TCP semantics and does not create any additional overhead. DCM+ is used from the TCP sender to control the sent amount of data on the transmission link. As given in the published paper [8][9] and [10], the window increase follows the following equations:

$$\frac{RTO_{new}}{RTO_{old}} = \frac{RTT_{new}}{RTT_{old}} \tag{4.1}$$

$$CWND_{new} = \begin{cases} CWND_{old} + 2 * rate_{CA} \; ; & cwnd < ssthresh \\ CWND_{old} + \left(\frac{2}{rate_{CA} * CWND_{old}}\right) \; ; & cwnd \geq ssthresh \end{cases} \tag{4.2}$$

Algorithm (4-1) below shows the behavior of TCP DCM+.

Algorithm 4-1 TCP DCM+



From the view point of TCP NewReno, it has modified behaviors in both ***SS*** and ***CA*** phases. However, from view point of TCP Westwood+ [35][36], ***CA*** only is modified. This modification is done through C++-*function overloading* in the Westwood+ implementation files under ns-3.29 network simulator. Through function overloading, we are able to redefine the behavior as needed. The complete behavior is placed in the belonging C++ file, which ends in ns3 with '*.cc'* . The new member function that we redefine in 'TcpWestwood.cc' file is:

```
void TcpWestwood::CongestionAvoidance(Ptr<TcpSocketState> tcb,
uint32_t SegmentsAcked)
```

This member function has 2 parameters, which are described as follows:

tcb: internal congestion state,

and

segmentsAcked: count of segments acked.

The behavior of this function is described in the algorithm (4-2):

Algorithm 4-2 Congestion Avoidance in TCP DCM+

```
rateCA = oldRtt / minRtt    ;

minRto = minRto / rateCA    ;

If (segmentsAcked > 0)

{     if (cwnd <= (currentBW * minRtt))

                  {cwnd += 2 * rateCA * segment Size;    }

            else {cwnd += 2 / (cwnd * rateCA);           }

}
```

After the bandwidth estimation (BWE) is calculated, DCM+ calculates the new values for both ***ssth*** and ***cwnd*** depending on the previous values ***RTT***, ***RTO***, the parameter ***rateCA*** and whether the calculated ***cwnd*** is less than ***ssth*** or not. As feedback signals, we use the previous states of both ***RTT*** and ***RTO***. The behavior of ***cwnd*** is observed to be dynamical, in that it tracks the state of ***ssthresh*** as shown in figure 4-3.

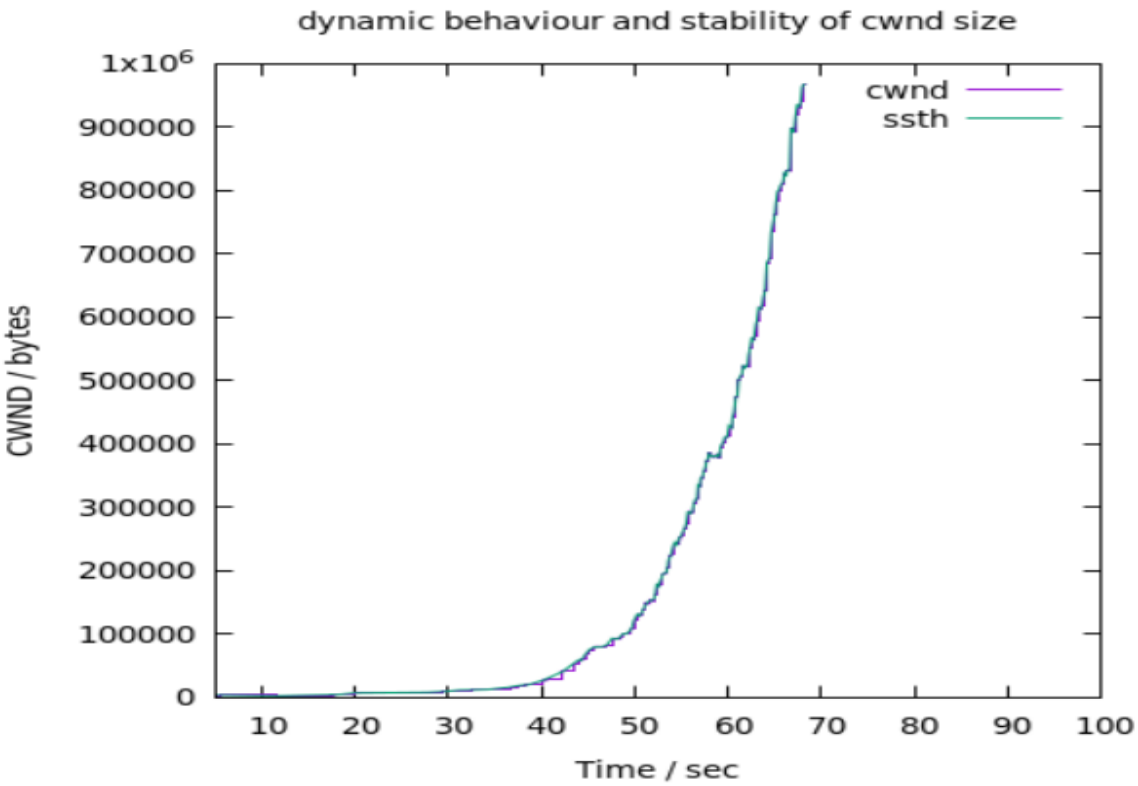


Figure 4-3: Window Dynamics of TCP DCM+

If a change (increase or decrease) of ***ssthresh*** has been observed within a specific time interval, then DCM+ keeps using the current value of ***cwnd*** until a newer ***ssthresh*** has been reached. Thereafter, ***cwnd*** moves and remains at the new state of ***ssthresh*** for the new time interval. This way, ***cwnd*** will *theoretically* never exceed the available ***ssthresh***. Hence, congestion events cannot occur. We *claim* that congestion events are *eliminated*. Figure 4-4 shows this behavior for packet error rate = 7.5e-3 and for MTU size =1200 bytes.

The design of TCP DCM+ is similar to TCP NewReno, which is detailed in RFC 6582. DCM+ uses the same 4 phases like NewReno (***SS***, ***CA***, fast retransmission (***FR***) and fast recovery (***FV***)). In DCM+, the behaviors in (***SS***) and (***CA***) have been so modified to enforce the ***cwnd*** to track ***ssth*** in the next time interval. TCP timing parameters ***RTT*** and ***RTO*** have been used as feedback signals to control the values of ***ssth*** and ***cwnd*** in the next interval.

During the design of this approach, we followed the idea, that a continuous increase in ***RTT*** values leads gradually to an increased queue length in the intermediate nodes. This will later result in a *true* congestion event (i.e., packet drop), and also higher delays of the returning ACK packets to the sending node. Late arrival of ACK packets at the source node may cause wrong interpretation of the network status as "congested", especially if the ***RTO*** threshold is exceeded. In this case, ***cwnd*** will drop down to the value of 1 segment or (MSS=1). So, we tied this idea of congestion with the parameters ***RTT*** and ***RTO***, and we introduced a new parameter called congestion rate or ***rateCA*** as in the equation (4.3), which is an important measure for congestion, especially if the ***CA*** phase has been entered. This parameter is crucial for determining the next appropriate values of ***cwnd*** and ***ssth***. It is defined as the ratio of the previous ***RTT*** divided by the *current minimal* ***RTT***. As a result, all parameters (***cwnd***, ***ssth***, ***RTT***, ***RTO*** and ***rateCA***) in the next interval, are affected, and therefor dynamically changing during the transmission.

$$\boldsymbol{rateCA = \frac{RTT_{old}}{RTTmin}} \qquad (4.3)$$

Now, we consider ***rateCA*** higher than **1** as ***advance*** or "Link Capacity Increasing", and on the other hand, values lower than **1** as ***danger*** or "Link Capacity Decreasing". Depending on the condition stated in ***CA*** phase of DCM+ [8][9][10], if ***cwnd*** is less than or equal ***ssth***, then ***rateCA*** will be used to start the retransmission in wide steps, otherwise, retransmission goes slowly and prevents any possible congestions. Please refer to figure 4-4, which is taken for the same parameters as figure 4-3, to see the dynamics of ***RTT*** and ***rateCA.*** We see, that increased values of ***RTT*** ($RTT_{old} < RTT_{new}$ means ***rateCA*** <1) lead to low values of ***cwnd***, while decreased values of ***RTT*** ($RTT_{old} > RTT_{new}$ means ***rateCA > 1***) lead to an increase of ***cwnd***. We clearly observe, that the average delay is minimized and it took only 68 seconds to transmit the sent data size.

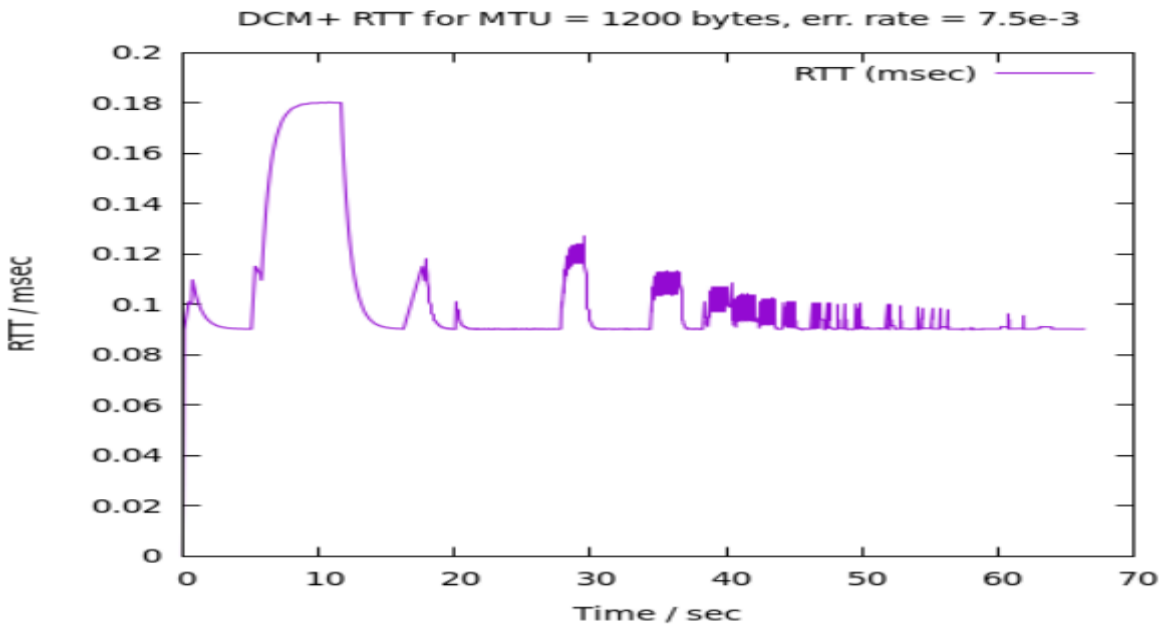


Figure 4-4: Changing of RTT as Indicator of Increase/Decrease of CWND

At each time point during the transmission, the value of the next ***RTO*** is also affected by the newly calculated ***rateCA***. If the current ***RTT*** is decreasing, then ***RTO*** shall be also reduced, as no congestion is expected. As described in the ***CA*** phase of DCM+, next ***RTO*** depends on the current ***rateCA***. This behavior is described through the following equations (4.4) and (4.5):

$$\boldsymbol{rateCA} = \frac{\boldsymbol{RTT_{old}}}{\boldsymbol{RTTmin}} = \frac{\boldsymbol{RTO_{old}}}{\boldsymbol{RTOnew}} \qquad (4.4)$$

The equation above can be reformulated as:

$$\boldsymbol{RTO_{new}} = \frac{\boldsymbol{RTO_{old}}}{\boldsymbol{rateCA}} \qquad (4.5)$$

Figures 4-5 and 4-6 depict an important property of TCP DCM+, namely the detection of link quality, i.e. wireless channel, as a result of the occurring ***RTO*** timeouts, which mean lost packets as a reason for bad link conditions. When a packet is lost because of bad wireless links, then the retransmission timeout counter is reset to a predefined value, here (in the ns3 simulator) the initial RTO value is equal 1 msec.

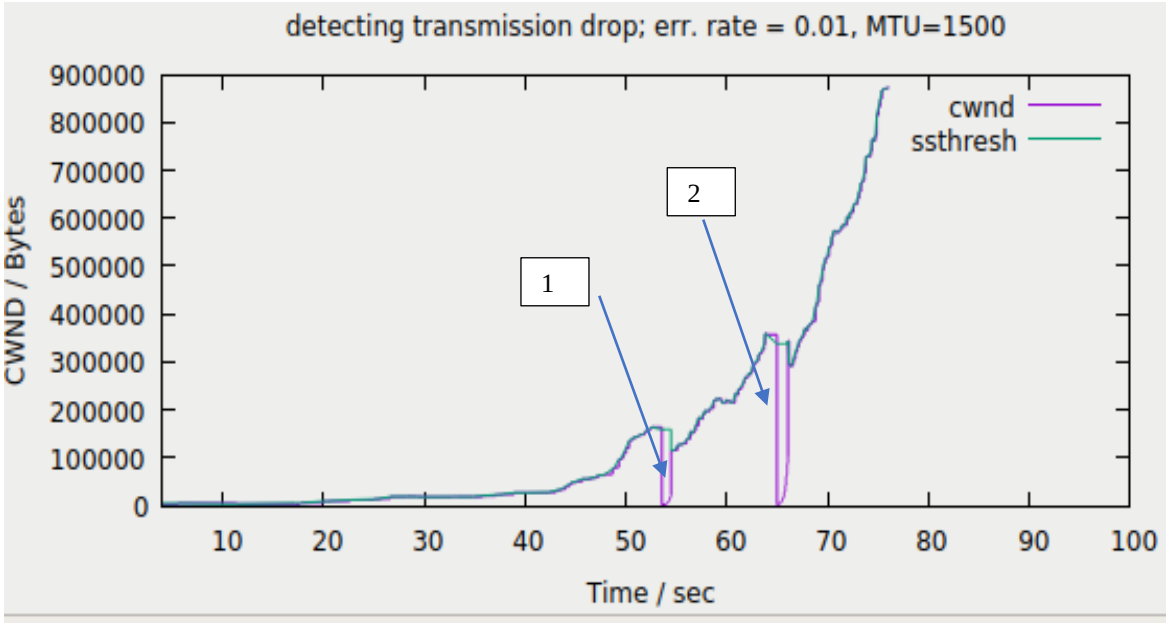


Figure 4-5: Drops of CWND as a Result of Wireless Channel Losses

The drops of ***cwnd***, which are highlighted in figure 4-5 and marked as (1) and (2) coincide with the ***RTO*** reset time points, which are 54 sec and 65 sec. The drop at time point 18 sec is minimal and cannot be observed on the ***cwnd*** curve. This time point is marked as (*) in the following plots of ***RTT*** and ***RTO***. At the drop points, the ***RTO*** counter is reset to 1 segment. Also, these time points appear on the ***RTT*** curve as *spikes*, and mean *lost* packets.

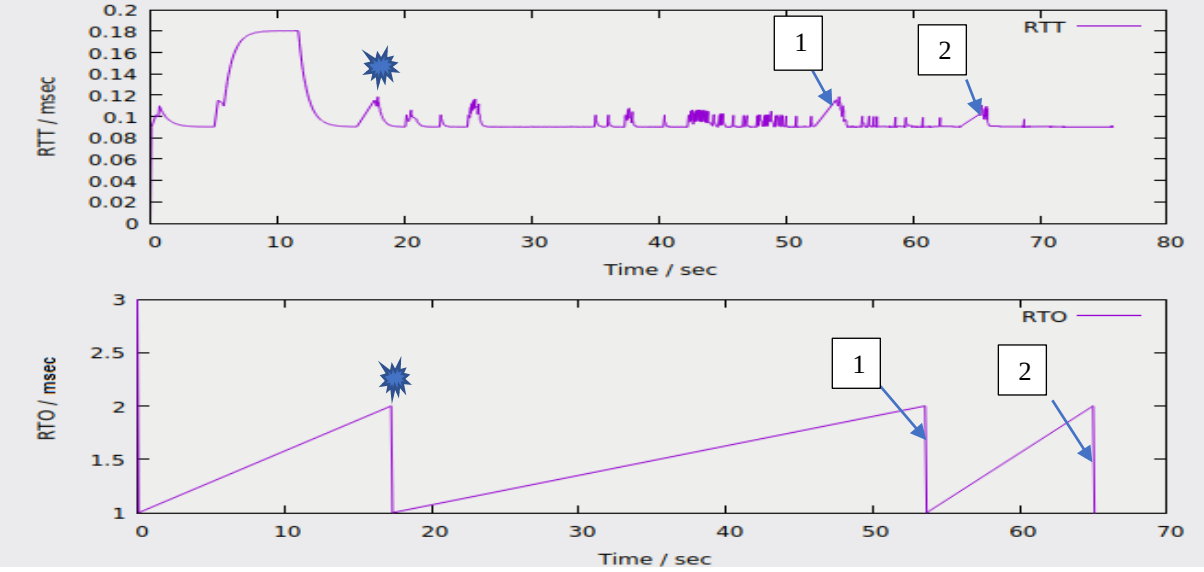


Figure 4-6: Drops of CWND as Spikes on RTT Curve

The next property of TCP DCM+ is *burst transmission*, which is visible in figure 4-7. It is the reason for the high throughput and quick delivery of data by this approach. It is presented as a relationship between sent bytes (sender) and received bytes (receiver). We observe the huge number of bytes that are received correctly at each time point.

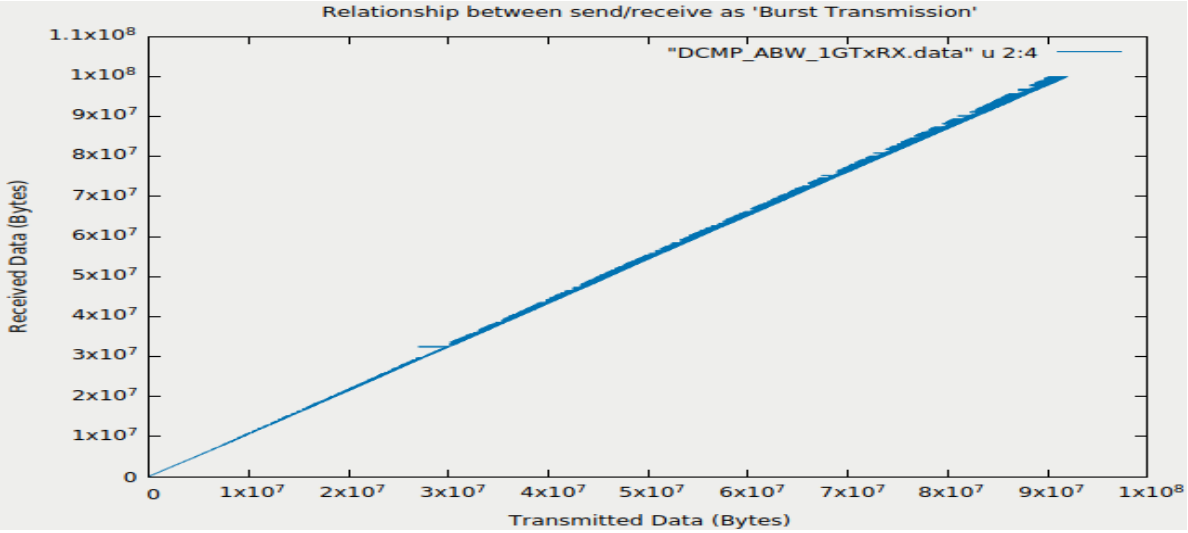


Figure 4-7: Send-Receive Relationship of DCM+ Connection

Figure 4-8 depicts this behavior again in more depth through an enlarged section of figure 4-7.

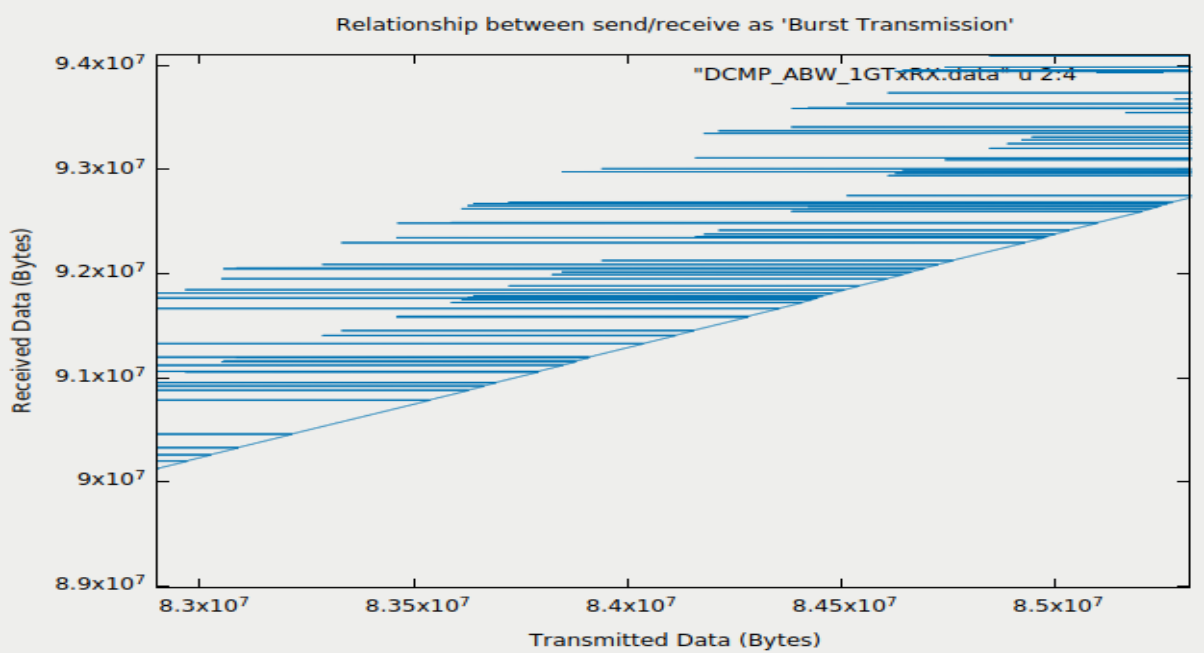


Figure 4-8: Enlarged Section of DCM+ Data Bursts

4.4 TCP SACK Option

This option is placed in the TCP options field inside the TCP header [43]. It was originally mentioned in RFC 1072. Currently, it exists officially as a proposal known as *selective acknowledgment* (SACK) TCP option, which is detailed in RFC 2018. SACKs work by appending a TCP option to a duplicate acknowledgment (DUPACK) packet. The SACK option contains a range of noncontiguous data, that are received successfully. It indicated the sender to send only the missing segments, which are not listed in the SACK option. To enable SACK for a TCP connection, SACK negotiation is required at the beginning of connection between the source and destination. TCP DCM+ benefits largely if SACK option is enabled. Figure (4-9) shows a connection with SACK enabled.

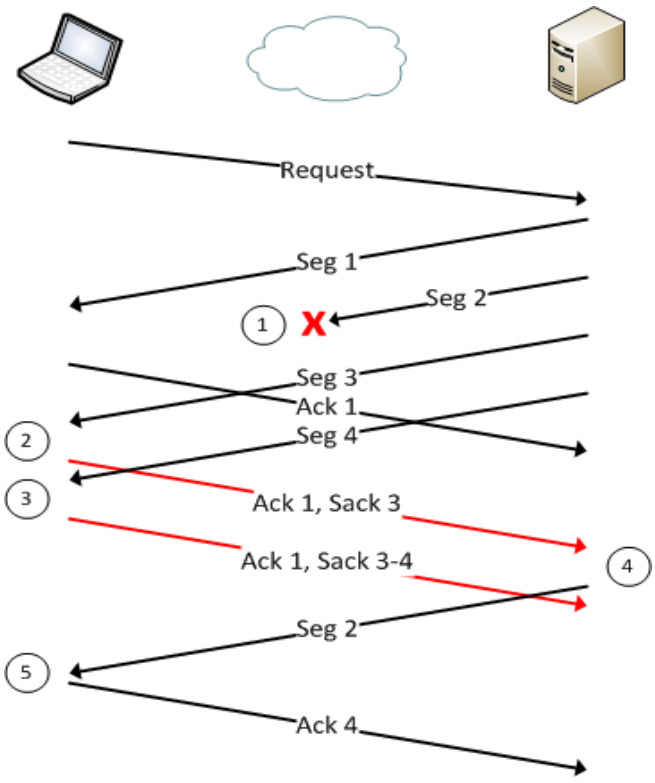


Figure 4-9: TCP Connection with TCP SACK Option Enabled

Figures (4-10), (4-11) and (4-12) compare 3 cases where the sizes of sent data are small (10 MB), mid (100 MB) and large (512 MB). The simulations are

done for TCP SACK enabled and disabled. The benefits are visible when TCP SACK option is enabled, especially when large and mid-size amount of data is transmitted. Both throughput and transmission time are improved. Large size files are sent much quicker (only 44% of the required time) and with very little drops (< 5% of the case without SACK), as depicted in figure 4-10, while mid-size files as in figure 4-11 have a little longer time (130%) to transmit the file.

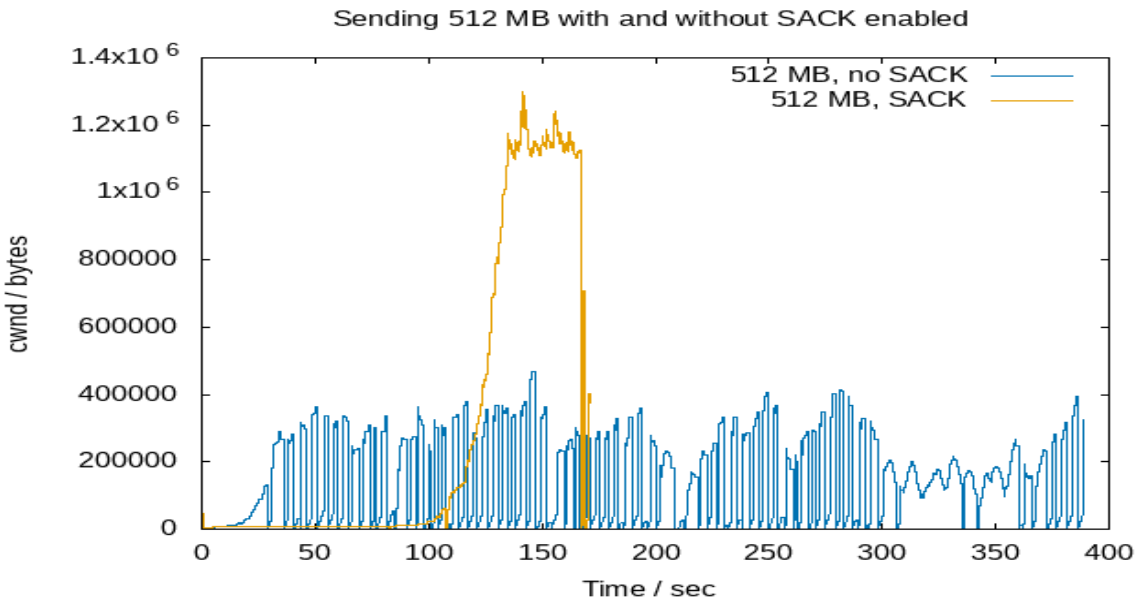


Figure 4-10: Sending a Large File (512 MB) with TCP DCM+

However, mid-size files with the SACK option enabled suffer only (< 7%) window drops compared with SACK not enabled.

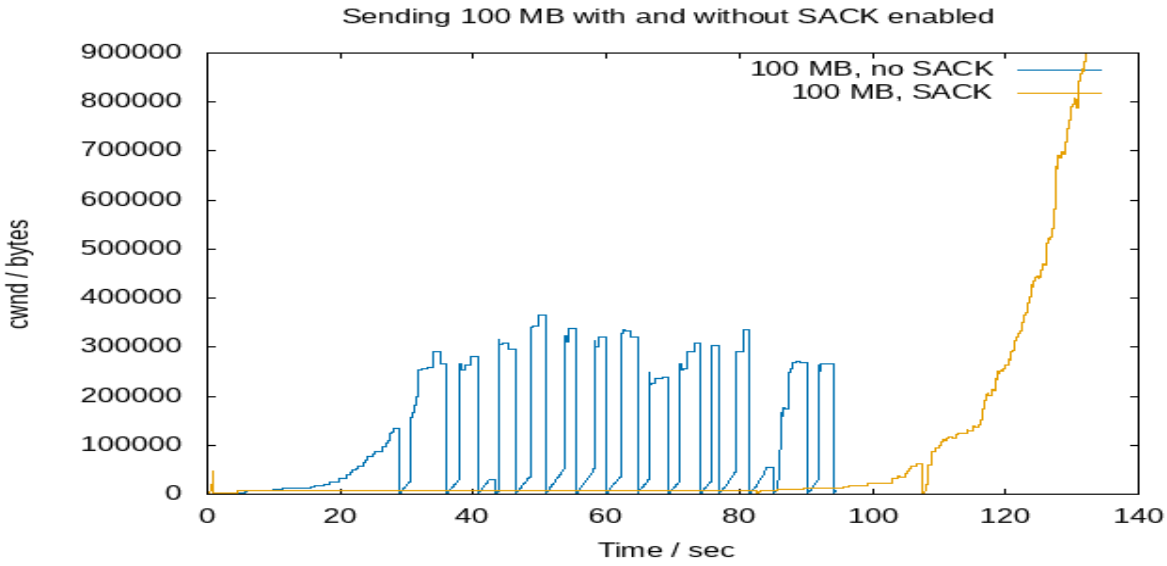


Figure 4-11: Sending a Mid-size File (100 MB) with TCP DCM+

As shown in the figure 4-12, small files don't benefit largely from SACK option. It may be recommended to disable SACK if channel conditions are good and the size of sent data is small.

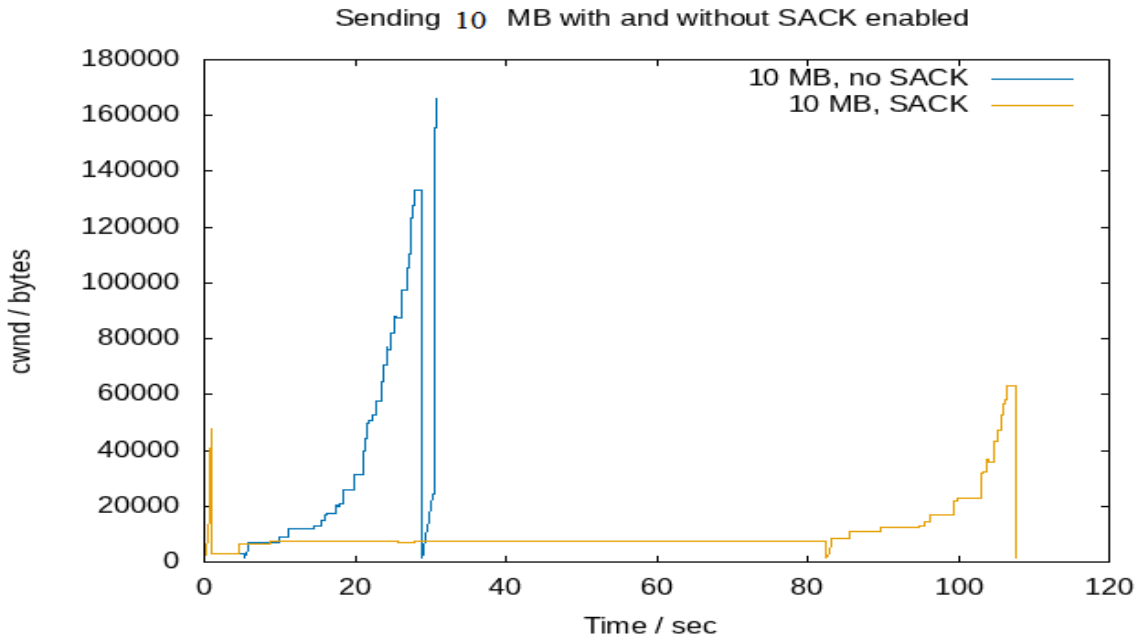


Figure 4-12: Sending a Small-size File (10 MB) with TCP DCM+

4.5 Summary

We presented a new approach, that finds its roots in many approaches like TCP NewReno, TCP Westwood+ and TCP WELCOME. DCM+ is designed like a standard TCP NewReno, and consists of 4 phases. It makes use of bandwidth estimation like TCP Westwood+. The behavior of congestion avoidance is added and overloaded in TCP DCM+ to modify its behavior as needed. The behavior of DCM+ during the ***CA*** phase enforces the ***cwnd*** to track ***ssthresh***, and it never exceeds it. Hence, we made the *claim*, that congestion events are minimized largely. The *only* drops of window size are the results of bad wireless links, which are the reason for lost packets. We also presented its

properties like *burst transmission* and high throughput, robustness and transmission speed.

Chapter Five: Simulation Results and Analysis

Content

5.1 Introduction

The simulations in this chapter clearly show the improvements achieved by TCP DCM+ like stability, robustness and short transmission time. Through intensive simulations we found, that DCM+ also shows improved metrics like high throughput, low average delay, minimized lost packets. Also, we observed no ***cwnd*** drops, except the lost packets caused by bad wireless channel. All these properties make DCM+ competitive against other TCP protocols.

5.2 Simulation Environment

As a simulator, we used the network simulator (*ns-3.29*), which is a discrete-event simulator (DES). Contrary to *ns-2*, *ns-3* is C++-only simulator, which makes debugging much easier than in *ns-2*, which for simulating a scenario applies (C++/TCL) bi-language. Some people find it inconvenient to learn and program with TCL. *Ns-3*, however, has an optional python interface, which makes it possible to invoke the C++ code from python. Yet, python bindings (interfaces) for *ns-3* are a work in progress, and some limitations are known by developers. Another important difference, is that *ns-3* has an emulation mode, which allows for the integration with real networks. Also, *ns-3* came to resolve the problems already existing in *ns-2*. To analyze the results, we used many tools like *NetAnim*, which is the main tool for the analysis in *ns-3*. Other tools like *Gnuplot*, *TraceMetrics* or *Wireshark* have been applied depending on the file type, that we wish to analyze.

As *ns-3* [44] is primarily developed on GNU/Linux platforms, we used the virtualization software *VirtualBox* 5.22 [45], later updated to 6.0, in order to

install *Ubuntu Linux* 18.04 as a guest OS, while MS Windows 7 SP1 ultimate is used as a host OS. More details about installing last version of *ns-3* can be found under the simulator's main site.

5.3 Performance Metrics

While trying to troubleshoot network problem like degradation or outage, measuring network performance is crucial to determine when the network is slow and what are the roots for the problem (e.g., bandwidth outage, misconfiguration saturation, network device defect, etc.). These needed indicators are usually called metrics and are required to produce tangible figures when analyzing network performance. For the reason of detailed comparison, we introduced a new metric, which we called *normalized advancing index* (NAI). It is defined as the ratio of throughput divided by the product of lost packets (given in bytes) and error rates. Its unit is (1/second), and should indicate the speed of delivering the complete size of data from one end to the other despite the existence of lost packets at a specific error rate. The figures produced in this chapter are created according to the following equations:

$$\textbf{\textit{Throughput}}\ (\textbf{\textit{Kbps}}) = \frac{\textit{Total Number of Bytes Transmitted}}{\textit{Time Interval}} \tag{5.1}$$

$$\textbf{\textit{Utilization}}\ (\%) = \frac{\textit{Databits} \times 100}{\textit{Bandwidth} \times \textit{Interval}} \tag{5.2}$$

$$\textbf{\textit{Loss rate}}\ (\%) = \frac{\textit{Total packets dropped}}{\textit{Total packets sent}} \tag{5.3}$$

$$\textbf{\textit{NAI}} = \frac{\textit{Throughput}}{(\textit{LostPackets} * \textit{MTUsize} * \textit{ErrorRate})} \tag{5.4}$$

$$\textbf{\textit{CTT}} = \textit{Last\_Ack\_Time - First\_Ack\_Time} \tag{5.5}$$

The figures in this chapter are shown for different values of performance metrics for different TCP congestion control protocols (DCM+, Westwood+, NewReno, BIC, Ledbat and Hybla). The simulations are executed for different packet error rates. The simulation environment is a virtual machine of ns-3.29 under Linux Ubuntu. The VM is inside Oracle VirtualBox 6.1.x. The following parameters are constant through this thesis:

Table 5.1: Simulation Environment Parameters

Data size	Bottleneck BW	Access BW	MTU Size	Duration (sec)
100 MB	100 Mb/sec	1000 Mb/sec	1500 Bytes	5000

5.3.1 Throughput

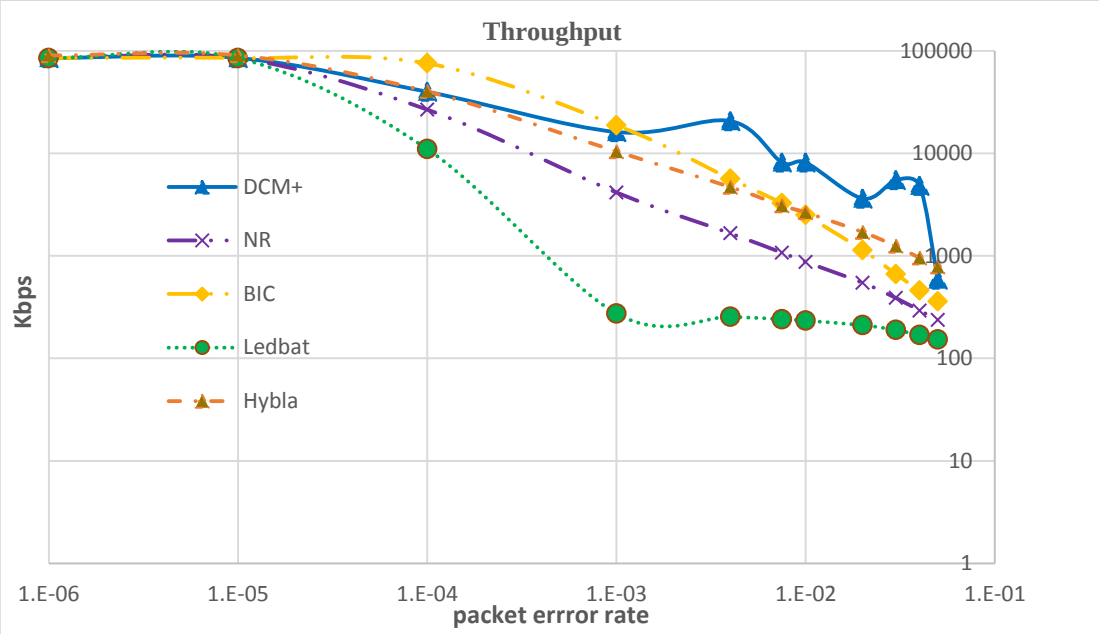


Figure 5-1: Comparing Throughput of DCM+ with Different TCP Protocols

In figure 5-1, we see the plots of the throughput for different protocols, and we clearly see the advantage of DCM+ over most other protocols. This high throughput extends nearly over the most range of error rates, which is from 1e-6 to 0.05. For error rates less than 1e-3, we see that only BIC protocol performs better, but that is at the expense of other metrics like packet delivery ration

(PDR), average delay and normalized advancing index (NAI), where BIC performs worst.

5.3.2 End-to-End Average Delay

Figure 5-2 shows the average delay for the different protocols. Our approach is among the few protocols with least average delay in all tested cases. BIC protocol has the highest E2E delay. We conclude, that DCM+ does not cause long queues (full buffers) in the intermediate routers.

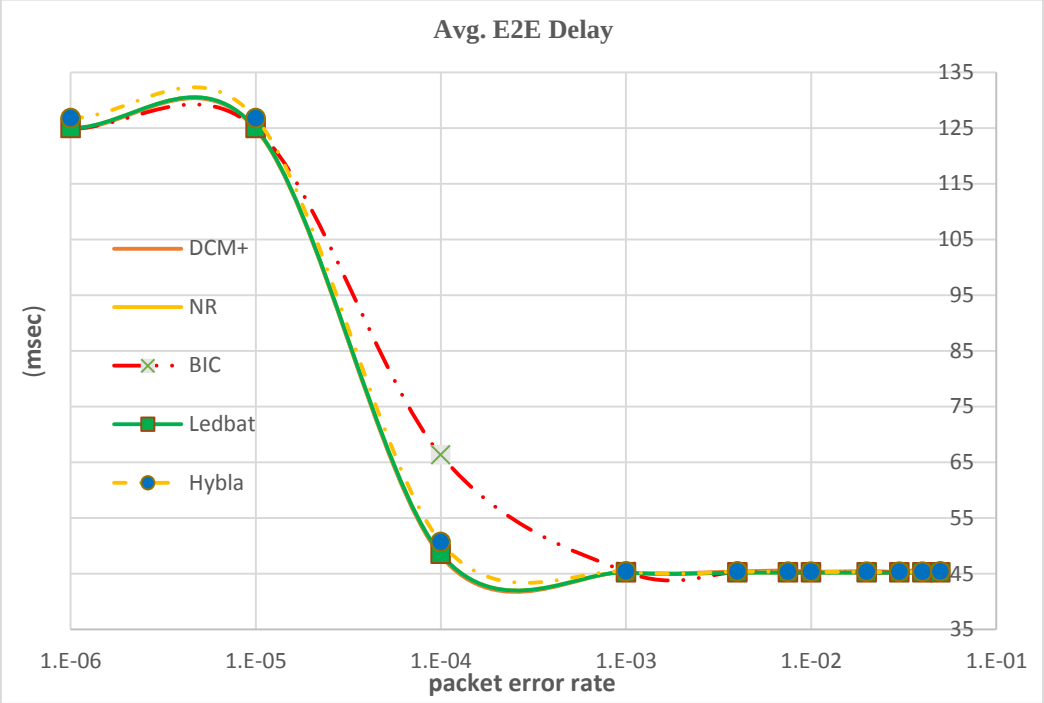


Figure 5-2: Comparing Delay of DCM+ with Different TCP Protocols

5.3.3 Packet Delivery Ratio (PDR)

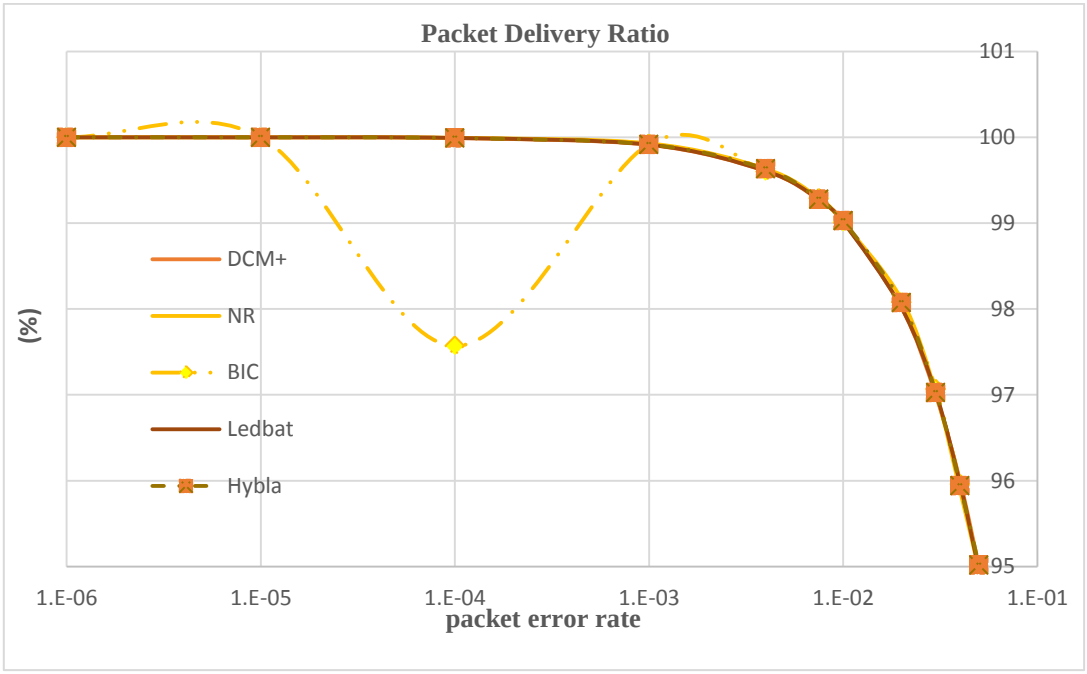


Figure 5-3: Comparing PDR of DCM+ with Different TCP Protocols

In figure 5-3, it is clear that DCM+ has very similar PDR-behavior like other protocols. It reaches values of 99.998 % for low error rates (1e-6), and has like Hybla the best performance for high error rates (< 0.05).

5.3.4 Normalized Advancing Index (NAI)

NAI can be considered as a robustness measure. From figure 5-4 we see, that DCM+ performs better than all other protocols in this research study. It is obvious, that DCM+ has the best results for all used error rates. This reflects the highest speed and best quality for TCP systems and applications.

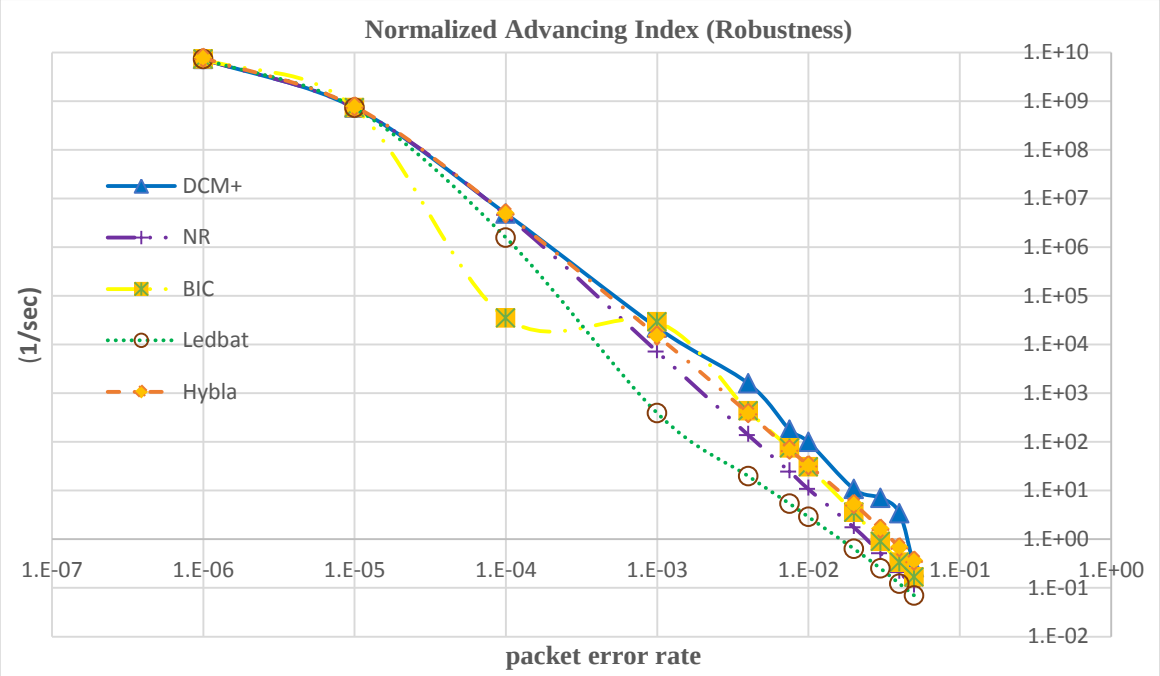


Figure 5-4: Comparing NAI (Robustness) of DCM+ with Different Protocols

5.3.5 Complete Transmission Time (CTT)

CTT is defined as the time needed for the last ACK segment to arrive at the sender. When transmitting data, it is desired to finish transmission in short time without causing congestions. We claim, that this is the case for DCM+ protocol as depicted in figure 5-5. It has the lowest (***CTT***) among all tested protocols. Based on our results, TCP DCM+ applications and devices can extremely speedup data transmission, hence finish using the link earlier, which results in less power consumption.

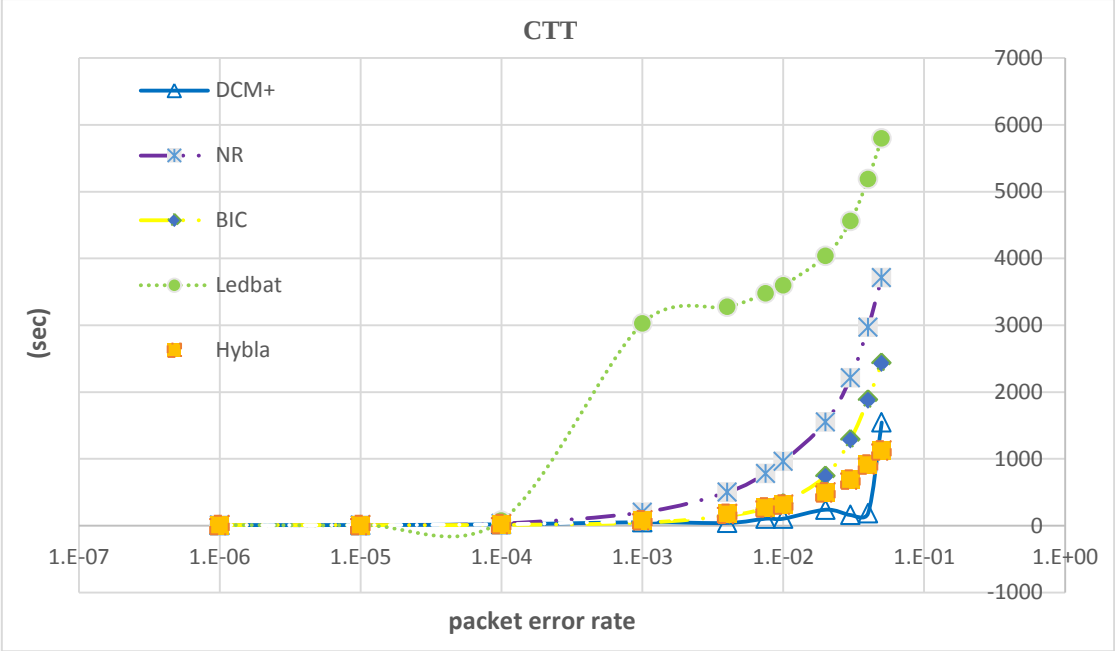


Figure 5-5: Comparing CTT of DCM+ with Different TCP Protocols

5.3.6 Packet Losses

The total number of lost packets during transmission depends on many factors like bandwidth between routers, bandwidth used from destination node, MTU size, error rate and TCP buffer size. We see in figure 5-6, that DCM+ has beside Hybla the lowest percentage of lost packets.

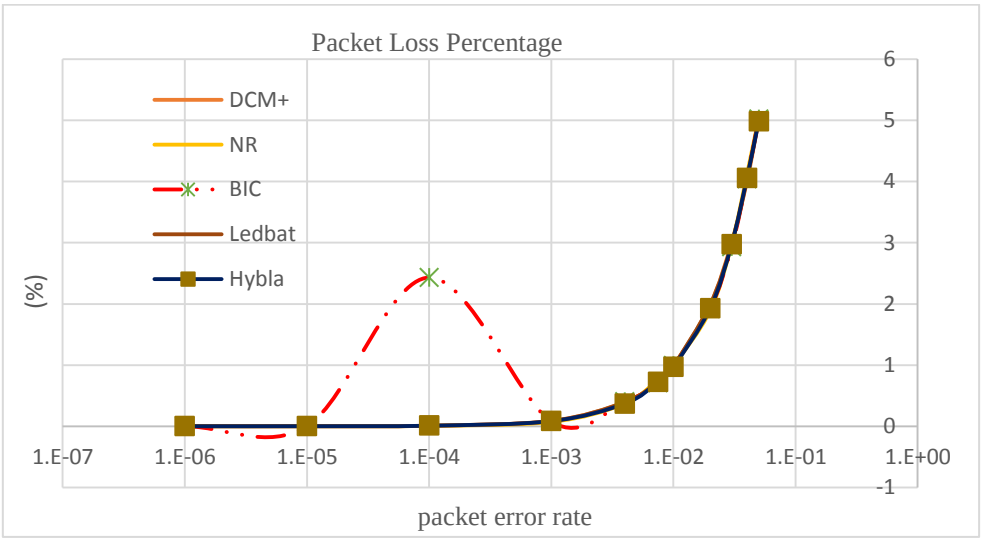


Figure 5-6 Comparing Packet Losses of DCM+ with Different TCP Protocols

5.4 Effect of TCP Buffer Size on the Window Size of TCP DCM+

The plots in figure 5-7 are taken for different sizes of TCP buffer. The simulations are shown for the following sizes: 4MB, 2 MB, 1 MB, 512 KB, 256 KB, 128 KB, 64 KB, 32 KB and 16 KB. It is visible, that depending on these different sizes, the number of drops and window sizes are also different. The parameters used in figure 5-7 are shown in table 5-2.

Table 5-2: Simulation Parameters for Different TCP Buffer Sizes

Parameter	MTU	Packet Error Rate	Access BW	Bottleneck BW
Value	750 bytes	0.01	10 Gbps	100 Mbps

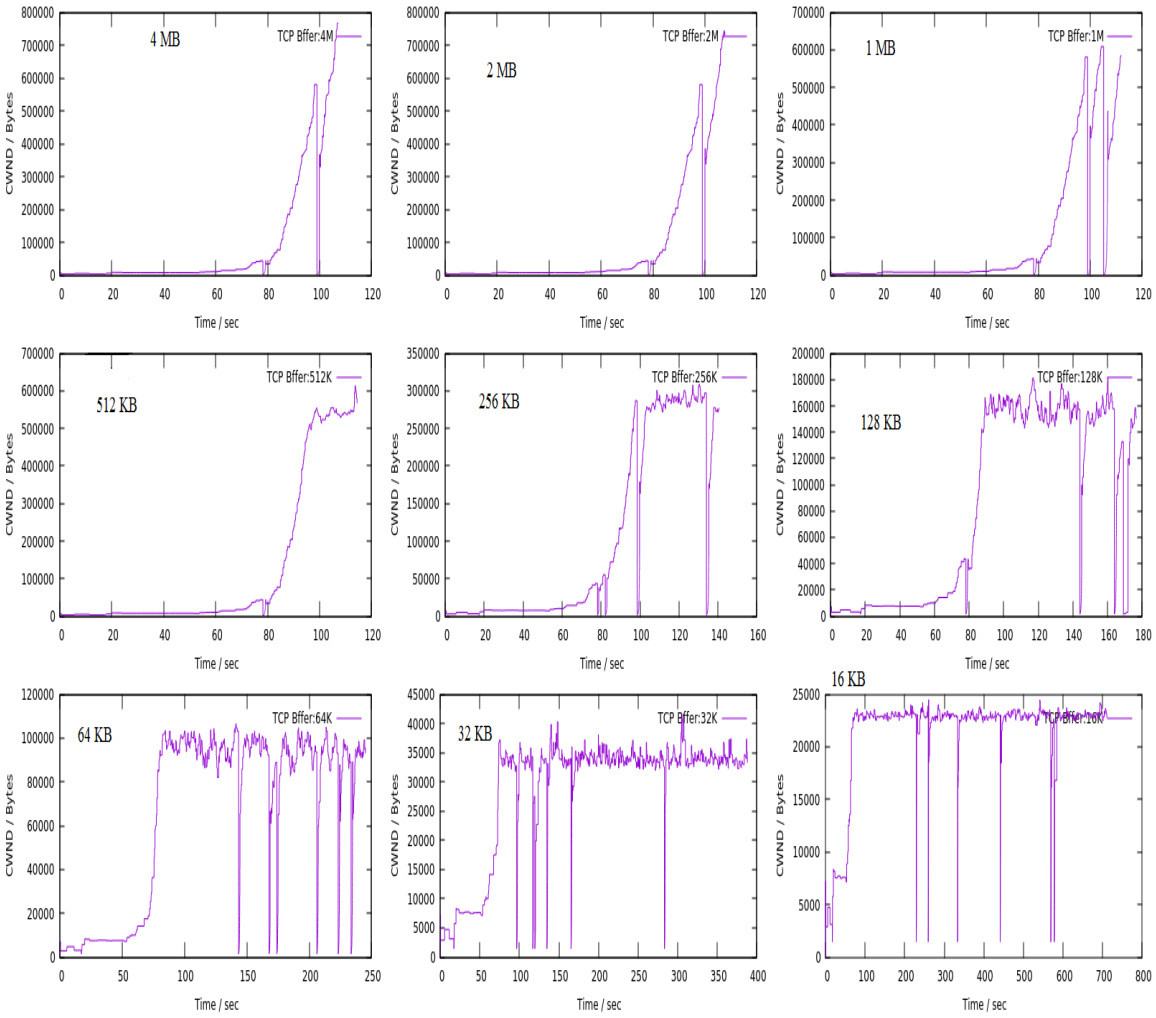


Figure 5-7: Effect of TCP Buffer Size on the DCM+ Connection

Figure 5-7 shows, that more time is needed for the transmission with decreasing the TCP buffer size. The number of lost packets (window drops) increases. Also, the window size and throughput decrease. Figure 5-8 below shows the robustness (NAI) of the DCM+ transmission for different buffer sizes. With increased buffer size, the robustness increases until we reach a maximum at the size equal 4 MB.

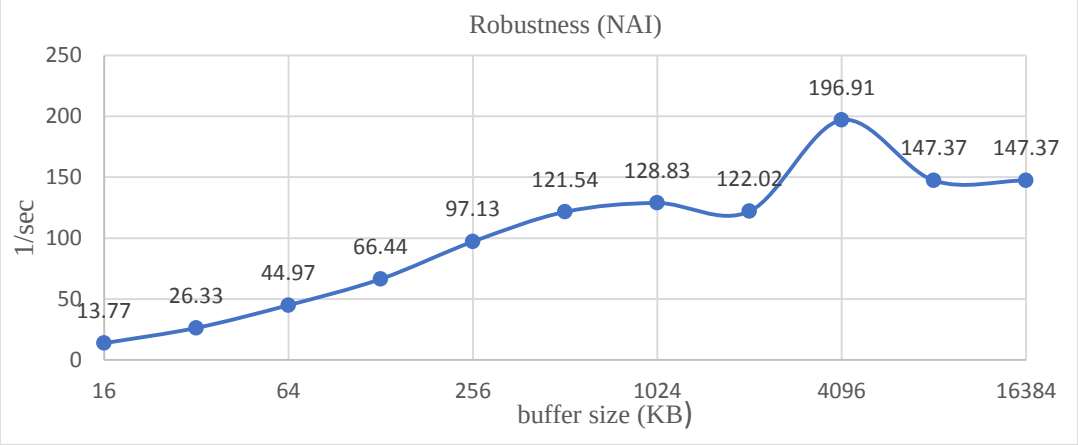


Figure 5-8: Effect of TCP Buffer Size on Robustness (NAI)

In figure 5-9, we present the number of drops as a function of the TCP buffer size. As shown below, it is decreasing with increased buffer size. At 4 Mb we have a minimum of just 2 drops.

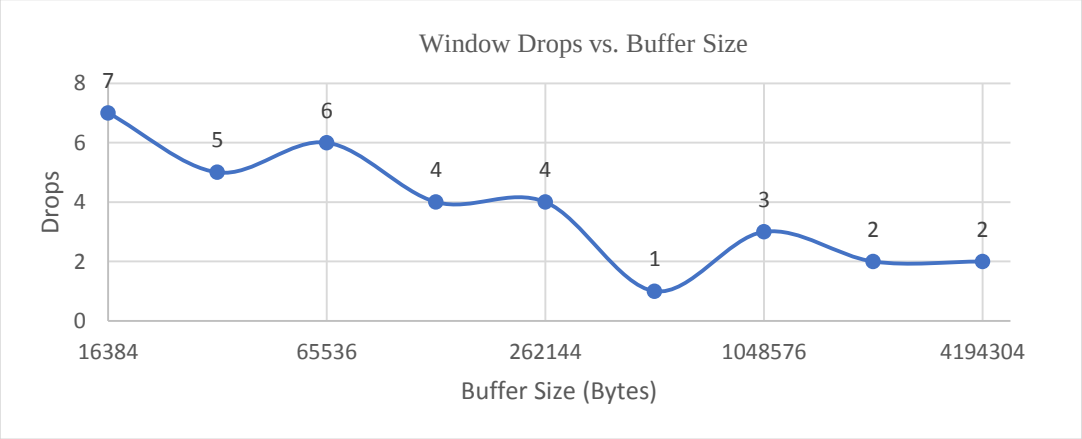


Figure 5-9: Effect of TCP Buffer Size on the Window Drops

The buffer size has a big influence on ***CTT***. We see from figure 5-10, that it is exponentially decreasing with increased buffer size. The sizes greater than 512 KB deliver the lowest transmission times.

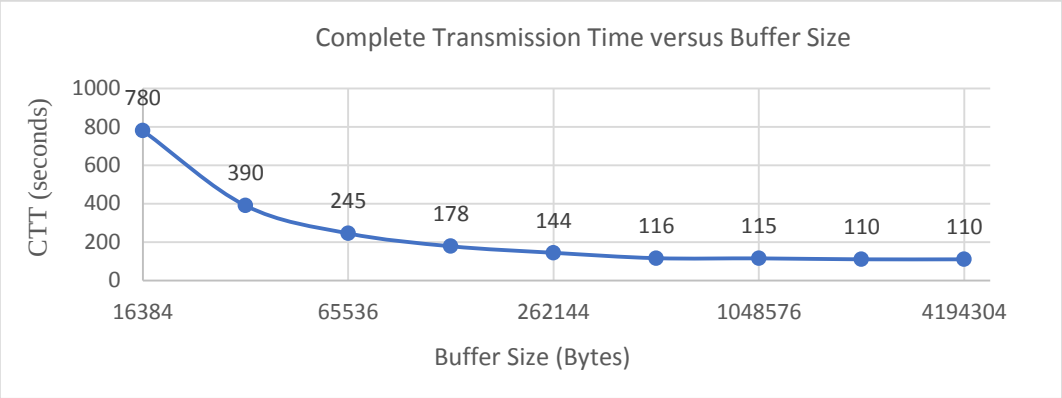


Figure 5-10: Effect of TCP Buffer Size on the Transmission Time (CTT)

The throughput of TCP DCM+ connection as a function of the TCP buffer size is sketched out in figure 5-11. As shown, it reaches a maximum of 16.5 Mbps at 512 KB. With increased sizes, the throughput fluctuates around 13 Mbps.

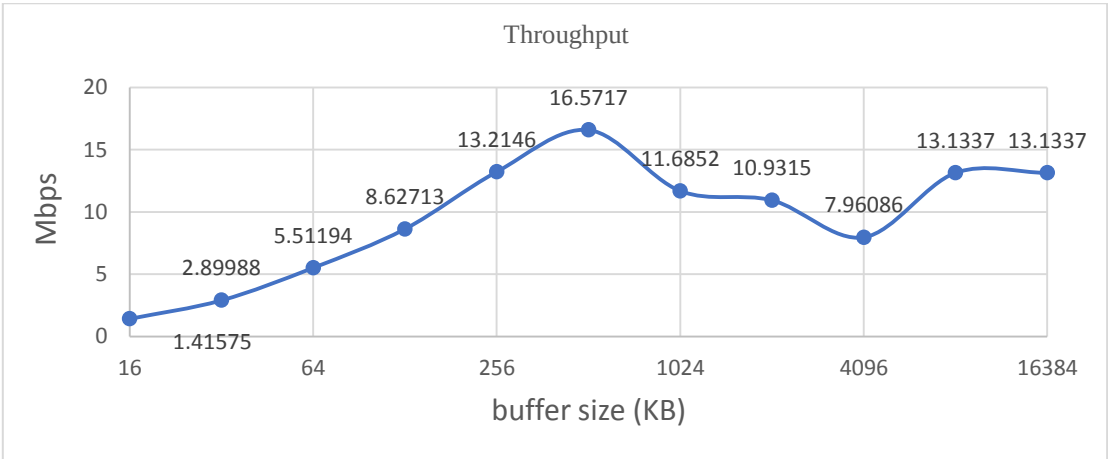


Figure 5-11: Effect of TCP Buffer Size on the Throughput of DCM+

The total number of lost packets during the TCP DCM+ transmission is given in figure 5-12. At 512 KB buffer size, we have the highest throughput and

also packet losses. Decreasing these losses is achieved by increasing the buffer size. From figure 5-12, it is visible, that the least losses are achieved when the buffer size is 4 MB.

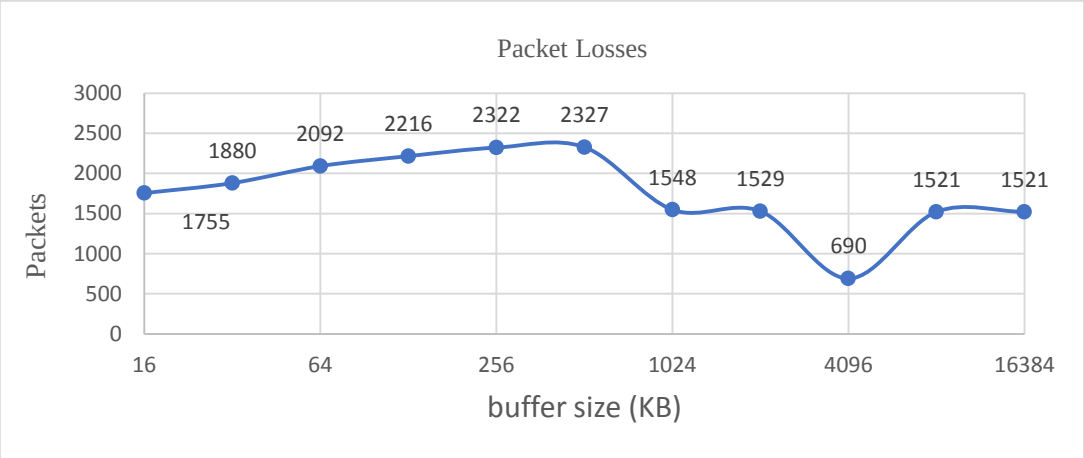


Figure 5-12: Effect of TCP Buffer Size on the Packet Losses

5.5 TCP DCM+ versus TCP Vegas

TCP Vegas is a technique that reacts to queueing delay. The main idea behind this method is as follows: if the queueing delay is large, then decrease the window size, else increase it [46]. Figures 5-13 (a) and (b) present a comparison between TCP Vegas and TCP DCM+ for 6 different cases with random packet error rates, but the same buffer size in all cases. The measurements of this section are displayed in appendix C, table C-8.

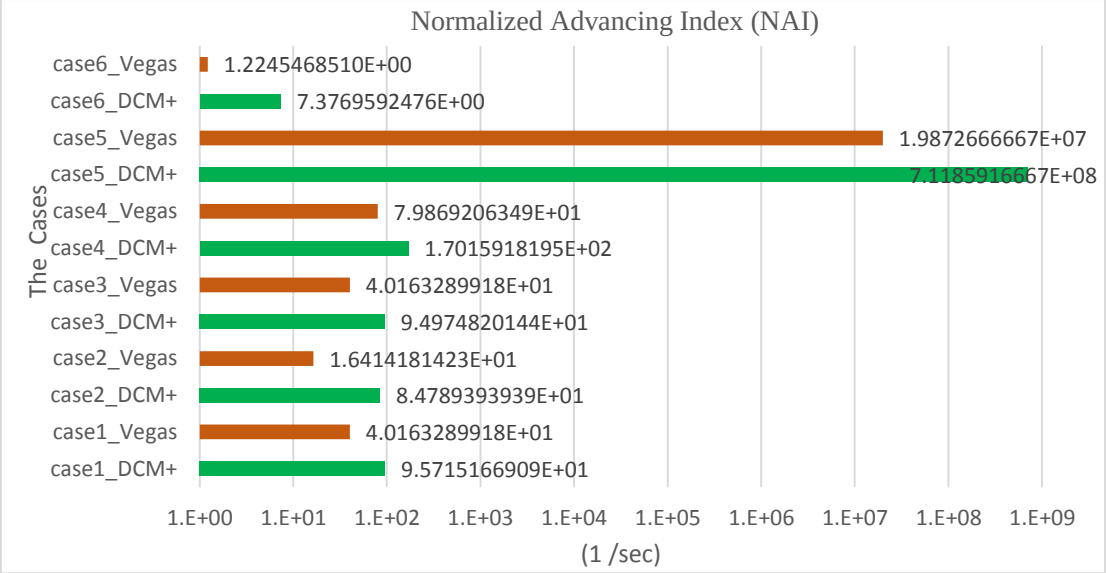


Figure 5-13 (a): Comparing robustness (NAI) for DCM+ vs. Vegas

For each of the following cases, the parameters for both DCM+ and Vegas are equivalent. In figure 5-13 (a), the comparison is made for robustness (i.e., NAI), while in figure 5-13 (b) we compare the transmission time (CTT).

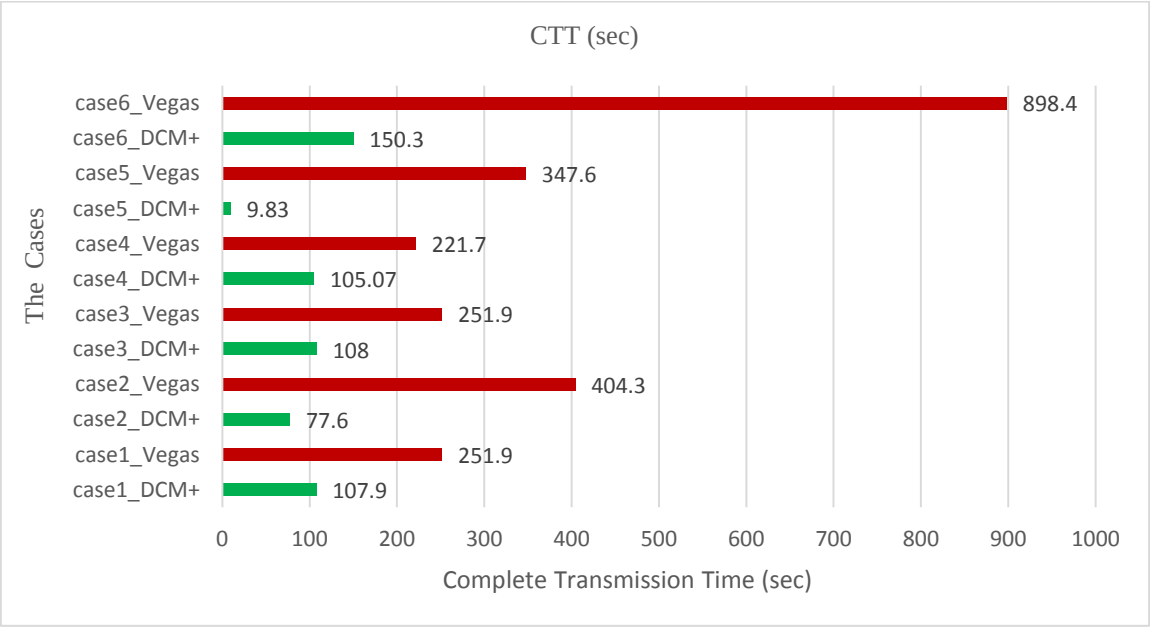


Figure 5-13 (b): Comparing CTT for DCM+ vs. Vegas

In the following table 5-3, we have written down the ratios (TCP DCM+ / TCP Vegas) for robustness (r_{NAI}) and transmission time ($r_{CTT)}$. From this table, we have the conclusion, that the product of both ratios in all cases equal "1". That means, if we know the transmission ratio, then we can calculate the robustness ratio, and vice versa, but it is still not clear, if this rule holds for other approaches and topologies.

Table 5-3: Improvements of TCP DCM+ against TCP Vegas

	Case 1	Case 2	Case 3	Case 4	Case 5	Case 6
NAI ratio	2.38	5.16	2.36	2.13	35.82	6.02
CTT ratio	0.43	0.19	0.43	0.47	0.028	0.17
Product	1.02	0.99	1.01	1.00	1.00	1.02

$$P = r_{NAI} * r_{CTT} = 1 \quad (5.6)$$

We can also reformulate the above equation to:

$$\frac{\mathbf{NAI_{DCM+}}}{\mathbf{NAI_{Vegas}}} * \frac{\mathbf{CTT_{DCM+}}}{\mathbf{CTT_{Vegas}}} = \mathbf{1} \quad (5.7)$$

Or equivalently,

$$\frac{\mathbf{NAI_{DCM+}}}{\mathbf{NAI_{Vegas}}} = \frac{\mathbf{CTT_{Vegas}}}{\mathbf{CTT_{DCM+}}} \quad (5.8)$$

From the table above, we have the worst-case improvements for both NAI and CTT in case 4, and the best-case improvements in case 5. We conclude, that DCM+ largely improves the TCP transmission. In all cases we clearly identify the supremacy of TCP DCM+.

5.6 Optimizing the Segment Size (MTU)

Optimizing the segments size is crucial before sending data with DCM+. To understand the effect of the MTU optimization, we consider the following cases. The simulations below are done for different packet error rates and MTU sizes. Following two cases are distinguished:

Case 1: Same error rate but different segment sizes.

In figure 5-14, we see, that unwanted drops are occurring as a side effect of changing the ***MTU*** size. In this figure we sent data of the size 100 MB and the error rate is 1e-2. The SACK option is enabled. Here, if ***MTU*** = 1500 bytes, we finish transmission in lower time compared with the case ***MTU*** = 1200. In the first case, it suffers just 1 unwanted drop. On the other hand, if ***MTU*** = 1200 bytes, we have the longest time to finish transmission (***CTT***) and suffers 4 window drops.

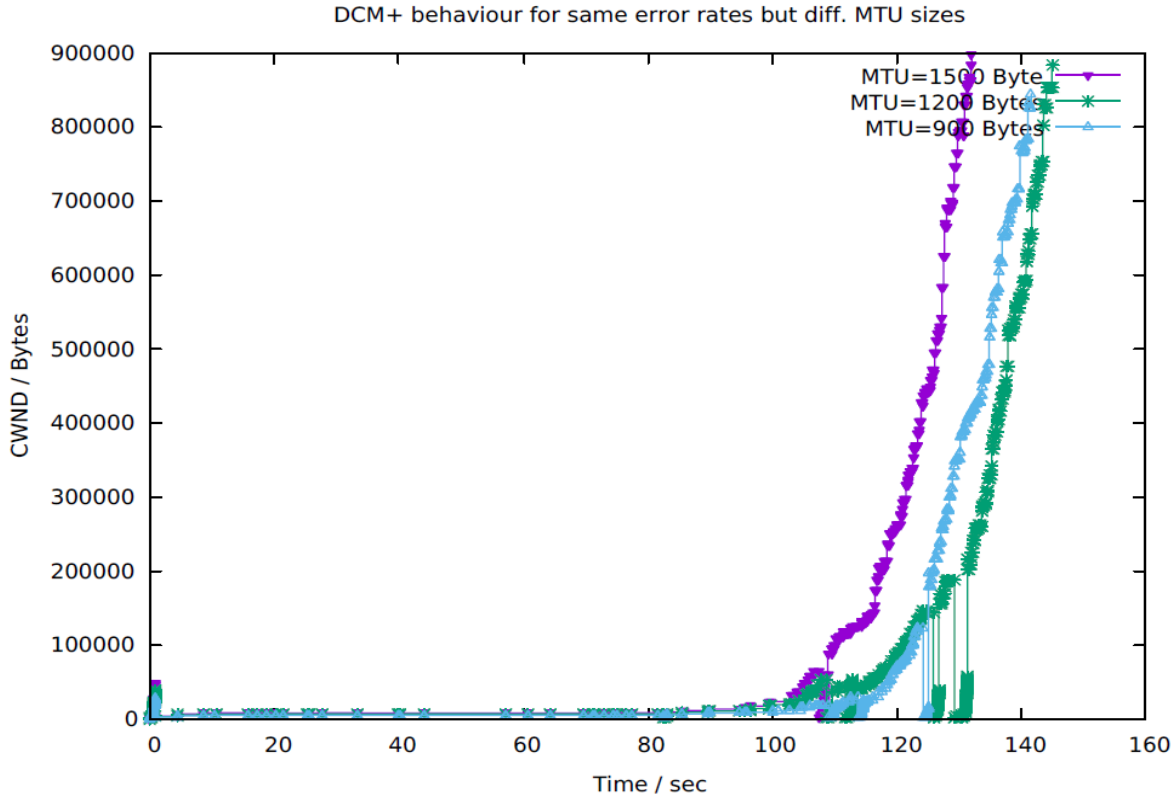


Figure 5-14: Window Dynamics for Different MTU/Same Error Rate

It is also clear, that the performance is best when MTU = 1200 bytes. It finishes transmission faster and suffers no unwanted drops, as shown in figure 5-7. It gives a hint, that an additional increase of the performance of DCM+ may be possible if the used MTU size is optimized before the transmission.

Case 2: Different error rates but same segment sizes

In figure 5-15, the size of data to be transmitted is (256 MB), and the used error rates are: (1e−3, 1e−2). We see, that for the error rate = 1e−3, the transmission is much quicker (***CTT*** is reduced by the half) while the throughput is improved by 1.8 times. Besides, it does not make any false drops.

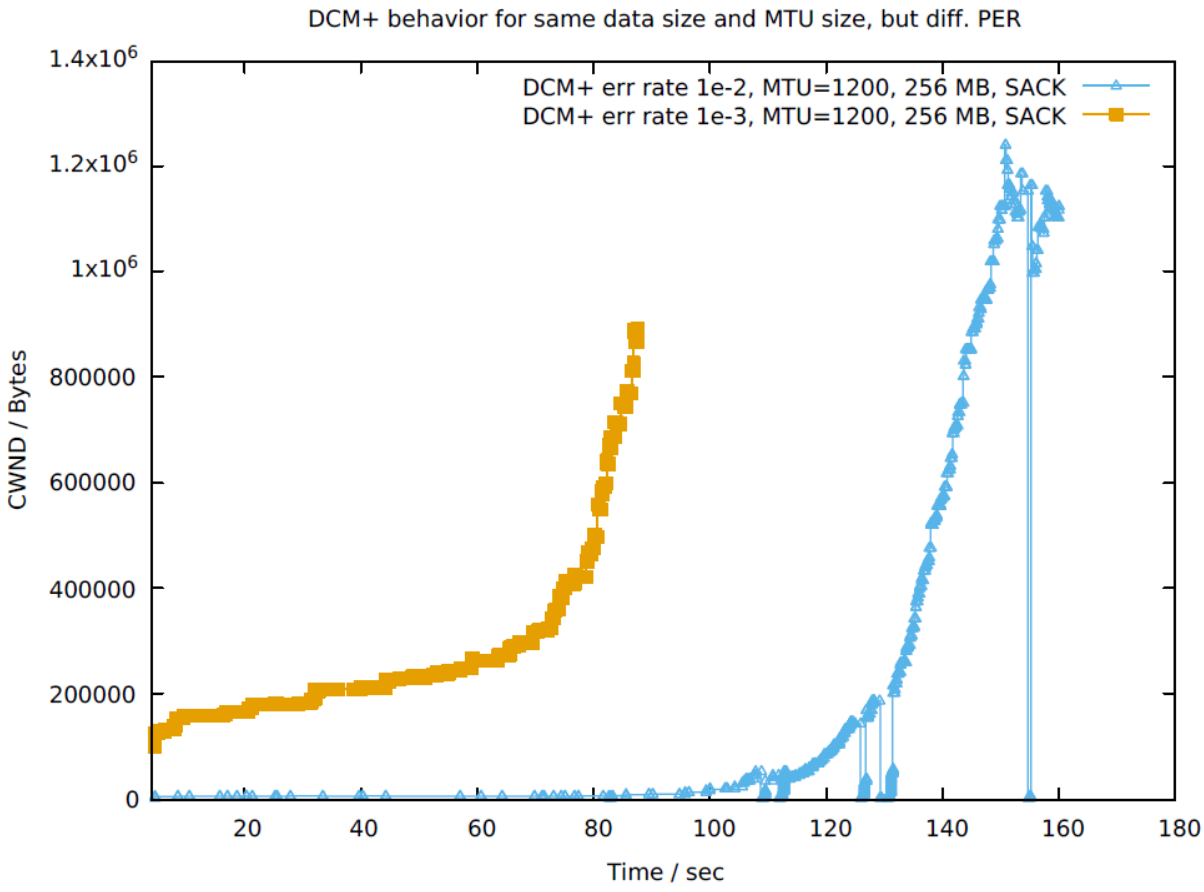


Figure 5-15: Window Dynamics for Different Error Rates/Same MTU

5.7 Impact of Access Bandwidth on the Performance of DCM+

The following parameters are used during the following section.

Bottleneck BW: 100 Mbps; packet error rate = 0.02; Data Size = 100 MB;

MTU = 1500 BYTE; SACK option = true.

Figure 5-16 depicts the network throughput as a function of access BW at the destination node. If the access BW is equal the bottleneck BW, then we have a maximum. Values of access BW, that are greater than the bottleneck BW deliver throughput values that fluctuate around the maximum throughput.

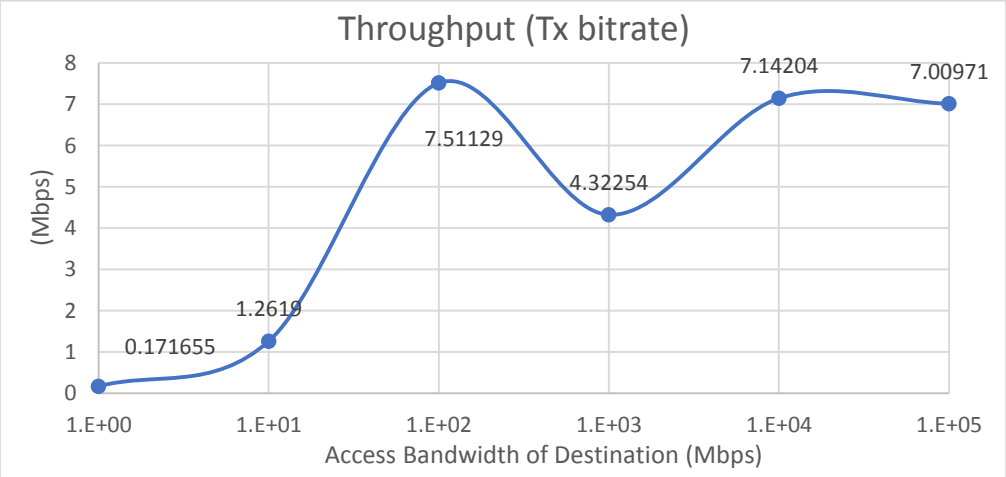


Figure 5-16: Throughput of DCM+ as function of Access BW

Figure 5-17 shows the average delay as a function of access BW. We see, that a minimum delay is achieved for values greater than the bottleneck BW. The improvements in average delay for values much higher than bottleneck are minimal.

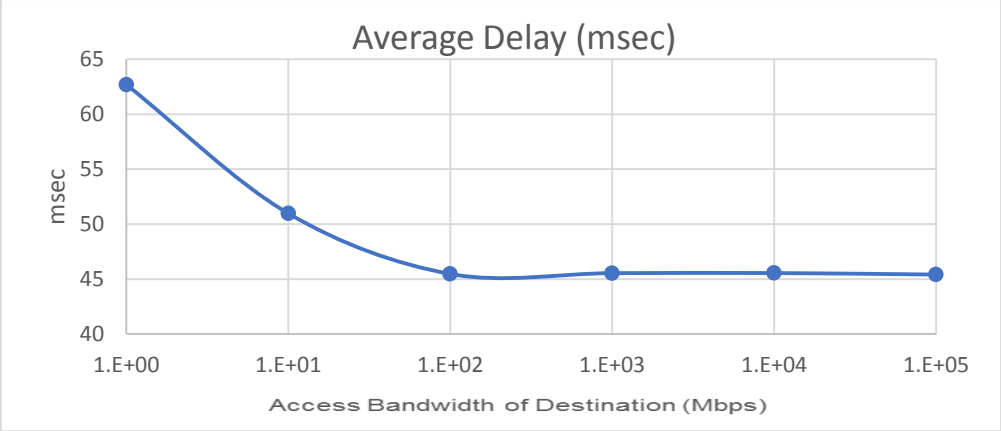


Figure 5-17: Avg. Delay of DCM+ as function of Access BW

Decreasing the average delay is possible as long as the BW at the destination is smaller than the existing bottleneck BW. Here, the maximal decrease in average delay is 1.17 msec per Mbps in the interval 1 to 10 Mbps. If the used BW at the destination is much larger than the bottleneck, then the minimization of the delay is negligible and equal 0.01 msec per Mbps.

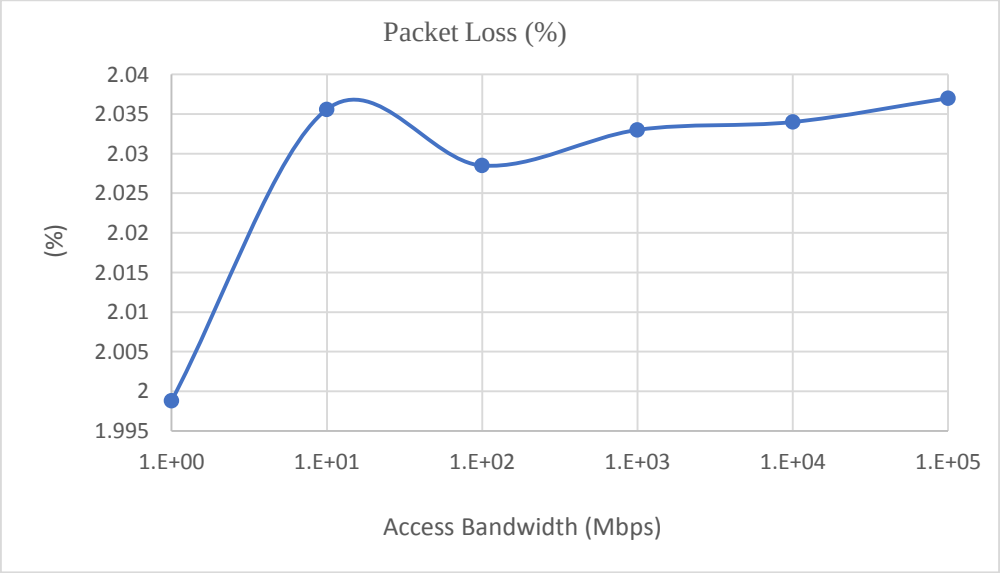


Figure 5-18: Packet Losses Percentage of DCM+ as function of Access BW

The packet losses as a function of access BW is shown in figure 5-18. As the throughput increases for access BW greater than bottleneck BW, it is clear that the losses will also increase.

5.8 Impact of Bottleneck Bandwidth on the Performance of DCM+

The following parameters are used during this section.

Access BW: 1 Gbps; error rate = 0.02; Data Size = 100 MB;
MTU = 1500 byte; SACK = true

The maximal throughput as previously said is if the BW of the bottleneck is equal the destination BW. The maximum is at 1 Gbps as shown in figure 5-19. The improvements are minimal for higher values of bottleneck BW.

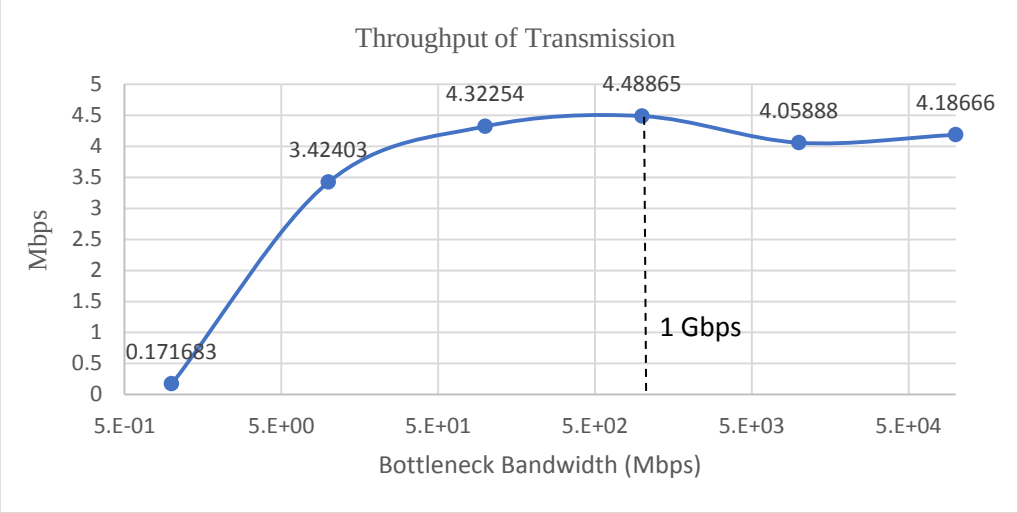


Figure 5-19: Throughput of DCM+ as function of Bottleneck BW

Figure 5-20 shows the delay as a function of bottleneck BW. We see very similar characteristic as in figure 5-17, but the highest improvement of delay minimization is nearly 0.87 msec per Mbps increase in the bottleneck BW. Values of bottleneck BW higher than the used access BW have no impact on delay.

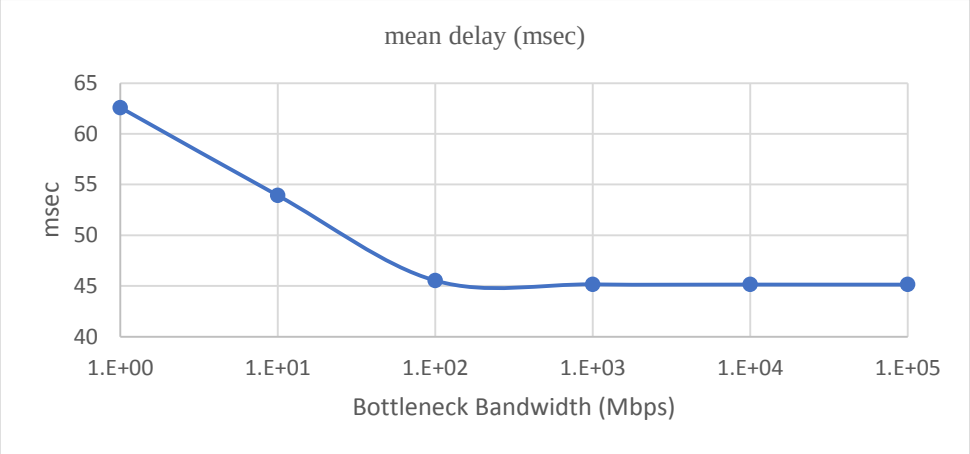


Figure 5-20: Avg. Delay of DCM+ as function of Bottleneck BW

In figure 5-21, we have a look at the impact of increasing the bottleneck BW on the packet losses. In this figure, we have a decrease of losses as the bottleneck BW increases. However, the losses are minimal as the difference between maximal (2.08 %) and minimal (1.98 %) losses is just (0.1 %) in the range 1 Mbps up to 100 Gbps, which means stable and robust transmission.

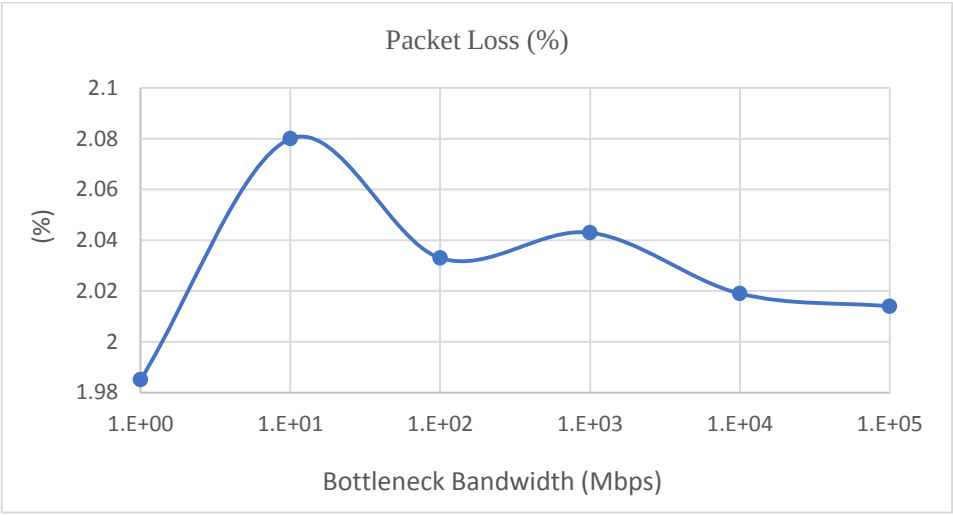


Figure 5-21: Packet Losses Percentage of DCM+ as function of Bottleneck BW

5.9 Properties of DCM+ and comparing with DCM and Westwood+

Figure 5-22 shows the transmission rule 2, where the product $\boldsymbol{P}$ of the ratios $\boldsymbol{r_{NAI}}$ and $\boldsymbol{r_{CTT}}$ is theoretically equal 1. This figure shows rule '2' for five cases that compare DCM+ against TCP Vegas. It is clear that for case one the robustness ratio is ≈5 while the CTT ratio is ≈0.194 and $\boldsymbol{P}$ is ≈ 0.97. For case four it is ≈1.004 .

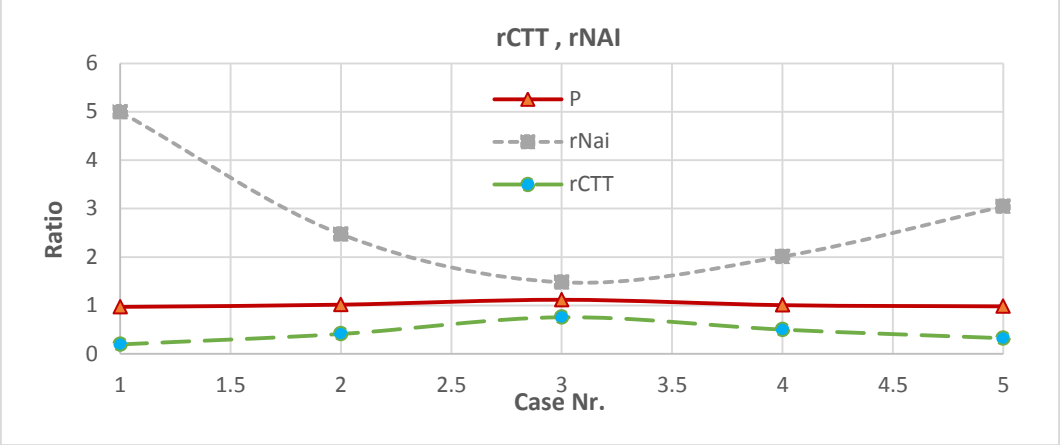


Figure 5-22: DCM+ Rule '2': Robustness Ratio is Inverse Proportional to the Ratio of Transmission Time

Figure 5-23 below shows the robustness curves for DCM+ and Vegas. The ratio of the robustness for case 1 is nearly '5'.

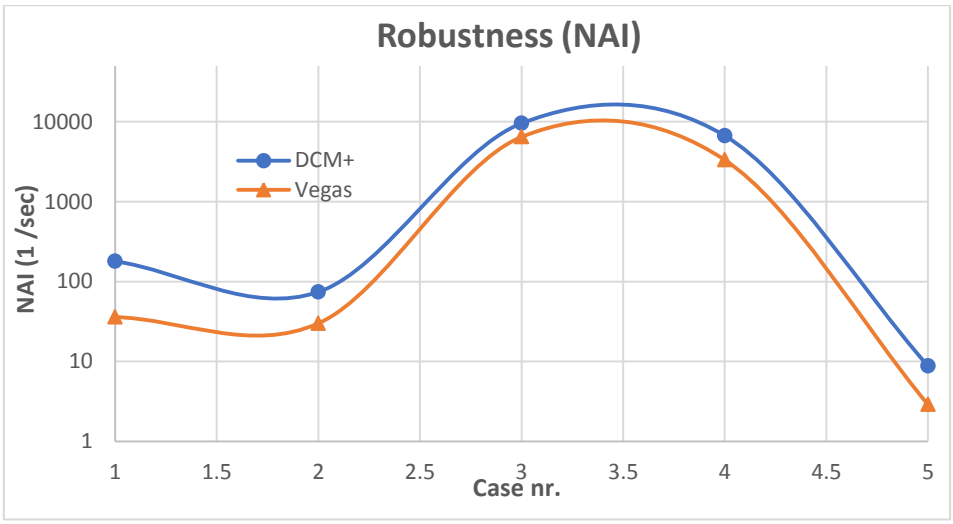


Figure 5-23: Robustness curves for DCM+ and Vegas

Figure 5-24 shows a comparison of the robustness between DCM+ and Westwood+. DCM+ shows the same robustness as Westwood+ for some error rates, which means that DCM+ suffers less packet losses and, hence, better fairness.

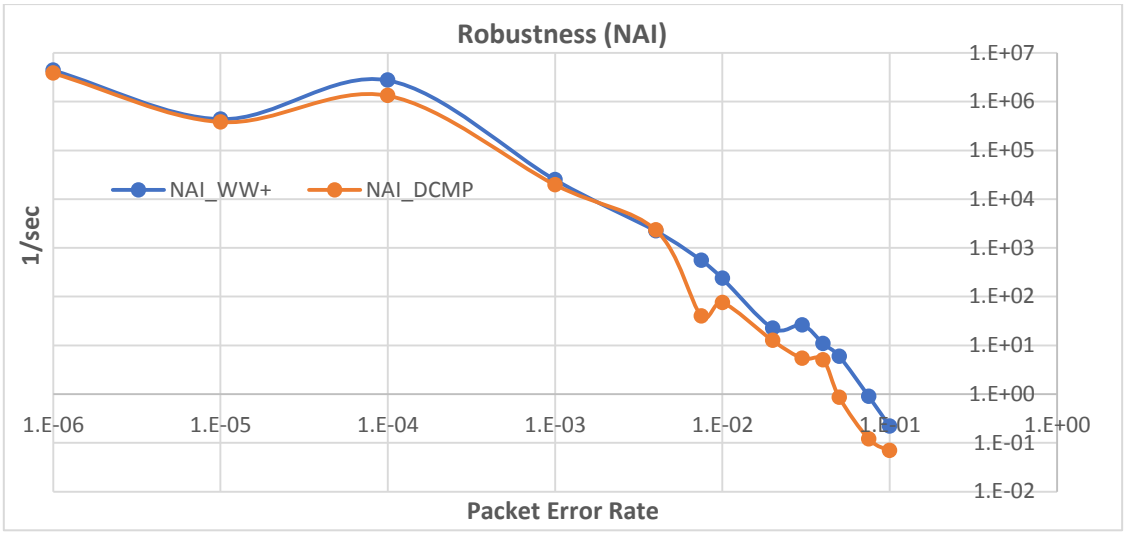


Figure 5-24: Comparison of the Robustness of DCM+ vs Westwood+

Figure 5-25 is about the utilization of DCM+ and Westwood+. We see that DCM+ uses less bandwidth than Westwood+ if the error rate is low. If the link suffers higher losses and needs more bandwidth to achieve the best performance, then DCM+ can make use of a utilization that exceeds Westwood+. This will result in a better fairness than for Westwood+.

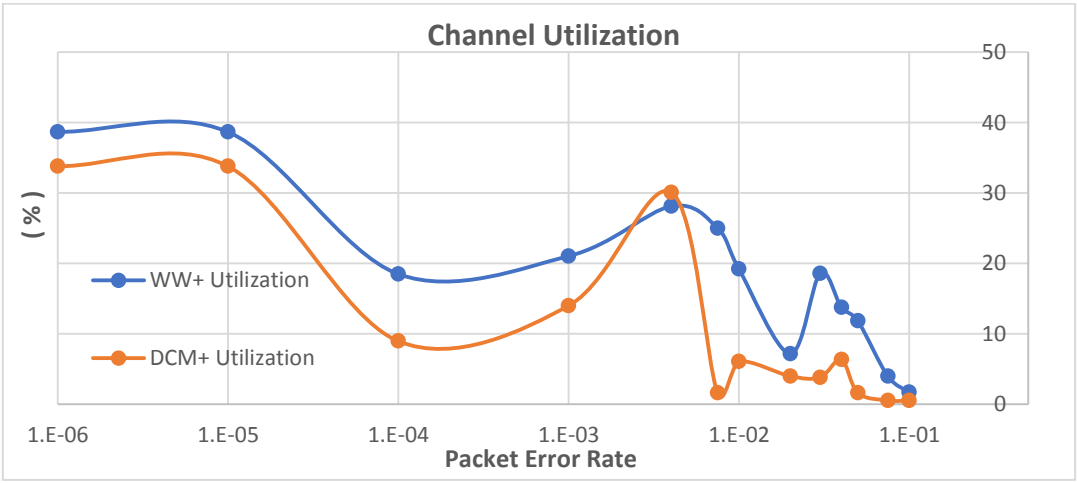


Figure 5-25: Comparison of the Utilization of DCM+ vs Westwood+

Figures 5-26, 5-27, 5-28 and 5-29 shows the superiority of DCM+ over DCM in terms of throughput, complete transmission time, robustness and utilization. From very low packet error rates (1e-6) up to (0.05), DCM+ owns much better throughput values against DCM as shown in figure 5-26.

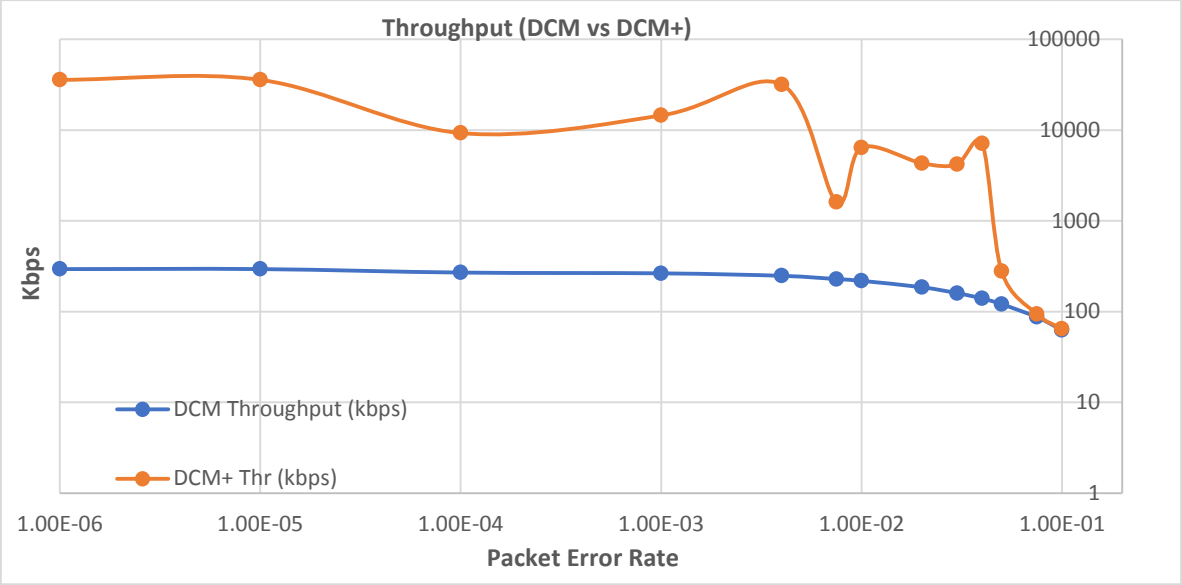


Figure 5-26: Throughput Comparison: DCM+ vs DCM

Figure 5-27 shows that DCM+ under the same transmission parameters owns much shorter transmission times. We see that the error rate 0.04 is the limit for better performance. It is clear that DCM+ is at least 120 times faster than DCM.

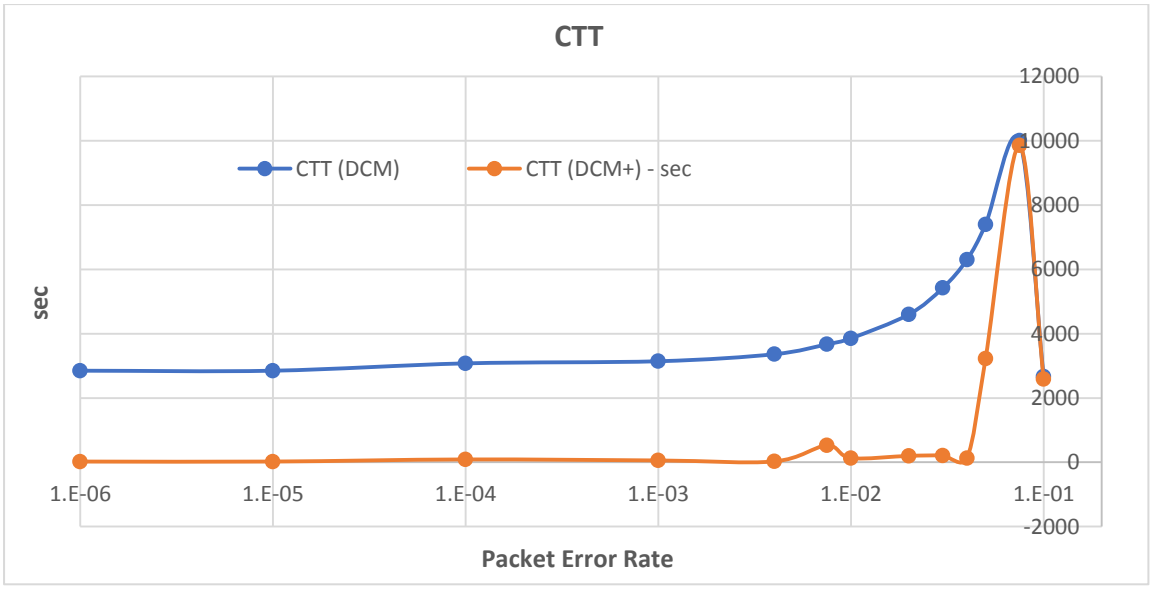


Figure 5-27: CTT Comparison: DCM+ vs DCM

Figure 5-28 compares the robustness of DCM+ and DCM. While DCM shows piecewise linear behavior for all all error rates, DCM+ is nonlinear over the regions with higher error rates in order to minimize the packet losses.

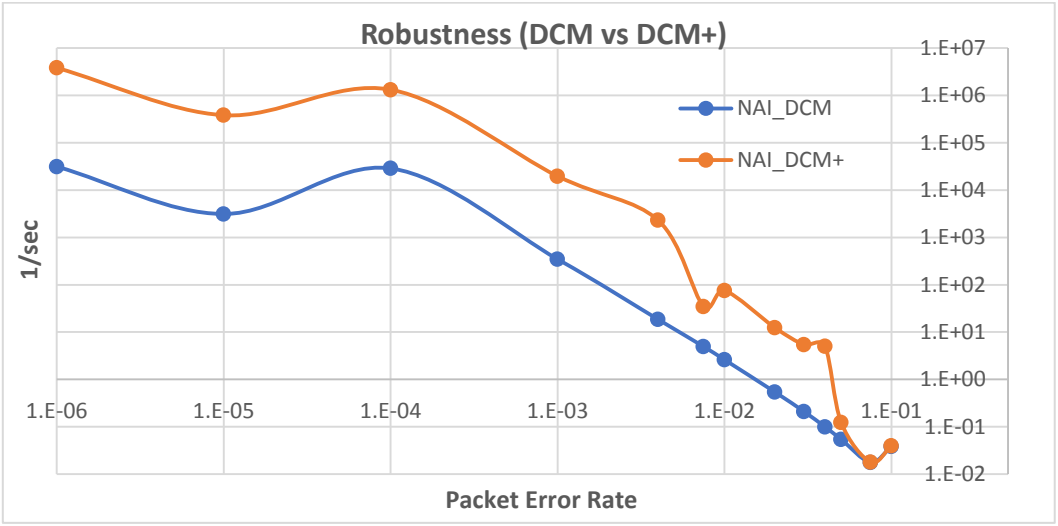


Figure 5-28: Robustness Comparison: DCM+ vs DCM

Figure 5-29 shows the utilization of DCM+ and DCM. DCM+ has the ability to dynamically select the most appropriate bandwidth in order to achieve the best robustness. Beyond the error rate 0.05, we don't get any gains compared with DCM.

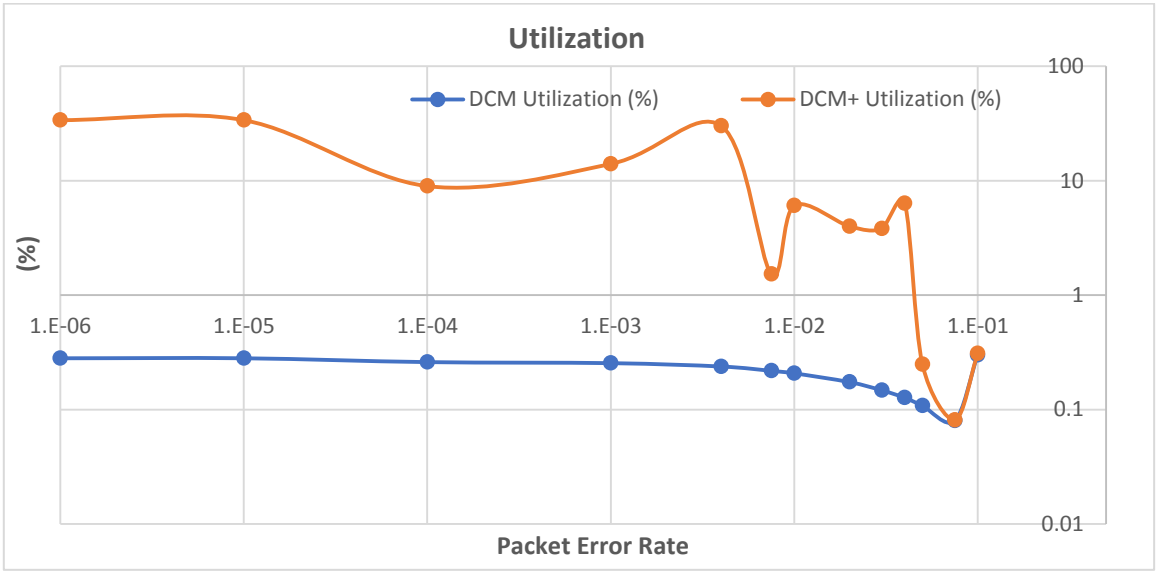


Figure 5-29: Utilization Comparison: DCM+ vs DCM

5.10 Summary

As a result, our simulations of many cases with different values of parameter (error rate, data size, MTU size, protocol, bottleneck bandwidth and access bandwidth) show, that next ***cwnd*** does not exceed ***ssth***, and hence, only very little congestion events could occur. We also found, that ***cwnd*** is changing dynamically and quickly as a reaction on the continuously changing channel capacity. This has been reflected as a higher throughput, ***NAI*** and lower ***CTT***.

In this research work, we have shown, that our approach is stable and robust. It has the ability to minimize the average delay and packet losses, but also to improve the throughput (over 1200% higher than NewReno, over 350% than BIC and 400% than Hybla) at error rate = 0.004.

Chapter Six: Conclusion and Future Work

Content

6.1 Conclusion

In this thesis, we have proposed a new TCP algorithm for controlling the congestion events. It is an L4-only protocol that is designed to be suitable for the different data networks, i.e. wired/ wireless and MANETs. It is a sender-side technique that estimates the available channel capacity (link bandwidth) after each congestion event and before the transmission. It uses the same algorithm as TCP Westwood+ to estimate the bandwidth. Through the simulations, we have shown, that this approach is best appropriate for tough wireless environments with packet error rates between 1e-4 and 5e-2, which are the default values for bad links.

The main idea of DCM+ is based on the ratio between the previous and the current ***RTT*** measurements, which we expressed as the parameter ***rateCA***. In ns3.x simulator, the default algorithm to estimate ***RTT*** value after each ***ACK*** is the ***Karn*** algorithm. The parameter ***rateCA*** helps the TCP sender to detect the status of the channel whether congested or not. According to this value, the transmission may be very fast or very slow. On the other hand, if we detect that ***rateCA*** is increasing, then we can also minimize the ***RTO*** timer, which can additionally speed-up the transmission process. During the transmission, the ***cwnd*** always tracks and never exceeds the value of ***ssthresh***. This has the effect, that less congestion events could occur, and hence, it results in less packet losses and better robustness. Through the robustness measure that we introduced in this thesis, we found that the behavior of DCM+ leads to higher throughputs, improved fairness, less losses, shorter end-to-end delays and transmission times. The analysis of the results shows 2 important properties. The first one is the transmission ***burstiness***, which can be seen in the linear relationship between Tx and Rx. The second property is the inverse relation

between the ratios of ***CTT*** and robustness, which simplifies the judgement of the behavior of DCM+ against other techniques.

The comparison of DCM+ against 12 other TCP protocols has been executed. The advantage of DCM+ is clear in all simulations and their results. Hence, DCM+ shows best performance among all tested approaches. It finishes transmission much faster; it has the highest throughput and it, theoretically, causes no new congestions on the transmission link. This is of great benefit for new devices, that may be currently using the same channel.

6.2 Future Work

We may need more research to enhance the stability and robustness beyond the barrier of (4%) packet error rates. The analysis of more complex topologies by the existence of reverse TCP traffic and different TCP protocols could be very helpful to further understand the fairness and friendliness of DCM+. Further step could be the implementation of DCM+ as an independent module under the network simulators (ns2 and ns3). Practical implementation under *Linux* kernel may be also very helpful to test the performance in real-world networks. We can here summarize the future research topics as follow:

- Generalization of the DCM+ approach through a mathematical model,
- Enhancing the performance of DCM+ through improving the RTT estimation technique,
- Checking the appropriateness of DCM+ for mobile networks like LTE (4G) and (5G).
- Enabling the integration with machine learning approaches like reinforcement learning (RL) to minimize the unwanted drops, and to understand the wireless channel characteristics.

References

[1] S. Floyd, M. Allman, A. Jain, and P. Sarolahti, RFC 4782: Quick-Start for TCP and IP. Internet Engineering Task Force (IETF), 2007.

[2a] Classification Network, http://www.geocities.ws/azlee98/index.html .

[2b] Internet Service Providers, 2015.
Weblink: http://internetserviceprovider.blogspot.com/

[3] GSM Association, The Mobile Economy, North America 2017.

[4] D. Clark et al., Making the world (of communications) a different place, ACM SIGCOMM Computer Communication Review, July 2005.

[5] H. Elaarag, Improving TCP Performance over Mobile Networks, Stetson University.

[6a] W. K. Leong, Z. Wang, B. Leong, TCP Congestion Control Beyond Bandwidth-Delay Product for Mobile Cellular Networks, CoNEXT '17: Proceedings of the 13th International Conference on emerging Networking EXperiments and Technologies, pp. 167–179, Nov. 2017.

[6b] R. Wang, K. Yamada, M. Y. Sanadidi, M. Gerla, TCP With Sender-Side Intelligence to Handle Dynamic, Large, Leaky Pipes, IEEE JOURNAL ON SELECTED AREAS IN COMMUNICATIONS, VOL. 23, NO. 2, FEBRUARY 2005.

[7] S. Floyd and R. Mahajan, Controlling High-Bandwidth Flows at the Congested Router. ICSI Tech Report TR-01-001, April 2001.

[8] R. Hamamreh, D. Khader, DCM+: a multi-purpose protocol for congestion control, 2019 IEEE 7th Palestinian International Conference on Electrical and Computer Engineering (PICECE), 2019.

[9] R. Hamamreh and D. Khader, Adaptive Control of Congestion in Tough Wireless Environments Using DCM+ Algorithm, Int. J. Communications, Network and System Sciences, 2019, 12, 113-123.

[10a] R. Hamamreh and D. Khader, DCM+: Robust Congestion Control Protocol for Mobile Networks, IARIA Conference, ICSNC 2019, Valencia, Spain.

[10b] P. Goyal, M. Alizadeh, H. Balakrishnan, Rethinking Congestion Control for Cellular Networks, HotNets-XVI, November 30-December 1, Palo Alto, CA, USA , 2017.

[11a] S. Floyd and K. Fall, Promoting the Use of End-to-End Congestion Control in the Internet. IEEE/ACM Transactions on Networking, August 1999.
[11b] S. Mangiante, M. Schapira and A. Navon, Congestion Control for Future Mobile Networks, Conf. Paper: CHANTS'18, India, 2018.
[12] S. Floyd, M. Allman, A. Jain, and P. Sarolahti. RFC 4782: Quick-Start for TCP and IP, Internet Engineering Task Force (IETF).
[13] T. Kelly, S. Floyd, and S. Shenker, Patterns of Congestion Collapse. 2003.
[14] R. Gummadi, S. Shenker, S. Floyd, Adaptive RED: An Algorithm for Increasing the Robustness of RED's Active Queue Management, 2001.
[15] R. Pan, Active Queue Management, Cisco Systems, 2006.
[16] T. B. Reddy and A. Ahammed, Performance Comparison of Active Queue Management Techniques, Journal of Computer Science 4 (12): 1020-1023, 2008 ISSN 1549-3636, Science Publications.
[17] Y. Tian, K. Xu and N. Ansari, TCP in Wireless Environments: Problems and Solutions. IEEE Communications Magazine , S27-S32, 2005.
[18] K. Miller and L. Hsiao, TCPTuner: Congestion Control Your Way, 2016.
[19] C. Zhang and V. Tsaoussidis, TCP-real: improving real-time capabilities of TCP over heterogeneous networks, NOSSDAV '01 Proceedings of the 11th international workshop on Network and operating systems support for digital audio and video, pp. 189-198, ACM New York, 2001.
[20] T. Henderson and S. Floyd, RFC 6582: The NewReno Modif. to TCP's Fast Recovery Algorithm, Internet Engineering Task Force (IETF), 2012.
[21] M. Allman, V. Paxson and E. Blanton, TCP Congestion Control. RFC 5681. https://doi.org/10.17487/rfc5681, 2009.
[22] S. Floyd, T. Henderson and A. Gurtov, "The NewReno Modification to TCP's Fast Recovery Algorithm", RFC 3782, April 2004.
[23] J. Olsen, On Packet Loss Rates for TCP Network Modeling, 2004.
[24] Y. Tian, K. XU and N. ANSARI, TCP in Wireless Environments: Problems and Solutions, IEEE Radio Communications, March 2005.
[25] Microsoft Networking Blog. Category-Ledbat. 2018. https://blogs.technet.microsoft.com/networking/category/windows-transports/ledbat/
[26] S. Arianfar, TCP's Congestion Control Implem. in Linux Kernel, 2012.
[27] P. Sarolahti and A. Kuznetsov, Congestion Control in Linux TCP, 2002.

[28] Y. R. Yang and S. L. Simon, General AIMD Congestion Control, National Science Foundation, Technical Report, May 9, 2000.
[29] R. N. Shorten, D. J. Leith, J. Foy and R. Kilduff, Analysis and design of congestion control in synchronized communication networks, Hamilton Institute, NUI Maynooth, June, 2003.
[30] G. Boggia, P. Camarda, A. D'Alconzo, L. A. Grieco, and S. Mascolo, Modeling the AIADD Paradigm in Networks with Variable Delays, 2006.
[31a] R. Ferorelli, L. A. Grieco, S. Mascolo, G. Piscitelli and P. Camarda, Live Internet Measurements Using Westwood+ TCP Congestion Control, MIUR Research Project 488/92, 2004.
[31b] R. Al-Saadi, G. Armitage, J. But and P. Branch, A Survey of Delay-Based and Hybrid TCP Congestion Control algorithms, IEEE Communications Surveys & Tutorials, 2019.
[32] L. A. Grieco and S. Mascolo, "Performance evaluation and comparison of Westwood+, New Reno, and Vegas TCP congestion control", *ACM Comp. Comm. Rev.*, vol. 34, pp. 25 – 38, April 2004.
[33a] S. Park, et. al., ExLL: an extremely low-latency congestion control for mobile cellular networks, CoNEXT '18: Proceedings of the 14th International Conference on emerging Networking EXperiments and Technologies, pp. 307–319, December 2018.
[33b] C. Casetti, M. Gerla, S. Mascolo, M.Y. Sanadidi and R. Wang, TCP Westwood: End-to-End Congestion Control for Wired/Wireless Networks, Kluwer Academic Publishers, 2002.
[34] J. Hoe, Start-up Dynamics of TCP's Congestion Control and Avoidance Schemes, MSc. Thesis, Massachusetts Institute of Technology, June 1995.
[35a] L. A. Grieco and S. Mascolo, End-to-End Bandwidth Estimation for Congestion Control in Packet Networks, 2002.
[35b] Y. Zaki, et. al., Adaptive Congestion Control for Unpredictable Cellular Networks, SIGCOMM '15, August 17 - 21, pp. 509-522, London, 2015.
[36] S. Mascolo, C. Casetti, M. Gerla, M. Y. Sanadidi and R. Wang, TCP Westwood: Bandwidth Estimation for Enhanced Transport over Wireless Links, *ACM Mobicom 2001*, Rome, Italy, July 2001.
[37] A. Dell'Aera, L. A. Grieco, S. Mascolo, Linux 2.4 Implementation of Westwood+ TCP with rate-halving: A Performance Evaluation over the Internet, Tech. Rep. N. 08/03/S, 2004.

[38] C. Caini and R. Firrincieli, TCP Hybla: a TCP enhancement for heterogeneous networks, INTERNATIONAL JOURNAL OF SATELLITE COMMUNICATIONS AND NETWORKING Int. J. Satell. Commun. Network. 2004; 22:547–566 (DOI: 10.1002/sat.799), 2004.
[39] L. Xu, K. Harfoush, I. Rhee, Binary increase congestion control (BIC) for fast long-distance networks, IEEE INFOCOM 2004.
[40] W. Hua, G. Jian, Analysis of TCP BIC Congestion Control Implement., 2009.
[41] S. Shalunov and J. Iyengar, RFC6817: Low Extra Delay Background Transport (LEDBAT), Internet Engineering Task Force (IETF), 2012.
[42] R. Hamamreh, M. Bawatna, DCM: Protocol for Dynamic Avoiding End-to-End Congestion in MANETs, Journal of Wireless Networking and Communications 2014, 4(3) :67-75.
[43] M. Mathis, J. Mahdavi, S. Floyd and A. Romanow, TCP Selective Acknowledgement Options. RFC 2018, April 1996.
[44] Network Simulator (ns3), https://www.nsnam.org/.
[45] Oracle VirtualBox, https://www.virtualbox.org/.
[46] R. Srikant, The Mathe of Internet Congestion Control, Birkhäuser, 2004.
[47] TCP Westwood+ Congestion Control website: https://c3lab.poliba.it/index.php/Westwood
[48 a] R. S. Chang, Computer Networks, https://slideplayer.com/slide/2749292/
.
[48 b] G. A. Abed, M. Ismail and K. Jumari, A Survey on Performance of Congestion Control Mechanisms for Standard TCP Versions, Australian Journal of Basic and Applied Sciences 5(12):1345-1352, Dec. 2011.

Appendix A: Published Papers

Paper1: IEEE Conference in Gaza (April 2019)

Paper2: SCIRP / IJCNS

Paper3: IARIA (Conference in Valencia – Spain – Nov. 2019)

Appendix B: DCM+ Source Code

```
void TcpWestwood::CongestionAvoidance (Ptr<TcpSocketState> tcb,
uint32_t segmentsAcked)
{
   NS_LOG_FUNCTION (this << tcb << segmentsAcked );

      double rateCA ;// Rate of congestion

      rateCA = static_cast<double>(m_oldRtt.GetSeconds()) /
static_cast<double>(tcb->m_minRtt.GetSeconds()) ;
      this->m_minRto= static_cast<Time>( static_cast<double>
(m_minRto.GetSeconds()) /rateCA ) ;


  if   ( segmentsAcked > 0 )

    {

    if (tcb->m_cWnd <= static_cast<uint32_t> (m_currentBW *
static_cast<double> (tcb->m_minRtt.GetSeconds ())))

       {
             tcb->m_cWnd += static_cast<uint32_t> ( 2*rateCA );
     }      else

     {

         tcb->m_cWnd += static_cast<uint32_t>  ( 2 / (
static_cast<double> ( tcb->m_cWnd) * rateCA)) ;
     }

          NS_LOG_INFO ("In CongAvoid, updated to cwnd =  " << tcb-
>m_cWnd <<", and ssthresh = " << tcb->m_ssThresh ) ;
     }
   }
```

Appendix C: Tables of Simulation Results in Chapter 5

Table C-1: Measurements of TCP DCM+ Throughput (Kbps)

error rate	DCM+	NR	BIC	Ledbat	Hybla
1.00E-06	85424.7	85078.2	85424.7	85078.2	90521.5
1.00E-05	85424.7	85078.2	85424.7	85078.2	90521.5
1.00E-04	39985.9	26770.5	75686.3	11031.7	40235.8
1.00E-03	16191.3	4165.15	18787.4	274.136	10460.1
4.00E-03	20713.5	1660.08	5653.28	254.581	4698.02
0.0075	8149.35	1073.2	3276.99	240.844	3093.64
0.01	8165	874.047	2500.27	233.41	2654.59
0.02	3633.28	546.539	1138.1	210.725	1707.25
0.03	5500.75	388.013	664.214	189.079	1246.67
0.04	4836.52	293.019	460.197	168.445	954.074
0.05	587.097	237.567	360.947	152.864	784.448

Table C-2: Measurements of TCP DCM+ Packet Delivery Ration (PDR)

error rate	DCM+	NR	BIC	Ledbat	Hybla
1.00E-06	99.99857	99.99857	99.99857	99.99857	99.99856588
1.00E-05	99.99857	99.99857	99.99857	99.99857	99.99856588
1.00E-04	99.98995	99.99288	97.56887	99.99145	99.99002551
1.00E-03	99.91099	99.92806	99.92089	99.9137	99.91512257
4.00E-03	99.60482	99.63137	99.5987	99.60611	99.6300651
7.50E-03	99.27015	99.28837	99.2902	99.27415	99.27607625
1.00E-02	99.0181	99.00574	99.00939	99.01226	99.0278272
2.00E-02	98.01206	98.10951	98.07573	98.01145	98.07354285
3.00E-02	96.99295	97.02601	97.06824	97.0198	97.02532085
4.00E-02	95.9936	95.88556	95.96028	95.96395	95.94160065
5.00E-02	95.00237	94.95806	94.97931	94.96039	95.01630119

Table C-3: Measurements of TCP DCM+ Packet Losses (%)

error rate	DCM+	NR	BIC	Ledbat	Hybla
1.00E-06	0.0014283	0.0014341	0.001428	0.001434	0.001434
1.00E-05	0.0014283	0.0014341	0.001428	0.001434	0.001434
1.00E-04	0.0100479	0.0071248	2.43113	0.00855	0.009974
1.00E-03	0.089005	0.0719435	0.079114	0.0863	0.084877
4.00E-03	0.395184	0.368628	0.401296	0.393893	0.369935
7.50E-03	0.729845	0.7711632	0.70979	0.725847	0.723924
1.00E-02	0.981903	0.99426	0.990606	0.987735	0.972173
2.00E-02	1.98794	1.89049	1.92427	1.98855	1.92646
3.00E-02	3.00705	2.97399	2.93176	2.9802	2.97468
4.00E-02	4.0064	4.11444	4.03972	4.03605	4.0584
5.00E-02	4.99763	5.04194	5.02069	5.03961	4.9837

Table C-4: Measurements of TCP DCM+ Normalized Advancing Index (NAI)

error rate	DCM+	NR	BIC	Ledbat	Hybla
1.00E-06	7.29E+09	7.26E+09	7289574400	7260006400	7724501333
1.00E-05	7.29E+08	7.26E+08	728957440	726000640	772450133.3
1.00E-04	4.87E+06	4.57E+06	35081.82654	1568952.889	4904935.619
1.00E-03	2.23E+04	7.11E+03	29148.93576	389.8823111	15128.7322
4.00E-03	1.58E+03	1.38E+02	430.7260952	19.74931394	388.46677
0.0075	1.78E+02	2.45E+01	75.01984708	5.383633617	69.42553934
0.01	9.94E+01	1.07E+01	30.65465134	2.8658494	33.16618058
0.02	1.07E+01	1.74E+00	3.552226286	0.634504822	5.316982968
0.03	7.07E+00	5.16E-01	0.896687149	0.249918546	1.653956882
0.04	3.40E+00	2.08E-01	0.333816253	0.12156608	0.686229894
0.05	2.64E-01	1.09E-01	0.166490868	0.069737475	0.363406963

Table C-5: Measurements of TCP DCM+ Average Delay

error rate	DCM+	NR	BIC	Ledbat	Hybla
1.00E-06	124.785	124.961	124.909	124.961	126.755
1.00E-05	124.785	124.961	124.909	124.961	126.755
1.00E-04	48.1966	48.5637	66.2789	48.5497	50.6474
1.00E-03	45.2303	45.2296	45.3605	45.2083	45.3775
4.00E-03	45.4196	45.2311	45.277	45.2077	45.3814
0.0075	45.6092	45.2247	45.2418	45.201	45.3329
0.01	45.3512	45.2241	45.2374	45.1997	45.3268
0.02	45.4831	45.2235	45.2425	45.1983	45.3161
0.03	45.3572	45.2207	45.2412	45.1974	45.3153
0.04	45.849	45.2164	45.2369	45.1964	45.3379
0.05	45.762	45.2147	45.2341	45.1958	45.3803

Table C-6: Measurements of TCP DCM+ Throughput

error rate	DCM+	NR	BIC	Ledbat	Hybla
1.00E-06	9.82705	9.82705	9.82705	9.82705	9.23883
1.00E-05	9.82705	9.82705	9.82705	9.82705	9.23883
1.00E-04	20.8397	31.3329	11.9867	75.9722	20.8627
1.00E-03	51.3938	199.197	44.2103	3027.05	79.3608
4.00E-03	40.7255	501.293	147.354	3273.2	177.226
0.0075	104.794	778.227	255.071	3475.2	270.243
0.01	104.404	958.691	335.442	3598.06	315.921
0.02	239.654	1549.1	745.053	4036.08	497.211
0.03	159.799	2211.41	1291.46	4558.22	690.081
0.04	186.741	2970.03	1888.2	5189.59	914.311
0.05	1544.51	3713.82	2436.92	5795.91	1124.77

Table C-7: Impact of TCP Buffer Size on DCM+ Performance Metrics

buffer size (KB)	Throughput (Kbps)	Packet Loss	NAI
16 k	1415.75	1755	13.76759734
32 k	2899.88	1880	26.32515177
64 k	5511.94	2092	44.9667508
128 k	8627.13	2216	66.44239711
256 k	13214.6	2322	97.1271203
512 k	16571.7	2327	121.540043
1 M	11685.2	1548	128.8290784
2 M	10931.5	1529	122.017179
4 M	7960.86	690	196.9062957
8M	13133.7	1521	147.3691519
16M	13133.7	1521	147.3691519

Table C-8: Comparing DCM+ against Vegas

Case Id	Throughput (Kbps)	delay (msec)	lost packets	CTT (sec)	NAI
case1_DCM+	7913.73	45.3964	689	107.9	9.5715166909E+01
case1_Vegas	3330.34	45.224	691	251.9	4.0163289918E+01
case2_DCM+	11192.2	45.2965	1100	77.6	8.4789393939E+01
case2_Vegas	2113.49	45.1575	1073	404.3	1.6414181423E+01
case3_DCM+	7920.9	45.4733	695	108	9.4974820144E+01
case3_Vegas	3330.34	45.2247	691	251.9	4.0163289918E+01
case4_DCM+	8070.65	45.3906	527	105.07	1.7015918195E+02
case4_Vegas	3773.82	45.2257	525	221.7	7.9869206349E+01
case5_DCM+	85423.1	124.776	1	9.83	7.1185916667E+08
case5_Vegas	2384.72	45.2011	1	347.6	1.9872666667E+07
case6_DCM+	5930.19	45.6565	2233	150.3	7.3769592476E+00
case6_Vegas	956.616	45.2183	2170	898.4	1.2245468510E+00

Table C-9: Impact of Access Bandwidth on the Performance of DCM+

Bottleneck BW	Tx bitrate (Kbps)	mean delay (msec)	Packet Loss (%)
1.E+06	171.683	62.585	1.985
1.E+07	3424.03	53.922	2.08
1.E+08	4322.54	45.5361	2.033
1.E+09	4488.65	45.1647	2.043
1.E+10	4058.88	45.1463	2.019
1.E+11	4186.66	45.1423	2.014

Table C-10: Impact of Access Bandwidth on the Performance of DCM+

Access BW	Tx bitrate (Kbps)	mean delay (msec)	Packet Loss (%)
1.E+06	171.655	62.69	1.9988
1.E+07	1261.9	50.974	2.0356
1.E+08	7511.29	45.47	2.0285
1.E+09	4322.54	45.54	2.033
1.E+10	7142.04	45.55	2.034
1.E+11	7009.71	45.41	2.037