AN EFFICIENT SLOW-START AND CONGESTION AVOIDANCE MECHANISMS FOR TRANSMISSION CONTROL PROTOCOL OVER LONG TERM EVOLUTION ADVANCED NETWORK

GHASSAN AKRAM ABED

THESIS SUBMITTED IN FULFILMENT FOR THE DEGREE OF DOCTOR OF PHILOSOPHY

FACULTY OF ENGINEERING AND BUILT ENVIRONMENT
UNIVERSITI KEBANGSAAN MALAYSIA
BANGI

2013

FASA MULA-PERLAHAN YANG CEKAP DAN MEKANISME MENGELAK KESESAKAN BAGI PROTOKOL KAWALAN PENGHANTARAN MELALUI RANGKAIAN EVOLUSI JANGKA PANJANG TERMAJU

GHASSAN AKRAM ABED

TESIS YANG DIKEMUKAKAN UNTUK MEMPEROLEH IJAZAH
DOKTOR FALSAFAH

FAKULTI KEJURUTERAAN DAN ALAM BINA
UNIVERSITI KEBANGSAAN MALAYSIA
BANGI

2013

DECLARATION

I hereby declare that the work in this thesis is my own except for quotations and summaries, which have been duly acknowledged.

15 January 2013

GHASSAN AKRAM ABED
P53024

ACKNOWLEDGMENTS

I am grateful to ALLAH s.w.t, the omniscient, for guiding me to conceptualize, develop and complete my PhD thesis. Indeed, without His help and will, nothing is accomplished. I am indebted to my supervisor, Professor Dr. Mahamod Bin Ismail for his thoughtful comments, guidance, attention, encouragement during the periods of research, simulation, and writing thesis. Thank you for the invaluable advice and assistance. My thanks go to my co-supervisor, Professor Dr. Kasmiran Bin Jumari, who has initiated ideas, encouraged and given support.

Firstly, To Prophet of Islam MUHAMMAD (Peace and blessings of God be upon him) I dedicate this thesis.

I wish to extend my warmest thanks to all those who have helped me with my work in the Department of Electrical, Electronic and Systems Engineering, Faculty of Engineering and Built Environment, Universiti Kebangsaan Malaysia (UKM). I also thank all my colleagues and my friends especially in the computer laboratory, who helped me during the practical sessions.

This research work was partially supported by the university research grant UKM-OUP-ICT-36-185/2011 and OUP-2012-182.

To my wife, Zainab, my beloved daughters, Ayah and Dania, and my son AbdulRahman, goes my deepest gratitude for their love, understanding and patience and always lifting my spirits through this period. Without their existence besides me, this thesis would have never been completed.

May Allah shows mercy to my father, Akram, who left this world before he could see this moment that he had always hoped to. To my mother, my brothers, and sisters, I would like to pass and dedicate this success.

Finally, I wish to extend my grateful appreciation to all those who have contributed directly and indirectly to the preparation of this study. Not forgetting also, my examiners and all committee members for their helpful suggestions.

ABSTRACT

Transmission Control Protocol (TCP) is the dominant protocol in modern communication networks, within which the issues of reliability, data flow and congestion control must be handled efficiently. TCP provides a communication service at an intermediate level between an application program and the Internet Protocol (IP). With the evolution of Third Generation (3G) toward 4G wireless network, such as Long Term Evolution-Advanced (LTE-Advanced) standard, the use of TCP becoming more important for reliable end-to-end data delivery. As TCP was initially designed for wired networks, the LTE-Advanced network performance will degrade due to signal impairment and heavily congested links. Consequently, the traditional TCP versions such as Tahoe, Reno, Newreno, Sack, Fack, and Vegas, could not fulfil LTE-Advanced requirements to support huge number of user equipment's that need wider channel bandwidth, low latency, and the enormous data stream transferred over this network. The objective of this thesis is to develop an efficient congestion control technique during slow-start and congestion avoidance phases for new TCP agent over LTE-Advanced network. Instead of using exponential increments as in traditional TCP variants, the proposed slow-start algorithm duplicates the congestion window (*cwnd*) and then approximates *cwnd* size using a quadratic interpolation approach until reaching the slow-start threshold (*ssthresh*). An improved Additive Increase Multiple Decrease (AIMD) algorithm control *cwnd* increment to avoid the congestion by deriving a new relationship between AIMD parameters. Moreover, multiple TCP technique achieves virtual multi-flows estimated dynamically according to network congestion conditions over the same connection. The proposed congestion control mechanism is integrated within a new high speed TCP agent called TCP Rapid using Network Simulator 2 (NS-2) and is verified over a traffic model of an LTE-Advanced network. Performance comparison in term of throughput, packet loss, queue size and goodput, as well and the behaviour of the congestion window proved that the proposed mechanism outperform the standard TCP variants. The throughput of TCP Rapid increased by 3 folds compared to Tahoe and doubled with Newreno while the goodput is 45% to 57% better than Reno and maintaining minimum packet loss and adequate queue size. Finally, it can be concluded that the proposed TCP is substantially efficient for wide-bandwidth low-latency networks such as LTE-Advanced, thus achieving higher throughput and a lower packet loss level.

ABSTRAK

Protokol Kawalan Penghantaran (TCP) ialah protokol dominan dalam era rangkaian komunikasi moden yang melibatkan isu-isu seperti keboleharapan, pengaliran data dan kawalan kesesakan yang perlu ditangani dengan cekap. TCP menyediakan perkhidmatan komunikasi pada tahap sederhana yang menghubungkan antara sesuatu program aplikasi dan Protokol Internet (IP). Seiring dengan evolusi teknologi Generasi Ketiga (3G) yang sedang mengorak langkah ke arah rangkaian tanpa wayar 4G seperti piawaian Rangkaian Evolusi Jangka Panjang Termaju *(LTE-Advanced)*, penggunaan TCP menjadi semakin penting dalam proses penghantaran data hujung-ke-hujung yang boleharap. Memandangkan TCP pada asalnya direkabentuk bagi rangkaian berwayar, prestasi rangkaian *LTE-Advanced* akan merudum. Hal ini disebabkan oleh pelemahan isyarat dan pautan yang terlalu sesak. Natijahnya, versi awal TCP seperti Tahoe, Reno, Newreno, Sack, Fack, dan Vegas tidak mampu memenuhi keperluan *LTE-Advanced* untuk menyokong bilangan peralatan pengguna yang tinggi yang memerlukan lebar jalur saluran yang lebih luas, kependaman yang rendah, dan aliran data yang amat banyak yang dipindahkan melalui rangkaian ini. Objektif tesis ini adalah untuk membangunkan teknik kawalan kesesakan yang cekap ketika fasa mula-perlahan dan mengelak kesesakan bagi agen TCP baharu melalui rangkaian *LTE-Advanced*. Selain menggunakan kenaikan eksponen seperti yang terdapat dalam varian TCP yang terdahulu, algoritma mula-perlahan yang dicadangkan akan menyalin tetingkap kesesakan (*cwnd*) and kemudiannya menganggarkan saiz *cwnd* dengan menggunakan pendekatan interpolasi kuadratik sehingga ia mencapai ambang mula-perlahan (*ssthresh*) tersebut. Algoritma Tambah Meningkat Ganda Mengurang (AIMD) yang dipertingkat mengawal kenaikan *cwnd* untuk mengelak kesesakan dengan cara menerbitkan hubungan baru antara parameter-parameter AIMD. Seterusnya, teknik TCP pelbagai mencapai berbilang-bilang aliran maya yang diramalkan secara dinamik berdasarkan keadaan kesesakan sesuatu rangkaian pada sambungan yang sama. Mekanisme kawalan kesesakan yang diusul diintegrasikan ke dalam agen TCP berkelajuan tinggi yang baharu yang dipanggil sebagai TCP Rapid dengan menggunakan *Network Simulator 2 (NS-2)* dan ditentusahkan pada model trafik rangkaian *LTE-Advanced*. Perbandingan prestasi dari segi truput, kehilangan paket, saiz barisan dan gudput, di samping sifat tetingkap kesesakan yang membuktikan bahawa mekanisme kawalan kesesakan yang dicadangkan menunjukkan prestasi yang lebih baik berbanding varian TCP piawai. Truput TCP Rapid juga meningkat tiga kali ganda berbanding Tahoe dan dua kali ganda berbanding Newreno, sementara gudputnya 45% hingga 57% lebih baik berbanding Reno dan mengekalkan kehilangan paket yang minimum dan saiz giliran yang mencukupi. Akhirnya, dapat disimpulkan bahawa TCP yang dicadang amat cekap bagi rangkaian yang berkependaman rendah dan berlebar jalur yang luas seperti *LTE-Advanced*, lalu menghasilkan truput yang lebih tinggi dan tahap kehilangan paket yang lebih rendah.

CONTENTS

LIST OF SYMBOLS

A	Quadratic interpolation coefficient
a	Control parameter in AIMD
B	Quadratic interpolation coefficient
b	Control parameter in AIMD
BW	Bandwidth
C	Quadratic interpolation coefficient
f	Scaling factor
k	Number of virtual TCP flow
N	Number of queued packets
p	Packet loss ratio
R_i	Burst rate
R_o	Bottleneck rate
S	Packets sent for each RTT
T	Packet sent for each second
t	Time
W	Window size
W_i	Initial window size
τ	Time period

LIST OF ABBREVIATIONS

3GPP	Third Generation Partnership Project
4G	Fourth Generation
ACK	Segment Acknowledgment
AIMD	Additive Increase Multiplicative Decrease
AMPS	Advanced Mobile Phone System
ARQ	Automatic Repeat Request
BDP	Bandwidth-Delay Product
BER	Bit Error Rate
BS	Base Station
BSD	Berkeley Software Distribution
C++	C Plus Plus programming language
CDMA	Code Division Multiple Access
CDMA2000	Code Division Multiple Access 2000
CoMP	Coordinated Multipoint
cwnd	Congestion Window
DNS	Domain Naming System
DRTT	Deviation of Round-Trip Time
DUPACK	Duplicated ACK
E-UTRA	Evolved Universal Terrestrial Radio Access
E-UTRAN	Evolved Universal Terrestrial Radio Access Network
ECN	Explicit Congestion Notification
EPC	Evolved Packet Core
EPS	Evolved Packet System
FDD	Frequency Division Duplex
FEC	Forward Error Correction
FTP	File Transfer Protocol
Gbps	Giga Bit per Second
GSM	Global System for Mobile communications
GPRS	General Packet Radio Service
HARQ	Hybrid Automatic Repeat Request

HSTCP	High-Speed TCP
HTTP	Hypertext Transfer Protocol
IMS	Internet Multimedia Server
IMT	International Mobile Telecommunications
IMT-2000	International Mobile Telecommunications-2000
IP	Internet Protocol
ISO	International Standards Organization
ITU	International Telecommunication Union
LAN	Local Area Network
LTE	Long-Term Evolution
LTE-Advanced	Long-Term Evolution Advanced
MAC	Medium Access Control
Mbps	Mega Bit per Second
MH	Mobile Host
MIMO	Multiple Input Multiple Output
MME/GW	Mobility Management Entity/Gateway
MMS	Maximum Segment Size
ms	Millisecond
NAS	Non-Access Stratum
NFS	Network File System
NS-2	Network Simulation
OFDM	Orthogonal Frequency Division Multiplexing
OSI	Open Systems Interconnection
OSS	Operations Support System
OTcl	Object-Oriented Tool Command Language
QoS	Quality of Service
P-GW	Packet Data Network Gateway
PDCP	Packet Data Convergence Protocol
PDN	Packet Data Network
PPL	Probability of Packet Loss
RED	Random Error Detection
RFC	Request for Comments
RLC	Radio Link Control

RN	Relay Node
RNC	Radio Network Controller
RRC	Radio Resource Control
RTO	Round-Trip Timeout
RTT	Round-Trip Time
rwnd	Advertised Receiver Window
S-GW	Serving Gateway
SAE	System Architecture Evolution
SCTP	Stream Control Transmission Protocol
SMTP	Simple Mail Transfer Protocol
ssthresh	Slow-Start Threshold
Tcl	Tool Command Language
tclcl	Tcl with classes
TCP	Transmission Control Protocol
TDD	Time Division Duplex
TDMA	Time Division Multiple Access
UDP	User Datagram Protocol
UE	User Equipment
UMTS	Universal Mobile Telecommunication System
WiMAX	Worldwide Interoperability for Microwave Access
WLAN	Wireless Local Area Networks
WWAN	Wireless Wide Area Networks
WWW	World Wide Web

LIST OF FIGURES

LIST OF TABLES

CHAPTER I

INTRODUCTION

1.1 INTRODUCTION

The era of future communication systems is upon us, and finally, it will enter into our lives and change the way we live, because many technological innovations and developments have actually come from our needs in life. The development of mobile systems and the prospect of wireless and cellular communications enquire several questions of the operators, developers, manufacturers, designers, and scientists, working in this field. The communications world is open to several alternatives, such as thoughts, ideas, suggestions, and activities of the near future, which may possibly provide the answers to these open points and indicate the next trends of the communications world, especially over wireless links (Mukherjee et al. 2003).

The fourth generation (4G) of mobile systems is now replacing the 3G and 2G families of standards. A new mobile generation has always appeared every 10 years since the first generation system in 1981. The second generation in 1992, and the third generation, which was released in 2001, followed this. The development of 4G systems began in 2002. Figure 1.1 shows the evolution of the radio access technologies from the first generation up to the LTE-Advanced in terms of user speed and data rates (3GPP 2009a). The Advanced Mobile Phone System (AMPS) was the first generation cellular technology that uses separate frequencies, or "channels", for each conversation that was released through the 1980s with limited data rates. Then, it was evaluated to Code Division Multiple Access (CDMA).

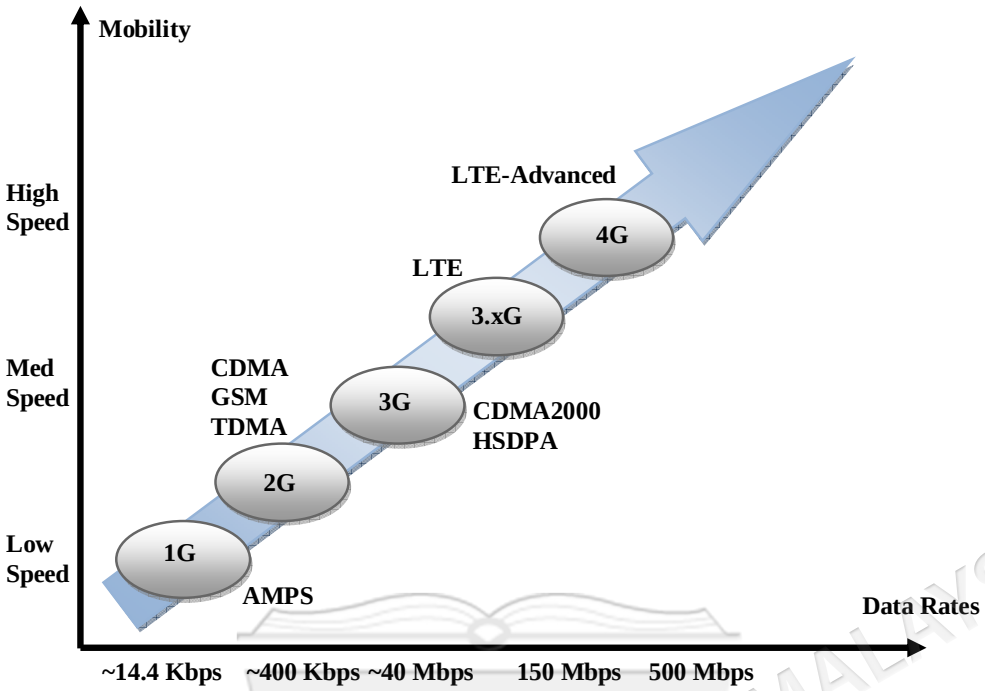


Figure 1.1 Evolution of the radio access technologies

Time Division Multiple Access (TDMA) and CDMA are characterized as a channel access method used by various radio communication technologies and they represent the second generation. In particular, CDMA2000 is a family of the third generation which uses CDMA channel access to send voice, data, and signalling data between mobile phones and cell sites. However, the real new revolution began with Third Generation Partnership Project (3GPP) in December 1998. 3GPP concept was established through the cooperation of wireless communications associations to issue a globally viable third generation system, with standard specifications, inside the range of the International Mobile Telecommunications-2000 project (IMT-2000) of the International Telecommunication Union (ITU). Meanwhile, the 3GPP specifications and standards are based on a Global System for Mobile communications (GSM) specifications. The ITU has devised the IMT-Advanced term to recognize the new mobile systems, which is capable of going beyond the International Mobile Telecommunications 2000 (IMT 2000). In specific, the requirements of the data rate have been amplified.

Third generation systems were developed to support high performance multimedia communication links, which made enhancements in user-to-user telecommunication links and provided a high quality platform for images and video transfers. Furthermore, the access to information and services on public, private, and heterogeneous networks would be enhanced by higher rate of data transfer capability and a new flexible communication ability to achieve the users' needs (Holma & Toskala 2000). 3GPP's Long-Term Evolution (LTE) networks achieved the next step towards high data rate transfers and the support of all multimedia functions and requirements. LTE networks were developed to improve the Universal Mobile Telecommunication System (UMTS) mobile phone standards to face future requirements. LTE network approach was not a standard itself, but it resulted in the newly evolved Release 8 of UMTS standard and contained most (if not all) of the extensions, expansions, and modifications of UMTS standards (Karakaya et al. 2009).

With the deployment of LTE networks, the wireless revolution will achieve an important milestone. For the first time, a wide-area wireless network will be universally designed for Internet Protocol (IP) broadband data (rather than voice). LTE networks are also rapidly becoming a dominant global standard for the fourth generation cellular networks, with nearly all the major cellular players behind it and working towards its success (Ghosh et al. 2010). The standardization of LTE in 3GPP is gotten an established state, and the modifications in the design are narrow. Since the end of 2009, the LTE system has been installed as a normal growth of GSM and UMTS. LTE specified by 3GPP as very high flexible for radio interfacing. LTE deployment started in the last of 2009 and the first LTE release is supported data rate up to 300 Mbps and delay of radio network not as much than 5 ms. In addition, LTE provided a spectrum significant increasing in spectrum efficiency if comparing with any other cellular systems. Furthermore, it takes into account different architecture in radio network that designed to speed-up the operations and to decreasing the cost (Astély et al. 2009). LTE systems are supporting Frequency Division Duplex (FDD) with Time Division Duplex (TDD) technique as a varied array of bandwidths to operating in a wide amount of dissimilar spectrum allocations.

3GPP is also conducted a research with the objective to identify the required enhancements for LTE systems to achieve the requirements of IMT-Advanced. In September 2009, the partners of 3GPP prepared the official suggestion to the proposed new ITU systems, which was represented by LTE with Release 10 and beyond to be the appraised and the candidate towards IMT-Advanced. After attaining the requirements, the main object to bring LTE to IMT-Advanced is that IMT systems must be candidates for the novel spectrum bands is still to be acknowledged (Kiiski 2010). In order to achieve this goal, 3GPP is presently evolving LTE-Advanced as a development of the standard of LTE.

LTE-Advanced is applying various bands of the spectrum, which are already valid in LTE, along with the future of the bands of IMT-Advanced. More developments of the spectral efficiency in the downlink and uplink are embattled specifically if users serve at edge of cell. In addition, LTE-Advanced aims at quicker exchanging between the resource of the radio states and between the additional enhancements of the figures of latency, but all at once, the "bit cost" must be decreased (Stencel et al. 2010). The IMT-Advanced represents the next generation in the systems of wireless communications, which aim to accomplish other main advance of the current third generation systems by reaching up to the uplink (UL) rate of 500 Mbps and 1 Gbps in the downlink (DL) (Nam et al. 2010).

With LTE-Advanced starting, there are many keys of requests and features that up come to the light. There are various high level purposes for the new specifications of LTE-Advanced, which have not been considered in the system specifications. It is needed to verify, while much effort rests to be accepted before fixing it in the specifications. At present, several core significant intentions for the LTE-Advanced can be illustrated as follows (Agilent 2010):

1. The data rate with the peak uplink of 500 Mbps and the peak downlink of 1 Gbps.
2. Provide spectrum efficiency with more than three times that that provided by LTE and offer spectrum efficiency in the uplink 15 bps/Hz and in downlink 30 bps/Hz.
3. The link latency in the case from idle status to the connected status is a smaller than 50 ms and less than 5 ms for one-way in a single packet transferring.

4. The edge throughput of users' cell to be doubles than that in the LTE.
5. The average throughput of any user is to be triple than that in the LTE.
6. The mobility environments are similar to that used in the LTE.

Then, LTE-Advanced or the fourth generation system of mobile communications is the most impressive technology of the next wireless generation networks. Previously, many researchers, developers, and scientists worldwide worked on the projects that are supported by governments and other institutions, aiming at more efficient and faster wireless networks through the integration of all available technologies. They sought to adapt the newly enhanced telecommunication systems, which could provide users with superior quality, efficiency, and opportunities, where the classic wireless communications are no longer feasible. Some researchers define the fourth generation concept as a significant enhancement of the third generation systems. However, for other scientists, 4G combines both cellular and wireless local area networks to introduce the new routing techniques, efficient solutions for sharing dedicated frequency bands. Moreover, they can introduce the development of transmission over transport layer, increasing mobility, or enlarge the available bandwidth of the network path.

One of the main solutions to improve the performance of the new wireless communication systems is by improving Transmission Control Protocol (TCP) performance. The TCP provides a communication service at an intermediate level between an application program and IP. In specific, when an application program send a large chunk of data across the Internet using an IP, instead of breaking the data into IP-sized pieces and issuing a series of IP requests, the software can issue a single request to TCP and let TCP handle the IP details. Meanwhile, data applications and packet transmissions are always built over the top of TCP, due to the capability of the TCP to provide end-to-end reliability in data retransmissions when missing IP packets. Originally, TCP was designed and developed to operate on the wired networks, where the possibility of packet loss could only be due to network congestion. Therefore, the congestion control mechanism used by TCP is adjusted upon the detection of packet loss.

However, in wireless networks, most of packet losses are due to the radio conditions and interference and not because of the network congestion alone. Errors in the wireless links are actually caused by several factors such as interference from the other telecommunication sources, fading due to mobility, and scattering resulting from the large number of signal reflecting (Chuah & Zhang 2006).

Besides, the average throughput in the cellular links is not only dependent on the congestion in the network path, but also on other factors, like bit error rate of the wireless medium, cell handover time, and the cell resident time (Chockalingam et al. 1998). Ordinarily, in contrast, the error characteristics of the wireless channels differ from those of the wired channels, i.e. the TCP gives poor performance when it is directly applied to the wireless networks. Conversely, the wired channels are characterized to deal with miniscule packet loss probabilities and small randomly spaced errors. Meanwhile, the wireless channels characterized by time varying packet loss probabilities are normally much larger than in the wired channels, and the errors, which are typically bursty on the wireless channels (Bao 1996).

1.2 TCP OVER LTE/LTE-ADVANCED NETWORKS

One of the main requirements of 4G systems is that the users should not feel any difference between a wired and a wireless network, and they should have multiple options for connectivity over heterogeneous networks. In other words, it needs a new adaptive, reliable, and efficient TCP that is able to fulfil these requirements. The direct approach to enhance the performance of TCP for the new generation networks is to modify the TCP itself because TCP still represents one of the major causes of poor performance in a wireless environment (Xylomenos & Polyzos 1999).

Meanwhile, the LTE-Advanced networks plan to support mixed voice, video and other messaging traffic data. Unfortunately, wired and wireless networks have different main features, such as bandwidth capacity, propagation delay, and reliability in links. The main reason beyond these differences is that packet losses are not caused by network congestion, as it is in the wired networks, but by many other factors of wireless specific reasons and features.

In the Wireless Local Area Networks (WLANs) or cellular systems, most lost packets are fundamentally due to the large amount of bit error rates in the wireless networks and the possibility of the handovers between two cells and base stations. In the mobile systems, however, most lost packets accrue are due to medium congestion and route breakages (Ghazaleh & Muhanna 2008).

For these reasons, TCP performs well over wired links and networks, but it suffers serious problems and performance degradation over the wireless networks. This will certainly happen when the networks transfer high data rate when the bandwidth reaches 1 Gbps, such as the LTE-Advanced system. The characteristics of the wired channels expect a very small amount of lost packet probabilities and randomly spaced errors, while wireless channels are characterized by time varying packet loss probabilities and are typically very large as compared with those of the wired channels (Bao 1996). It is also true that the standard protocols, such as TCP, perform poorly over the wireless links, which is due to lost or delayed packets. In addition, the TCP can deal with these problems and resolve some types of congestion in the network paths, but this may cause reducing in the packet flows. In some cases, such as when the TCP suffers unpredictable losses or packet delay (occurring in wireless channels), the TCP reduces packet rate even when the wireless link bandwidth is still with full capacity (Wisely 2009).

Many researchers have found that TCP supports wireless access inefficiently (Zorzi & Rao 1997) and many of these researchers found that the key problem is that wireless channel errors would lead to frequent failures of the TCP retransmission time. This will cause an extra congestion by the TCP added to the expected congestion of the network path (Hoque et al. 2007). Mostly, the networks are built on top of the TCP for data applications because it provides a reliable end-to-end retransmission scenario when the IP packets are dropped (Chuah & Zhang 2006). Considering the fact that TCP was originally designed for wired networks, whose packet losses were due to network congestion, the window size of the TCP would be adjusted upon detection of packet loss.

The LTE networks were designed to be packet-based and the architecture was developed to contain less network nodes and minimum links routing. This would reduce the protocol processing overhead and lead to a reduction of latency in the network queues. In addition, the evolved architecture of the LTE networks focuses on the use of the TCP/IP protocol as a packet data communications protocol, whilst others focus on Stream Control Transmission Protocol (SCTP).

The basic architecture of the LTE and LTE-advanced systems is dependent on the Evolved Packet Core (EPC) network element. Like all IP systems, in the EPC of LTE, the IP cannot provide a reliable service which degrades the packet transmissions in term of packet loss, packet duplication, or when the packet is delivered out of the sequence (Tan 2009). However, TCP can provide the required delivery service on the top of IP and can give a more reliable data link. TCP on the networks is responsible for controlling packet flow and resolving problems that are resulting from congestion by using congestion control mechanisms, in addition to providing connection management for all network traffic. Nonetheless, LTE and LTE-Advanced networks support a high throughput by an IP based transport network link. Then, the TCP is expected to play a hazardous role in the performance of LTE systems (Tian et al. 2005).

Due to the high error rates and because it is inappropriate to directly use the TCP on the LTE system, the LTE employs an error recovery technique at the link layer to overlap the error recovery mechanism which is provided by the TCP in the transport layer (Kliazovich et al. 2007). There are many end-to-end solutions to improve TCP over LTE/LTE-Advanced systems during the handover period, or to increase the size of the TCP window. The problem lies in the required modification of the current TCP variants (Liao et al. 2005). The TCP actually provides a very good service, where reliability is paramount; however, it is not generally suitable when delays are critical or when the network has a low propagation delay (Bannister et al. 2004). In fact, this is expected with the LTE-Advanced networks, which support many applications, such as voice and video services.

Added to that discrete loss in the data transfer is less important than its time delivery by TCP over the classic mobile systems. Therefore, more research is still required to address these particular transmission characteristics of the new generation systems.

When the TCP is used over the LTE/LTE-Advanced systems, some packets are surely neglected and the TCP sender will be aware of this loss through the acknowledging transmission mechanism. This means that when the TCP sender sends packets and does not receive an ACK after a while, a loss has occurred and the sender must retransmit these packets. In the LTE and LTE-Advanced systems, a high bit rate, which is similar or even larger than that offered in the wired networks, is offered.

This will certainly cause a lot of pressure to the performance of the TCP; which is more than that of the traditional wireless networks (Kliazovich et al. 2007). Standard TCP versions assume that most lost packets (about 99%) in wired channels are due to the congestion in the network traffics and the remaining (about 1% only) are due to damages (Kozierok 2005). Therefore, it is important to take other actions to avoid any possible congestion in the network, where the congestion is controlled by the TCP to prevent the sender from overflowing the network capacity. This mechanism is known as the TCP congestion control. This TCP congestion control is certainly not a new technique to improve the TCP performance, but many algorithms, which are implemented for different TCP variants, have been developed to deal with network congestion (Tan 2009).

Many research and studies have suggested improving the ordinary TCP versions in order to obtain a better performance on 3G and beyond 3G networks, but many of these trials have proposed to solve either the handover or the bit error rate problems, such as I-TCP, M-TCP, Freeze TCP, and Snoop TCP (Ghazaleh & Muhanna 2008). In the incoming 4G mobile systems, a high data rate is expected and the expected services provided by this generation should be carrying because it needs a robust TCP protocol to overcome the new challenges like handover, bit error rate, and asymmetry between the base station and mobile host.

1.3 PROBLEM STATEMENT

The LTE-Advanced systems are becoming the first choice of operators when constructing a new network infrastructure. Even though the LTE-Advanced system can offer a high speed data service that can be benefit by wideband applications, the large bandwidth also results in high control signalling costs. The next-generation wireless networks, such as the LTE and Worldwide Interoperability for Microwave Access (WiMAX), can achieve the throughputs of several Mbps to the mobile users with TCP. In addition, the TCP protocol is on top of most Internet, voice, and video applications today and all these use TCP for World Wide Web (WWW) E-mail, and other traffics (Sharma et al. 2010). Both the LTE and LTE-Advanced systems suppose TCP as an end-to-end protocol, which can control congestion states and deal with packet loss. In addition, the LTE/LTE-Advanced systems support the quality of service mechanisms over radio and transport links but there is no mechanism to control packets flow (Pacifico et al. 2009).

Each TCP includes a congestion control mechanism, which represents a critical design of the TCP. The main function of a congestion control is to adjust the end-to-end data rate over the network connections and to prevent sending beyond links capacity limits. Despite many enhancements implemented to improve the performance of the TCP congestion control, it still operates poorly in high bandwidth networks. This is due to a slow response when encounter large congestion event which reduces the size of the congestion window (*cwnd*) (Chan et al. 2004). The congestion window represents the number of packets that are allowed to be transmitting without being acknowledged.

Mobile users can potentially be affected by the TCP throughput due to mobility and handoff that may cause frequent disconnections. Recently, many congestion control protocols have been proposed, especially for streaming multimedia applications. Nonetheless, the major problem with the application of the TCP over the wireless networks is that it does not have any mechanism or indication to recognize between congestion losses and wireless random losses, which are caused by wireless channels and this, will surely cause a degradation of throughput (Chan et al. 2004).

TCP congestion control includes two phases with two mechanisms, namely, slow-start and congestion avoidance. The slow-start is used in combination with congestion avoidance to avoid sending data more than the network is capable of transmitting, and to prevent network congestion as well.

Slow-start mechanism is based on the use of exponential and linear increments to grow the congestion window gradually until it reaches a specified threshold. During the exponential growth phase, slow-start works by increasing the TCP congestion window each time the acknowledgment of the previous segments is received. This happens until either an acknowledgment is not received for some segments or it reaches the estimated threshold. If a packet loss occurs, the TCP will assume that it is due to network congestion and then reduce the packet flow rate. Once a packet loss occurs or when the threshold is reached, the TCP enters the linear growth (congestion avoidance) phase. At this point, the window increases by one segment for each Round Trip Time (RTT).

The problem in the linear and exponential increments of the slow-start mechanism is that it requires a long period to reach the slow-start threshold (*ssthresh*) of the network connection. This *ssthresh* is estimated by the congestion control to detect the connection limits and to later avoid going beyond it. In fact, the slow mode of the slow-start mechanism still represents a major challenge over the high-speed wired-wireless networks. This is because the handover of the mobile devices forces the TCP congestion control to build the congestion window after each connection establishment. Therefore, the TCP requires extra RTTs (required time to send and acknowledge packets) and this will minimize the number of packets transferred between the TCP senders and TCP receivers. As a result, the linear-exponential strategy in the slow-start mechanism used by the standard TCP variants cannot fulfil the requirements of the high speed networks like the LTE-Advanced and this will surely degrade the throughput of the TCP performance among the network elements.

On the other hand, the congestion avoidance mechanism also provides limited performance when it is applied over large-bandwidth low-latency networks. The reason is that the standard congestion avoidance mechanism cannot sense the available bandwidth of the network path and it still sends a small amount of packets but the bandwidth can accept more. In addition, the increment factors to control the avoidance phase are still imperfect and the mechanism always uses the same values for all types of network. In fact, most of the congestion avoidance mechanisms do not take into account the current conditions and the status of the network in term of congestion and available bandwidth. Moreover, the congestion avoidance mechanism sometimes uses poor approaches to set the congestion window after loss events such as setting the window size to one segment only when the connection can actually accept larger size.

Similarly, the typical congestion avoidance mechanisms do not include adaptive techniques that can assist simultaneous transfers of multiple packets over the same connection instead of sending over a single flow. Although some mechanisms can provide multiple flows, there is no adaptive method for estimating the optimum flow number. Mostly, these mechanisms lead to degrade the network performance due to the extra packet loss resulting from using a number of flows that do not fit the connection capacity.

The enhancement of the TCP can provide a better performance during the occurrence of random losses, dynamic traffic loads, and over large bandwidth product paths. Therefore, when the congestion control mechanism of the TCP is enhanced, the network positively gives higher throughput. In fact, it will not only keep the packet loss rate small, but also ensure that the bottleneck queue is small and reasonable to avoid the long packet delays. Additionally, the use of a single flow protocol over a large bandwidth link has become an incompetent methodology if we look for a high performance. This is because the protocols with a single flow cannot cover the available bandwidth to run at a high speed. Therefore, the choice of the multiple flow protocols becomes compulsory over future networks.

The enhancement of slow-start and congestion avoidance mechanisms because of the need to improve the performance of the congestion control for the existing TCP versions over LTE-Advanced represents the main target due to the fact that these versions are still used over the high-speed low-latency networks, in spite of the TCPs being developed over 20 years ago. The challenges of large-bandwidth and low latency links, high congestion traffics, and the huge data transferred over LTE-Advanced have motivated the developers to explore other innovative techniques to enhance the TCP congestion control for the next generation network. Thus, the evolution of the congestion control still represents a key factor to improve the protocol performance.

1.4 RESEARCH OBJECTIVES AND SCOPE

The primary objective of this study is to develop an efficient congestion control mechanism over a new TCP agent. This mechanism can achieve high performance over large-bandwidth and low-latency networks such as the LTE-Advanced networks. The proposed TCP congestion control must achieve the requirements of the future networks in order to give a high throughput and minimum packet loss. Furthermore, the enhanced congestion control should be able to open the TCP window, depending on the available bandwidth of the network connection with the capability to send data over many flows for the same connection. In addition, the slow-start mechanism in the enhanced congestion control should include a novel and adaptive technique to start-up the congestion window towards *ssthresh* in a shorter period.

In particular, this study has identified six significant goals, which are summarized as follows:

1. To propose an efficient approaches to start-up the congestion window within a shorter period in the slow-start phase.

2. To improve the increment and decrement strategy that is used to adjust the congestion window in the congestion avoidance phase.

3. To develop an adaptive method that can make the TCP congestion control send multiple packets at the same time over the same connection using the multiple flow technique.

4. To generate a new TCP agent using a network simulator that involves enhanced congestion control mechanism.

5. To formulate and simulate a realistic traffic model of the LTE-Advanced network using a network simulator.

6. To validate and evaluate the performance of the proposed TCP over the LTE-Advanced model.

Meanwhile, the main scope of this study is based on developing an innovative congestion control mechanism in both the slow-start and congestion avoidance phases with the multiple packet flows. This mechanism can be integrated with the new TCP agent that can provide a high throughput and a low packet drop. The proposed TCP can be considered as a high-speed protocol, which is appropriate to efficiently run over the LTE-Advanced network and fits the requirements of this particular network. Figure 1.2 demonstrates the scope of this study using a bottom-up diagram.

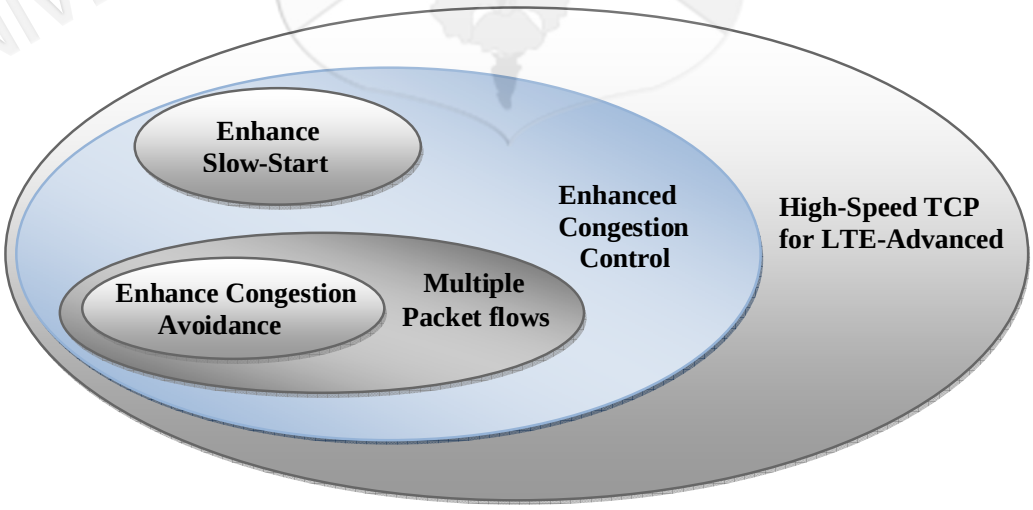


Figure 1.2 A bottom-up diagram of study scope

1.5 RESEARCH METHODOLOGY

In order to achieve the aforementioned objectives, the following methodology utilized in the study is outlined below and briefly described in Figure 1.3.

1. Reviewing the previous literature related to the behaviour of TCP over the wired-wireless networks and cellular systems.

2. Specifying several necessary theoretical expressions before studying and analysing the possible approaches to improve the congestion control in high-speed networks.

3. Developing an efficient technique to improve the slow-start mechanism in the TCP congestion control that offers a faster start-up to the congestion window using a novel increment approach.

4. Enhancing the congestion avoidance mechanism by improving the increment/decrement parameters that are used to adjust the congestion window size.

5. Improving the existing techniques to transmitting packets using extra virtual TCP flows over the same connection and proposing an adaptive method to estimate the optimum flow number.

6. Formulating an independent TCP agent using the Network Simulator 2 (NS-2) platform and integrating it with the enhanced congestion control mechanism.

7. Characterizing, modelling and simulating an accurate traffic model of the LTE-Advanced networks using the NS-2 modeller.

8. Experimenting and evaluating the proposed TCP over the proposed model of the LTE-Advanced network and comparing the performance and the behaviour of the proposed TCP with other standard TCP variants during different network conditions.

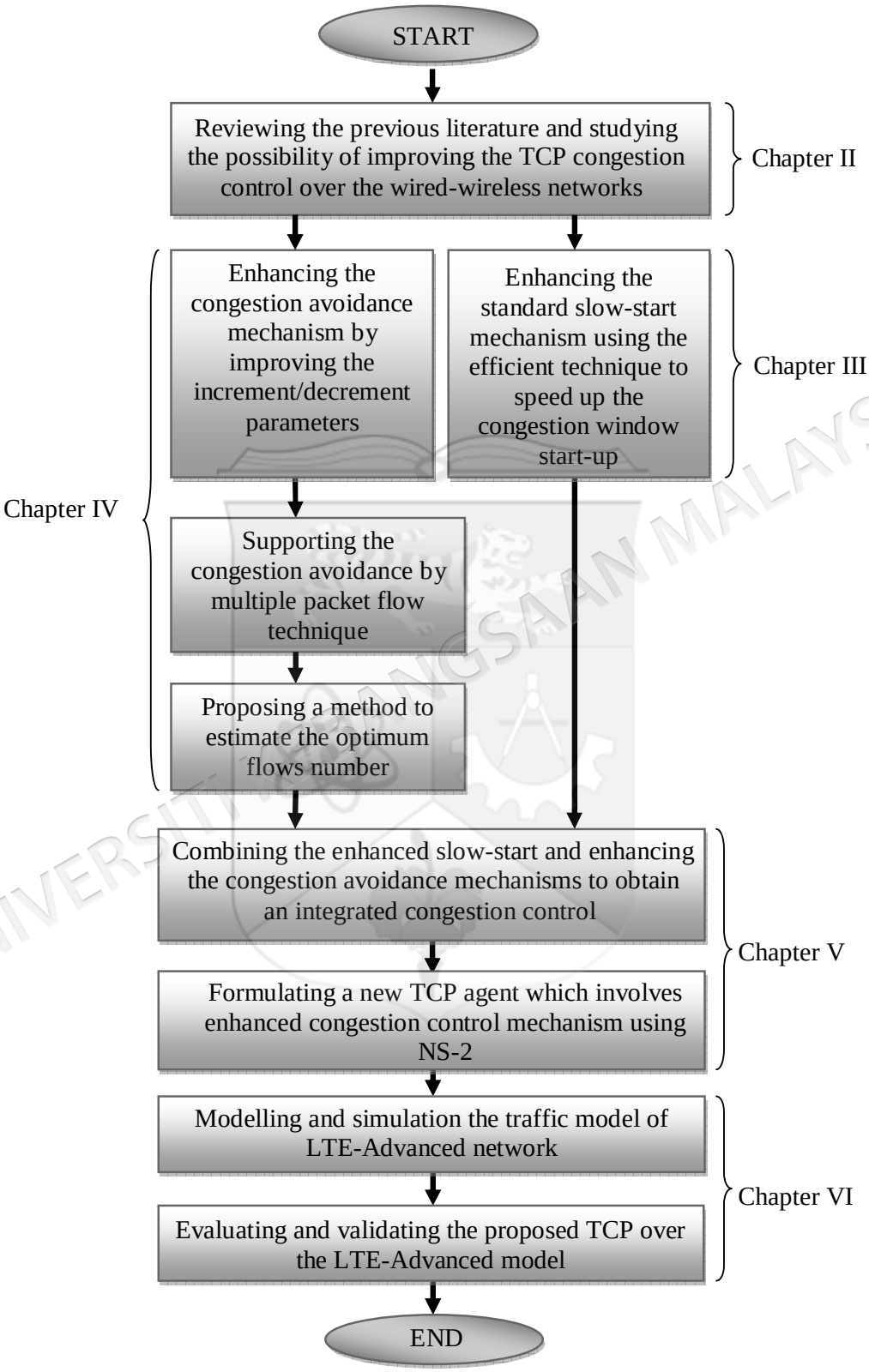


Figure 1.3 A block diagram of the briefed research methodology

1.6 THESIS CONTRIBUTIONS

The primary philosophical theme of this thesis is the concept of a new congestion control mechanism. This work explored the concepts of an enhanced TCP congestion control mechanism to improve the performance of the TCP over large-bandwidth low-latency networks such as the LTE-Advanced. This study also attempted to integrate the new TCP with enhanced congestion control algorithm in order to achieve high performance TCP that could fulfil the requirements and the limitations of this network. The following section summarizes the contributions of the current research work:

1. The first contribution of this thesis is the development of a new slow-start technique to permit the congestion window to grow faster instead of the linear-exponential increments that are used by the standard TCP versions. The new approach allows the congestion control to duplicate the *cwnd* size for each packet acknowledgment received, and then to approximate the *cwnd* to reach *ssthresh* using the linear interpolation method.

2. Another slow-start algorithm developed in this research was based on the same concept of duplicating *cwnd* but it is faster and uses less number of iterations to reach *ssthresh*. This slow-start mechanism is considered as an intelligent incrementing approach by approximating the *cwnd* using quadratic interpolation formula. This technique allows the *cwnd* to work with rapid clocking and delays the congestion point of the network connection.

3. The enhanced congestion avoidance mechanism was proposed to improve the increment and decrement scheme used in the congestion avoidance phase. The new approach included an enhanced relationship between the increment/decrement parameter to offer wide desired values of the control parameters instead of the limited values used in the standard mechanisms.

4. Additionally, the enhanced congestion avoidance mechanism adopted an innovative technique to send multiple packets simultaneously over the same connection. Due to the available bandwidth, extra packets can be injected in the network pipeline. This technique is supported by novel method to dynamically estimate the optimum number of the virtual flows used by the TCP congestion control after each loss event. This method centres upon monitoring the status of *cwnd*, before and after loss occurrences, and is taken as an indication of the number of the virtual flows that are permitted to be used by the TCP at this point of transmission.

5. The integration between the enhanced slow-start and enhanced congestion avoidance algorithms provides a rapid and adaptive congestion control mechanism, where it is merged within the new TCP agent (called TCP Rapid) and this TCP agent was formulated with an independent feature and characteristics over the NS-2 platform.

6. One of the significant contributions of this research work was to implement a realistic model for the LTE-Advanced network traffic using the NS-2 simulator and evaluating the performance of the TCP Rapid over the LTE-Advanced model.

1.7 THESIS ORGANIZATION

This thesis is organized into seven chapters. The first chapter introduces the definition, advantages, weaknesses, and TCP congestion control over LTE-Advanced systems. This chapter also states the objectives and the contributions of this the research work.

Chapter II provides a literature review that gives an overview of the TCP concepts and the recent enhancements over the wired-wireless networks. It also gives a detailed survey of the TCP extensions and modifications over the LTE/LTE-Advanced networks.

Chapter III elaborates on the two slow-start mechanisms with the derivation and formulation of the proposed algorithms in addition to evaluating the performance over the high-congested network.

Meanwhile, an adaptive congestion avoidance algorithm is illustrated in Chapter IV. The enhanced congestion avoidance is constructed from three stages, enhancing the relationship between the control parameters pair, adding the multiple packets flow technique, and estimating the number of the optimum flows dynamically. Comparisons of the standard and improved mechanisms are also included in this chapter.

The formulation and construction of the TCP agent and the required modifications in the NS-2 source files are demonstrated and discussed in Chapter V. In addition, a combination of the proposed congestion control mechanism with the new TCP is also described systematic in this chapter.

In Chapter VI, the LTE-Advanced model is established with all the required elements, traffics, and routings. Implementation, evaluation, and validation of the proposed TCP over the LTE-Advanced model using different metrics are also discussed in this chapter. Furthermore, various evaluations for the proposed TCP under a wide range of network parameters are also included in this section, and the results obtained are compared with other TCP variants.

Finally, Chapter VII concludes this thesis and presents open problems, and limitations of study, selects directions for further research, offers recommendations, and gives significant ideas for follow-up work.

CHAPTER II

LITERATURE REVIEW

2.1 INTRODUCTION

Ever since the first generation cellular systems have invaded the market, the problem of data services emerged and has exposed the users to many challenges. On other hand, the development of radio access platform in the new cellular systems has minimized those challenges and has paved way for the next generation of cellular systems. Apart from the 3G the 4G has continued to receive much attention for addressing the needs for high-speed data of the wireless mobility markets. With the evolution of 3G technologies such as UMTS, the TCP has become more important and popular in delivering data to the end-to-end applications. This popularity has forced the developers and researchers to propose many techniques to improve the performance of TCP in wireless networks. Consequently, many approaches have been proposed, such as retransmission over link layer, which notifies the TCP sender about the existing network conditions, or TCP versions with extra mechanisms.

In fact, mobile networks developers are seriously involved in improving the performance of TCP of all the mobile generations. Furthermore, the developers are motivated to enhance the TCP to get better design for 3G radio access and beyond 3G networks to get better operation from inadequate radio resources. Moreover, the developers are working on to optimize the TCP to increase the rates of data required by networks users. It is worth mentioning, that the TCP optimizations can enhance the performance of the network resources and provide optimum services to the users with low level of maintenance (Modi 2008).

2.2 LAYERING CONCEPT AND TRANSPORT LAYER

The communication functions and applications are divided into layers, and each device or terminal in these layers has been with an assigned 'stack'. This permits each layer to communicate and address the corresponding stack in other layers. Additionally, each layer deals with other two layers, below and above it. Therefore, the layering approach is a method to redesign the communication system vertically. Moreover, the services or the functions provided by any layer depend exclusively on the layer below it. In addition, each layer of the stack has a peer interfacing with the same layer on different nodes in the network (MIT6.02 2010). In the 1980's, the International Standards Organization (ISO) has developed a model for a network system, called as Open Systems Interconnection (OSI) model. This model has seven layers and is mostly used as a reference in the networking world as shown in Figure 2.1 (Shirazi 2009).

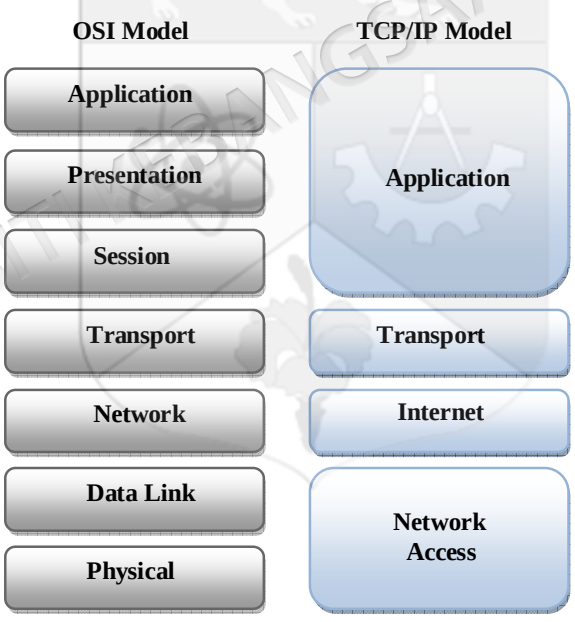


Figure 2.1 OSI model and TCP/IP model

Similarly, the TCP/IP model also has four layers as illustrated in Figure 2.1. Even though the TCP/IP model is not widely used, its layering assists in protocol design, because protocols that operate at a specific layer define the information that they act upon and define the interface to the layers above and below.

Furthermore, it prevents the technological or potential changes in one layer from affecting other layers above and below. The standard layered model used in the OSI model includes seven layers as shown in Figure 2.1 (Alex 2006). The functionalities of each layer is explained as follows:

1. The application layer is very much limited and has small role, it only provides end-user services, like e-mail and other Internet applications.

2. The presentation layer is responsible of data managements, data delivery, and data formatting of the upper layer (Application Layer) such as data conversation, data compressing, and data de-compressing.

3. The session layer manages the application processes of the end-user and the sessions of communication based on the request and response between applications such as, Authentication/Authorization.

4. The transport layer guarantees the data transfer towards the end-to-end terminals and provides other services such as, flow rate control, reliability, and connection orientation of the data stream.

5. The network layer transfers different data sequences from source to destination over one or many networks in same time. Furthermore, it maintains the Quality of Service (QoS) task and is responsible for packets routing and the routing within intermediate routers.

6. The data link layer is responsible for data transfer between networks entities and provide transmit/receive packets. Additionally, it resolves hardware addresses to detect the error in correction that may be happen in physical layer.

7. Physical Layer is the most complex layer in the OSI model and includes the main hardware materials used for data transmission in network such as, physical layer tools are physical cables, air link, and the transmission medium.

The fourth layer in the layered model is transport layer, which is responsible of transmitting services between any two applications in the network. These applications include, Simple Mail Transfer Protocol (SMTP), remote login such as Terminal emulation program for TCP/IP networks (TELNET), file transfer like File Transfer Protocol (FTP), web browsers such as Hypertext Transfer Protocol (HTTP), remote file systems such as Network File System (NFS), name-to-address translation such as Domain Naming System (DNS), voice and video streaming (Antila 2005).

Some protocols are connection oriented, this means that the transport layer can preserve the packets track and retransmit the other packets, which fail to reach. The best-known example of the fourth layer is TCP. In principle, the application can change from TCP to some other protocol easily, and particularly when the other protocol provides semantics similar to TCP (reliable and in-order delivery). Hence, one of the benefits of layering is the ability to change a layer without requiring other layers to change, as long as the service provided by the layer remains completed (MIT6.02 2010).

2.3 TRANSMISSION CONTROL PROTOCOL

TCP is a basic communication language, and a connection-oriented protocol tied with the transport layer. TCP comprises a collection of rules and procedures to control communication over links. Many TCP variants have been modified and developed to achieve the requirements of communications. Most of the current versions of TCP include a set of algorithms to control the congestion for critical links in network to improve the network throughput (Qureshi 2009). Of late, the TCP has grown rapidly to meet the increasing demand to transfer the media over high-speed links, which run on top of TCP. Furthermore, the TCP designed for wired networks can also be used in wireless networks due to the significant features such as, flow control, reliability, congestion control, and connection management for the network connections. The TCP is capable of adjusting its congestion window size, to improve the performance, but that may cause further performance degradation. This represents serious problem in mobile networks, which have rapid topological changes (Henna 2009).

Besides that, the TCP divides the sequenced data stream into packets, and confirms the packets delivery with the possibility of losing IP layer, retransmit, reorders, or packets duplication, and monitoring the network band capacity to avoid congestions.

Each TCP sender can regulate the size of the congestion window using the congestion control mechanism and the TCP can dynamically regulate the window size, depending on the packets acknowledgment (ACK) or by the occurrences of packets losses. If the congestion window is constant, the ACK timing, of the sent packets depends on the ACK of the first set of packets (early packets). Furthermore, the TCP window also depends on ACK clock and it calculates the sender flow rate, and when the required time to send packet and receive ACK –Round Trip Time (RTT) - changes with different values, the TCP sliding window determines the mean of the transmission rate of completed window per average RTT. Figure 2.2 illustrates an example of three packets transmitted using TCP sender and acknowledged with separated ACK by TCP receiver.

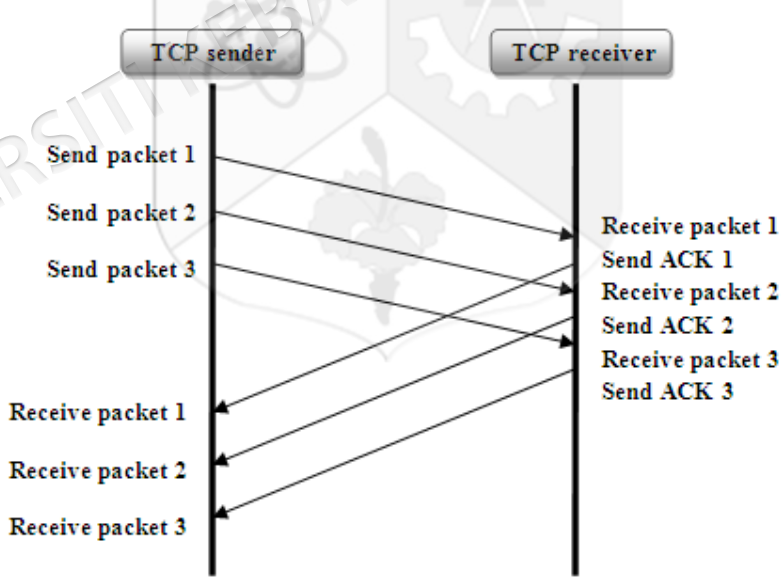


Figure 2.2 Three packets transmitting and acknowledging using TCP

The transmission window dependently controls the received ACKs for each RTT and the parameters and the control mechanism indicates the general differences between TCP versions. This controlling is used to obtain high packets rate with minimum losses by avoiding the network overloading and at the same time, it provides optimum sharing to the network bandwidth among connections (Möller 2005).

As discussed earlier, many TCP variants have been developed for different networks and applications. The TCP Tahoe and TCP Reno are mostly applied over many wired and wireless applications, because they include effective congestion control mechanisms. These mechanisms provide different size of congestion windows depending on ACK status. Thus, when packets are acknowledged the window size increases, or it will decrease when detected packets are lost. In TCP Tahoe, Reno, and Vegas (Vegas one of earlier TCP variants), the congestion control algorithm permits the window size to increase by one segment in every RTT. This increment stops when the window size reaches the congestion point.

Basically, the TCP seeks to provide reliability to the data transmitted between two hosts and it tries to apply set of rules to handle loss of packets, which results from physical errors in transmission or due to the congestion in cross traffics (Moraru et al. 2003) in internet or cellular systems. The provided reliability is hardly achieved by TCP because it is not easy to determine the available bandwidth for TCP packets flow, due to the complexity to indicate the effects of the congestion control of TCP and the network dynamically (Abrahamsson et al. 2002). In addition, understanding the behaviour and the approaches of TCP is essential to enhance its performance in wireless channels. Consequently, many researches have been geared up in this regard, however some of the problems have already been solved, but many other issues remain unaddressed (Antila 2005).

The following sections illustrate the main features, characteristics, and parameters of standard TCP, which play important role to control the packet transmission and regulate the behaviour of the congestion window.

2.3.1 TCP Round-Trip Time and Timeout

Each packet is sent from a source to destination and it waits to receive the acknowledgment, these procedures need period time to complete. The RTT represents the time elapsed between sending data and receiving an ACK (Siyan et al. 2002). In other words, if we have two nodes, A, and B, the time taken to transmit packet from A to B and the time to receive the ACK by node A to achieve the transmission or detect the packet loss is called as RTT. That means, in every link, the RTT must be equal to twice the link latency or the propagation delay of this link (Sharma 2006). The RTT value and the propagation delay represent the main factor and the basic parameter for each network topology, whether in real networks or in modelling.

TCP continually measures the time to acknowledging packets, in order to distinguish the packets that have not reached the destination (Mahmoodi 2009). If the packet transmitted from the TCP sender to corresponding host needs time to verify the transmission by receiving ACK and if the TCP sender does not get reply (ACK) in specific period, the sender supposes that the packet has been already lost and the packet will be resent again. In the congestion control, the TCP is performed independently in slotted time of RTT on each connection. This is to estimate the throughput requirement from sender side and to adjust the receiver window correspondingly. Besides this, RTT is necessary for throughput estimation, because, sometimes the link throughput will not increase even when the link capacity has extra capacity (Wu et al. 2010).

Significantly, TCP support mechanism in the sender side helps to estimate the optimal RTT of packet period over particular link, and this mechanism is used to estimate the appropriate Timeout (RTO) period (Sarolahti & Kuznetsov 2002). With each transmitted packet, there is a timer, and when the packet needs up-normal period or the when the packet time expires the timeout occurs and TCP sender must resend the packet with new period. Therefore, each packet has RTT to send from TCP sender to its peer and then reply ACK, along with RTO to detect the decision to retransmit the unacknowledged packets (Antila 2005).

Similarly, TCP sender carefully monitors the RTT on the network to detect the beginning of congestion and when a sudden increase in the end-to-end delay happens, it is called as an implicit congestion signal. This assists to detect the congestion quickly, well before the packet losses occur (Podlesny & Williamson 2010).

The timeout of the fake retransmission will break the conservation rules of packet transmission. These rules need the number of expectant packets that have been separated from the adjustment made by congestion control. Nevertheless, after the retransmission timeout, the TCP sender proceeds to slow-start phases for each packet, and then the congestion window will be back to initial size after timeout and this minimizes the performance of TCP (Sarolahti 2007). Normally, each implementation of TCP should depend on the packets transferred and the estimated RTT. The estimation of RTT is very important factor to evaluate the performance of TCP, and all implementation processes depend on packets dropped and the retransmission of these packets.

Karn and Partridge (1987) first initiated the development of TCP, when they published a manuscript of improving RTT in TCP. However, they have not registered as RFC, but have investigated some implementations of TCP. Later, the developed mechanism was registered as RFC 1122 and combined with Jacobson's (1988) mechanism for addressing congestion over networks links. The RFC is the acronym of Request for Comments, started in 1969 as a series of events for Internet and each new enhancements, additions, or modifications recommends by an RFC number. Actually, Jacobson (1988) has supposed the significant impact in TCP concept. Jacobson has proposed new mechanism to adjust the timeout period, which enables the TCP to execute low pass filter to the last RTT to calculate the next one. This algorithm is still used by most of current TCP versions (Haeri & Rad 2004). Among all the TCP source variants, only the TCP Vegas does not support the congestion control algorithm developed by Jacobson, but it applies other congestion control based on the RTT estimation. However, Vegas congestion control mechanism can provide the same packet rate but only with small network traffics (Sing & Soh 2005). Practically, Vegas, estimates the difference between the real input and the expected packet rate (Chan et al. 2004).

2.3.2 Connection-Oriented

TCP is a protocol, which provides a reliable link for end-to-end connections using algorithms to confirm the reliability of links by requesting the acknowledgment from the receiver. For that reason, TCP uses a transport layer protocol, because it provides a connection-oriented link and reliably delivers the packets transferred over unreliable links. It is noteworthy, that TCP is not based on the other low network layer, but it depends on the specifications of links (Subedi et al. 2008; Rahman et al. 2008).

Generally, TCP is very complicated protocol but it provides reliability, warranty and connection for data streaming over all its applications, as it exploits the acknowledge form receiver to determine the needs for packet retransmission. On other hand, TCP receiver can regulate the flow of data streaming form the sender by regulating the window size in a way to avoid overflow (Qureshi et al. 2009). Additionally, the TCP can provide synchronization to a huge number of connections (Swales 1999). Besides that, the connection-oriented facility enables the users to send packets only when the link between the two hosts is established, and at this point, TCP allows exchanging data between TCP peers.

The TCP connections include three main phases, connection establishment, data transmission, and connection termination. In connection establishment, the client attempts to associate with the corresponding server, while the server itself must be connected to other ports and must be ready to open up for connections; this process is called as termed passive open, and when the passive open is established, the client will start the active open. The last end between each TCP peers is termed as “socket”. The socket is defined as the integration between the source and the destination of the hosts addresses and ports, and each arrived packet must be identified with a corresponding socket. All the TCP peers are directly connected with others by socket connection and all the packet can write and read to any socket in the transport layer by interfacing it with the IP layer, which is beneath to it in the layering structure.

2.3.3 Reliable Delivery

TCP data stream is divided into sequence of packets and guarantees reliable delivery when packets are lost or duplicated and this avoids the network to reach congestion state or overloading (Möller 2005). The TCP is designed with application convenience, to provide reliable delivery service, and when the network loses one segment, then TCP must buffer and delay all segments through next RTT, until the sender successfully retransmits the lost segment (Iyengar et al. 2011). After packets delivery, the packets are reassembled to the ordinary form. This process is very complex, due to the problem in losing the related packets within transmission period, and this force the TCP to determine the lost packets and retransmit the missed packets again. When TCP fully covers the packets losses, packets duplication, delayed packets, data corruption, transmission synchronization, and data congestion, the collection of these procedures represent the reliable TCP delivery.

The functional overlap between network and transport layers reduces the stress in networks traffic, thus the network layer can also provide reliable delivery services. But the major advantage comes from TCP, when the network gets full transmission rate in same time with the guaranty to avoid congestion and also ensures that all applications operate cooperatively with other connections and terminals. Spontaneously, many new problems arise while using TCP in heterogeneous networks. In addition, many researchers, who have dealt with TCP over wireless channels, have proved that TCP does not always suffer from missing packets, because of wireless links, but also due to the inadequate delivery services introduced from the lower layers. Essentially, the TCP retransmission happens because not only the packets are lost or timeout expires, but also due to physical reasons. However, all these reasons will push the TCP sender to unnecessary slow-start phase and resend segments. Despite this, TCP is capable of providing a reliable oriented delivery for data stream, but it does not explicitly assign with QoS (Hughes 2006). For example, if some segments are lost, the TCP assumes that the loss has happened due to link congestion,, however sometimes the loss in segments also happen because of the bad connectivity between the source and the user terminal (Welzl et al. 2005).

2.3.4 Flow Control

Practically, TCP is responsible for matching the transmission rate between source and destination in the network. Besides, it also explores the maximum capacity of the channel bandwidth to introduce good performance. Hence, the standard TCP uses flow control mechanism to regulate data flow (Sanadhya & Sivakumar 2011). Of late, huge volumes of data are being transmitted over networks, which have eventually created the demand of flow and congestion control. Furthermore, the modern systems require efficient, reliable, and fair utilization in order to avoid data collision that might result due to congestion. To control the packet flow, the TCP utilizes window-based to manage the packet flow rate and to detect congestion in network.

Most of TCP versions (except Vegas) control the window size (increasing or decreasing) by triggering the sender to change the window size depending on the method used in congestion avoidance (Bajkó et al. 2004). However, each TCP is required to use manual tuning mechanism, to adjust the buffers and to provide significant scaling to the network path (Fisk & Feng 2001). Each connection includes buffer to receive packets, but these buffers have limited size and it becomes risky if they exceed the permitted limits or if the packets flows faster than the standard read out time of the buffers. However, many protocols have addressed these problems including TCP, by using flow control mechanism to avoid the buffer overloading. TCP flow is controlled based on the TCP receiver points in the sender buffer and the available capacity in receiver buffer. This mechanism also forces the sender to fix the actual quantity of packets that the sender should prepare to the link pipeline. This quantity must be equal or less than the available space in receiver buffer.

Data transfer between sender and receiver requires high degree of synchronization and compatibility to arrange the transmission process. Thus, the TCP flow control allows sending packets from sender to receiver by fully matching them and taking into account, the receiver read rate.

TCP flow control organizes the size of sliding window but the efficiency of flow control reduces due to the receiver buffer size. In other word, even flow control mechanism can provide large window, however, the limited size of buffers can degrade the total performance of data transmission (Luglio et al. 2004).

Flow control cannot be able to independently limit the congestion, because TCP receives a large window of packets, while the receive-window is limited. As a result, the sending rate and the flow control are directly related with congestion control and the transmitted data subjects are related with the congestion control conditions (Carney 2008). Therefore, many researchers have proposed a number of modifications in terms of flow and congestion control to get an adaptive TCP, which can be reliable to modern telecommunication networks. These modifications have now become as requirements such as slow-start and congestion avoidance mechanisms. These two algorithms are included in most of the TCP variants to solve some risks in network connections (Corral et al. 2000.

2.3.5 Sliding Window

TCP uses sliding window to control the packet flow by detecting the real amount of packets that are ready to be received by TCP host in specific time. Furthermore, the sliding window determines the maximum packet flow from the TCP sender to the receiver without acknowledgment. Sliding window is the technique of enabling the TCP sender to send a number of packets even without getting the required acknowledgment. The sliding window technique allows multiple packets to be transmitted automatically, and it controls the packet sending in order to avoid buffer overloading at the receiver. Generally, each transferred packet carries the number of sequence and the bytes organization in the sending state. However, in receiving state, the packets include the acknowledgment of the bytes, which are successfully received. This means that, the transmission line consists of two-way packets transmission (bidirectional), one for number of sequences and the other for acknowledgments. In fact, these two functions are very important to facilitate the communication links among TCP hosts and to ensure that all sent packets received successfully in orderly sequence.

Beyond any doubt, the function of sliding window increases the comprehensive throughput of TCP peers and thus for the network en bloc. The size of the window depends on the following factors (Chappell 2001):

1. The amount of traffic allowed on the network.
2. The TCP buffer size of that the receiver has advertised.

Sliding window allows the TCP sender to pump the possible packets in channel pipeline, without waiting for any notification from the receiver. While the size (width) of the sliding window represents the packet amount that can be injected by the sender into network, but without acknowledging the sender frequently.

Even though the sliding window allows the TCP sender to send packets without acknowledging, however, when the sender has not received ACK from receiver for long period, the sliding window forces the sender to stop sending more packets (Koga 2009). Nevertheless, the sliding window requires many resources, and it is not fairly considered as an ideal choice for real time communication systems (Wojek et al. 2008).

2.3.6 Congestion Control

The massive and rapid growth in Internet propagation and with the widespread use of TCP/IP, the congestion control mechanism becomes the key factor, which influences the level of performance of the amount of data stream within networks. For example, Internet congestion is one of the main issues in computer network, especially the rapidly growing network of Internet that necessarily has to be controlled. Generally, congestion happens when the number of received packages of a node is more than its output capacity. TCP Tahoe is the first TCP variant that included the first congestion control algorithm, which was developed by Jacobson and Karels (1986). The Tahoe has been modified in order to develop enhanced TCP variants with different congestion control (Jacobson 1988; La et al. 1998).

The window-based congestion control technique employed by TCP, tries to adjust the data flow rate speed, by adjusting the size of window, to avoid the network congestion. In addition, it fairly shares the network bandwidth over all possible connections (Kodama et al. 2008; Iguchi et al. 2005). The implementation of early TCP versions includes a simplified go-back-n model, but it does not include any assumption to the congestion control. In this model, the flow rate of data transmitted from sender to receiver does not wait for acknowledgment to send new data, but when the receiver successfully receives the segment without error, the receiver sends acknowledgment to the sender. On other hand, when ACK gets timeout, the sender will retransmit all the segments, beginning from the oldest lost segment (Fahmy & Karwa 2000).

The performance of TCP variants are directly affected by their own congestion control mechanisms and the packets amount transferred over network connections indicates the work and the behaviour of the congestion window. RFC 793 has standardized the first TCP version with basic configuration of window-based flow control. The TCP Tahoe represents the second generation of TCP versions, which includes two new techniques such as, congestion avoidance and fast retransmission. Reno is the third version of the first developed series and it is standardized in RFC 2011. The Reno has extended the congestion control mechanism by using fast recovery algorithm (Lai & Yao 2001).

Mostly, TCP variants have considered the properties and the characteristics of wired networks and do not depend on the lower network layers. Certainly, the congestion control of TCP does not performance well in heterogeneous networks and in high bandwidth (Subedi et al. 2008). This is mainly due to the slow response of congestion control over large bandwidth connection and the poor utilization of the available bandwidth (Sharma 2006). Generally, the wireless networks fail too often, hence, since the past decade the research community has focused on the performance of TCP over wireless channels to propose new approaches and new techniques. It is noteworthy that the expansion in modern high-speed and wired-wireless networks is significantly drawing the attention of researchers.

Nevertheless, modern high-speed and wired-wireless networks still support some ordinary TCP variants; however, many studies are focusing on enhancing the TCPs (Wang & Yuan 2008). Generally, due to the inability to understand the conditions of network, the standard TCP variants will be unable to fully adjust with the limited resources. In addition, the TCP variants will not be able to recognize the lost in packets if it happened by congestion or randomly. It can be concluded that all the above-mentioned issues lead the standard to perform dismally in wireless networks. In some standard TCP variants, such as, Reno, the congestion control exponentially increases the packets over the connection, and this slow-start increment must be controlled to avoid the expected overflow of the receiver buffer.

One of the TCP modification approaches is used for estimating the available bandwidth to provide fair sharing to all flows, and to adjust the window according to the available bandwidth and flows number (Wang & Yuan 2008). Other TCP variants are used as indicators to get accurate estimation of the available bandwidth and then to adjust the flow rate over the paths. In fact, the bandwidth estimation depends on many complex parameters and factors such as traffic stability of the network and the path length. Fundamentally, the congestion control mechanism involves into four phases or four algorithms; slow-start, congestion avoidance, fast retransmit, and fast recovery.

a. Slow-start phase

TCP sender to adjust data flow rate to the receiver in sending initialization uses slow-start mechanism. The process of new slow-start begins with every acknowledgement received from the receiver; therefore, the transmission rate from TCP sender fully depends on the acknowledgements returned by the TCP receiver. Even though the slow-start technique represents a part of the congestion control, it also can significantly influence the behaviour and performance of networks (Law & Hung 2001). The slow-start activates after the connection is established or after retransmission timeout. The straightforward objectives of slow-start mechanism assist the sender to realize the available bandwidth in the network by progressively enlarging the quantity of segments introduced into the network pipeline (Wang et al. 2000).

The creative slow-start process produced by Jacobson (1988) begins using a congestion window of only one segment size, and with every ACK, it increases the size of *cwnd* by one additional segment. This reason influences the *cwnd* to produce exponential increments of the segment added into the network for each RTT (Cavendish et al. 2009). The slow-start phase is truly not slow, when the network is not congested and or when the network response time is good. For illustration, the major positive transmission and acknowledgement of the segment will increase the window to double segments. When these two segments are successfully transmitted and the acknowledgements are achieved, the window size enlarges to four segments. Later, the window size is enlarged to eight segments, then sixteen segments and so on until it reaches the full size of window promoted by the receiver or until the congestion finally occurs. As shown in Figure 2.3, when connection is established, the *cwnd* is set to one segment at the beginning and increased by one segment for each successful ACK received (Allcock et al. 2005).

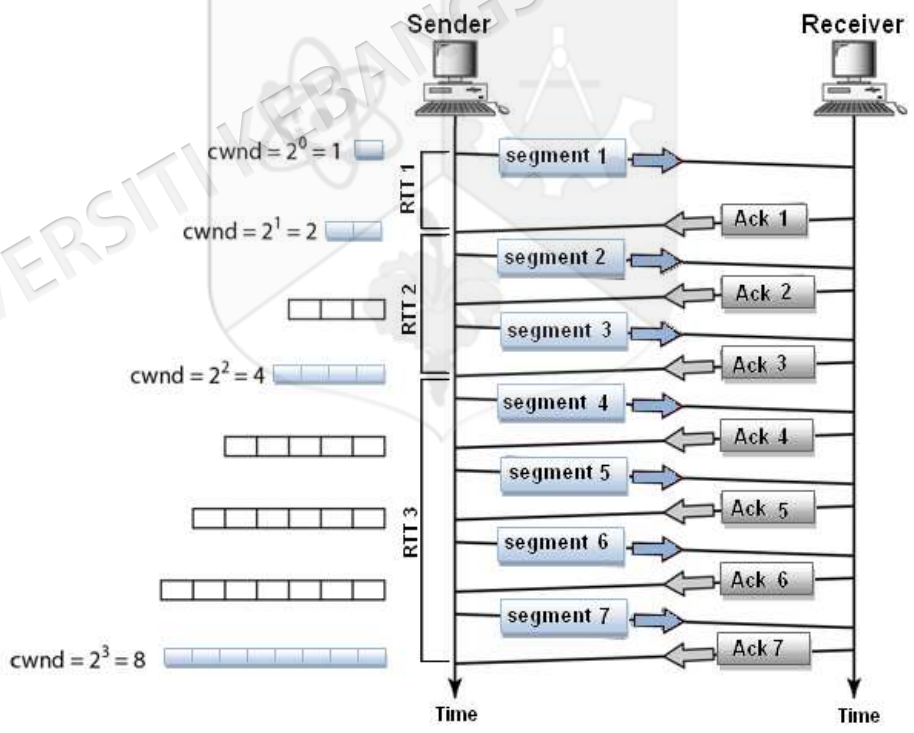


Figure 2.3 Exponential increments in TCP slow-start

That means, slow-start begins with one segment for *cwnd* (*cwnd=1*) and with each successful ACK, the *cwnd* is increased by one *(cwnd=cwnd+1*). While the exponential growth of *cwnd* for each RTT will duplicate the *cwnd* size (*cwnd =2* cwnd*) until the *cwnd* reaches the congestion point of the link.

Commonly, the TCP slow-start is applied via two variables such as, congestion window (*cwnd*) and slow-start threshold (*ssthresh*). The congestion window is a self-induced transmitting window at the sender side and it will increase TCP transmitting capacity. While the variable (*ssthresh*) is the edge for formative and the point at which TCP leaves the period of slow-start. The slow-start stage begins in the start-up when a new connection is established, and remains until it reaches a certain *ssthresh*. At this point, the TCP sender goes to the next stage, which is congestion avoidance, wherever it explores extra bandwidth by increasing the size of congestion window gradually (Cheng et al. 2005). The exiting of slow-start stage indicates that, the TCP connection has a stabile state and the congestion window carefully ties the capacity of the network path. However, the congestion window will not change geometrically, but it will move linearly when the connection goes out of slow-start. The slow-start mechanism has been discussed in Chapter III along with the enhanced slow-start mechanism.

b. Congestion avoidance phase

Jacobson (1986) has proposed congestion avoidance mechanism to fix the Internet failure problem and it is currently used in TCP implementations. Once the threshold value is reached, the TCP slows the rate of increasing of the window size, and when TCP transmission rate exceeds the network link capacity the packet loss occurs. The TCP immediately detects this packet loss and reduces the congestion window size to almost half of its current value, and then the congestion condition assigns TCP sender to enter into the congestion avoidance phase. Figure 2.4 illustrates the integrated operations and the function relationship between slow-start and congestion avoidance phases.

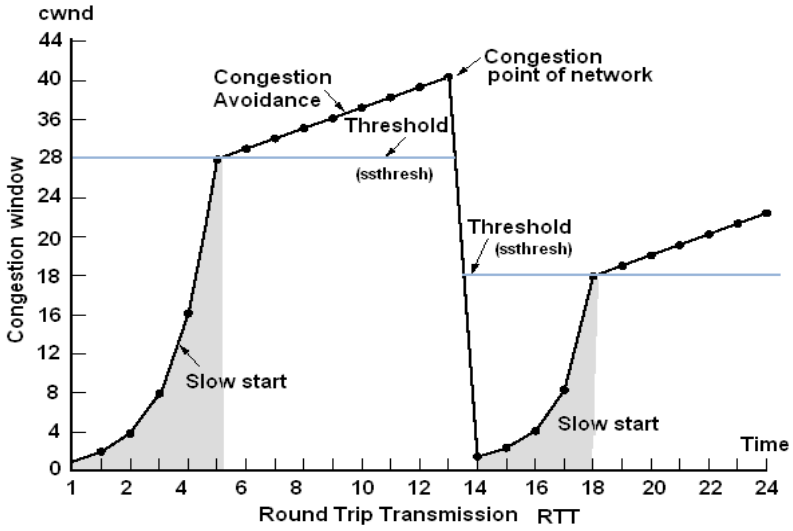


Figure 2.4 Combined operations of slow-start and congestion avoidance

The congestion avoidance mechanism is implemented when the congestion window size is larger than the recent slow-start threshold. In congestion avoidance, the dimension of one full-sized segment per RTT enlarges the congestion window and it is reduced to half of its earlier size when TCP sender senses congestion (Sarolahti 2007). This happens in some TCP variants such as Reno, but in Tahoe, the slow-start begins from *cwnd=1* and not from half of its previous size. Commonly, TCP congestion window grows in slow-start phase, and geometrically will increase the window until it exceeds *ssthresh*. When the *cwnd* is larger *ssthresh*, the TCP reaches the congestion avoidance stage, and in this stage, the prime objective is to keep high throughput and to avoid congestion. Therefore, the TCP oscillates its approach to the correct transmission rate.

Furthermore, if the TCP discovers a loss in segment it supposes that the congestion has occurred in the network path. As a remedial act, the TCP decreases the rate of flow by decreasing *cwnd*. However, if *cwnd* decreases, the TCP returns to slow-start phase. The congestion avoidance mechanism forces the TCP to steady the conditions to be below state of acceptable packet losses and it increases the congestion window in a constant volume for every RTT (Mathis et al. 1997).

The stability of congestion avoidance mechanism is used to save the transmission flow rate, which oscillates near the link capacity. This is accomplished by estimating the slow-start period and the transfer amount that increases the predictably in a linear style (Eddy & Swami 2005).

Congestion avoidance and slow-start are progressive mechanisms and have different objectives. If congestion happens, the TCP slows down the segments transmission rate of the network and again requests slow-start. This means that practically these two mechanisms are executed together (Parziale 2006). Furthermore, in initial transferring of data, the slow-start is used to clarify the sending route; however, the slow-start enforces the network connection to drop one or more packets due to the congestion, and when this occurs, the congestion avoidance is used to slow the flow rate. During this process, the timer of segment retransmission expires or receives duplicated ACKs, which will indicate the sender that congestion has occurred. Consequently, the sender instantly changes the size of transmission window to one-half of the current window size. Nevertheless, when the congestion is detected by a timeout, the congestion window is reset to one segment, which automatically forces the sender to enter the slow-start stage as shown in Figure 2.5.

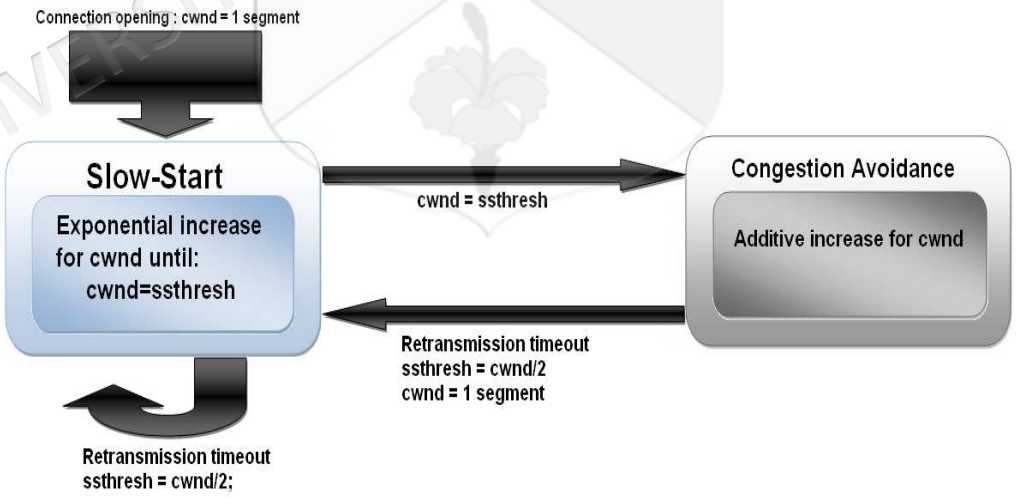


Figure 2.5 Slow-Start and congestion avoidance cycling

As shown in Figure 2.5, when *cwnd* > *ssthresh*, the TCP signals to enter congestion avoidance mode, while if *cwnd* = *ssthresh*, either slow-start or congestion avoidance may be used (Bansal 2005). In Figure 2.6, the *cwnd* size is doubled for every RTT that results in exponential *cwnd* increments. This increment continues until *cwnd* reaches or exceeds the value of *ssthresh*.

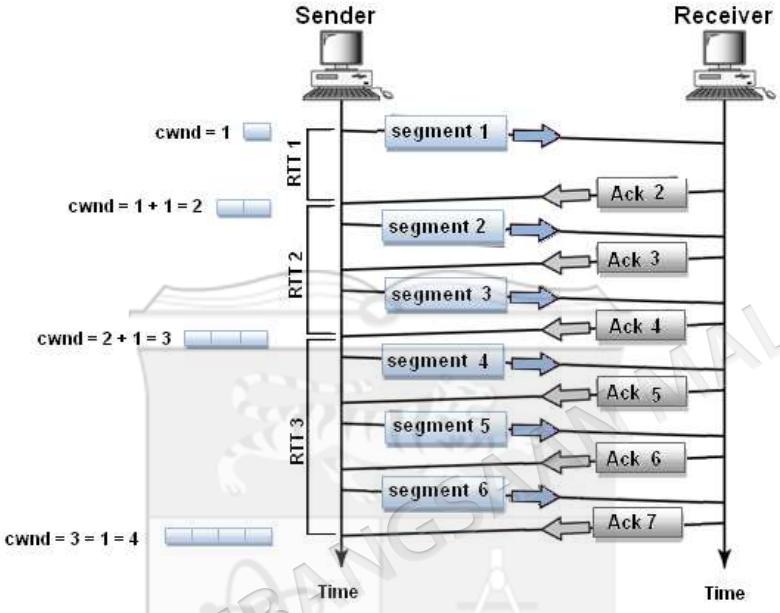


Figure 2.6 Traditional illustration of congestion avoidance

Figure 2.6 illustrates the activities of the TCP congestion control. Based on the figure initially, the TCP sender doubles the size of congestion for every RTT until the congestion window size becomes larger than *ssthresh*. Then, the congestion window is increased by one segment for each RTT as shown in Figure 2.7.

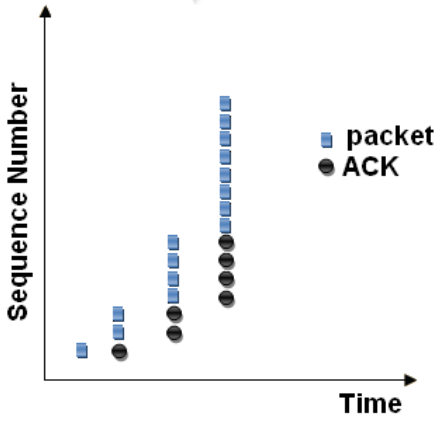


Figure 2.7 Congestion window duplication for each RTT

The scheme of congestion avoidance enables the network to run in the area of high throughput and low latency. The congestion control and congestion avoidance are self-motivated control approaches. Similar to all other control systems, they also consist of two fragments: a feedback mechanism and a control mechanism. The feedback mechanism permits the network to notify the clients (source or destination) of the recent state of the network, while the control mechanism lets the clients to regulate the load on the network. The feedback signal in a congestion avoidance structure indicates the clients whether the network is functioning under or beyond the border (Jain & Ramakrishnan 1988). In the window-increase algorithm with the slow-start phase, the source node transmits two segments for each ACK received.

As a matter of fact, the basic principles and the detailed functions of congestion avoidance mechanism and the relation with slow-start are related with the main contribution in this study. In fact, an enhanced congestion avoidance mechanism that increases the performance of TCP in large-bandwidth and low-latency wired-wireless links has been proposed in Chapter IV of this study.

c. Fast recovery and fast retransmit phases

Fundamentally, slow-start and congestion avoidance of TCP congestion control are secondary to the throughput after packet dropping. Fast retransmit and fast recovery have been considered to speed-up connection recovery without limiting the congestion avoidance features. Fast retransmit and recovery distinguish the losses in segments by duplicating the acknowledgements. A recovery process has been proposed based on fast retransmit and fast recovery mechanisms, to avoid the waiting of the retransmission timeout for every loss of segment (Olsén 2003). The mechanism of fast recovery adjusts the sending of new segments until the sender receives a non-duplicate acknowledgment. It deals with new incoming duplicate ACK as a sign for the dropped segment. If the *cwnd* is sufficiently inflated, then every duplicated ACK will be trigged to send new segment. Therefore, the acknowledgment clocking is conserved, and when the non-duplicated ACK reaches, the fast recovery will be completed and *cwnd* will be emptied (Gurtov 2000). As demonstrated in Figure 2.8 , once a loss for segment occurs, the TCP receiver will preserve the transmitting acknowledgment segments and specify the next predictable order number.

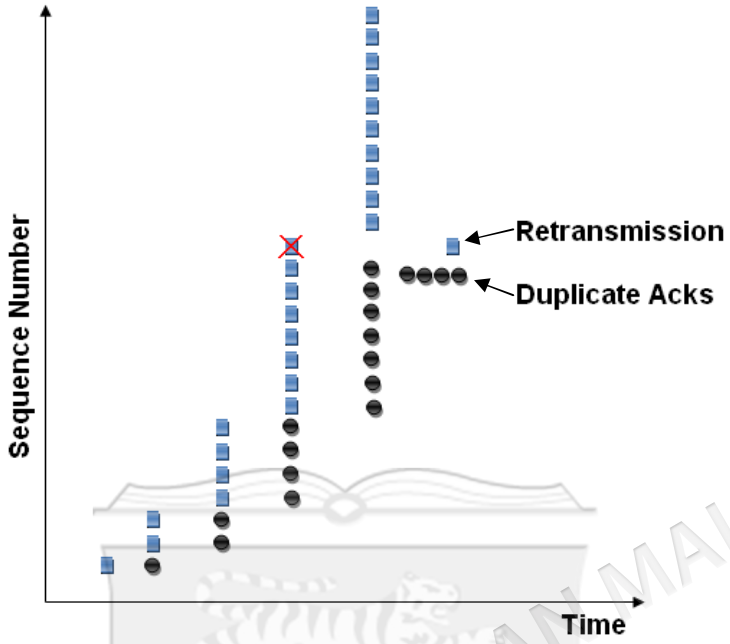


Figure 2.8 Sequence number with segment loss scenario

The sequence number would relate to the segment loss. When a single segment is lost, the TCP will create ACK for the next segments. These will enable the sender to receive duplicated ACKs. In fast retransmit, the TCP gets duplicated ACKs and it adopts to resend the segment and does not need to wait for the segment timer to expire. This process will speed the recovery of the segment losses. In fast recovery, when the segment is lost, the TCP attempts to keep the existing flow rate without returning to the slow-start.

The mechanisms of fast retransmit and fast recovery have been established to quickly solve the problem of packet losses without waiting for the RTOs. The fast retransmit mechanism was first formulated in TCP Tahoe (Wang et al. 2000); it is based on the concept of resending the unacknowledged segment after receiving three-duplicated ACKs; then it sets the size of *cwnd* to one segment and begins slow-start. The objective of receiving three-duplicated ACK is to enable the network to send segments irrespective of congestion situation (Ho et al. 2005). The fast retransmit mechanism for TCP Tahoe is illustrated in Figure 2.9.

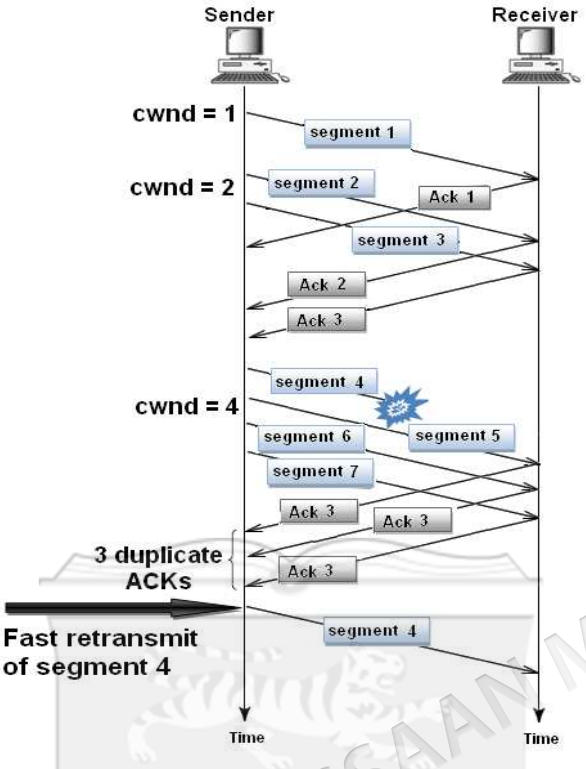


Figure 2.9 Fast retransmit mechanism of TCP Tahoe

Figure 2.10 shows fast recovery scenario of Reno congestion window and explains the relationship among timing, slow-start, and congestion avoidance phases.

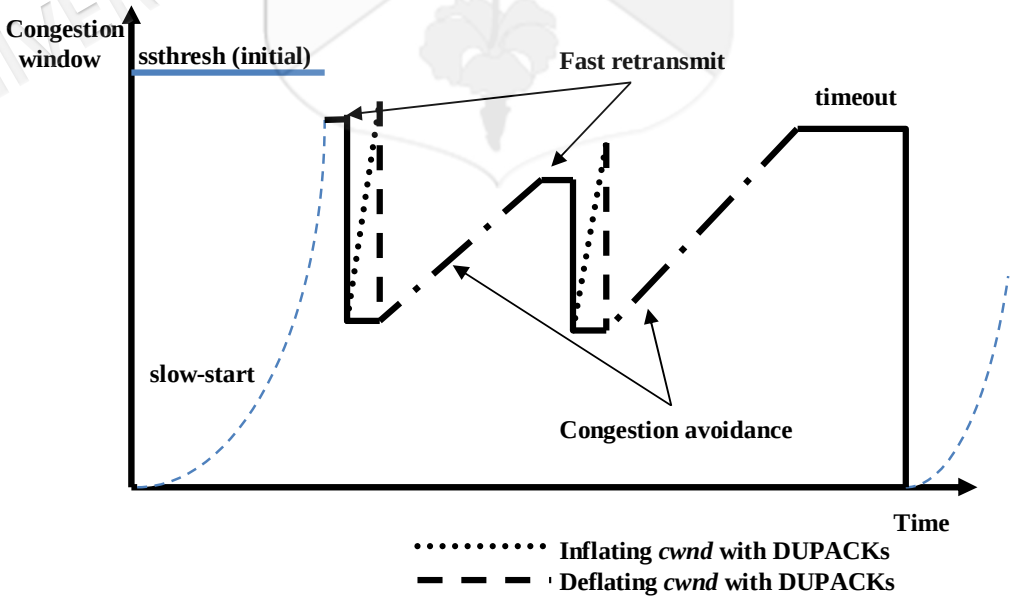


Figure 2.10 Fast recovery mechanism for TCP Reno

Fast recovery mechanism was included initially with TCP Reno (Floyd 2004); it substitutes the slow-start with congestion avoidance by decreasing the *cwnd* to be equal one-half (Lin & Kung 1998). Reno is the most popular TCP version and its congestion control algorithm is widely adopted by Internet TCP protocol. The congestion control mechanism of Reno comprises four procedures: slow-start, congestion avoidance, fast retransmit, and fast recovery. The first two stages are applied by the sender to regulate the quantity of packets inserted to the connection, but the last two procedures are used to recuperate from the losses in packets with no need to wait until RTOs (de AE Lima et al. 2003; Abrahamsson et al. 2002).

The sender in Reno uses extra incoming duplicate acknowledgment to clock the subsequent departed packets (Fall & Floyd 1996). If a loss occurs to a packet, the holes of the sequence will be rearranged and the receiver will be ready to accept new segments that are fitting with its window. The implementation of Tahoe produces fast retransmission, which is established on different suppositions. If three or more duplicated ACKs for a segment are transmitted from receiver to sender, the sender immediately recognizes that this segment has been missed and resends it before the RTO expires (Bartók & Cselényi 2001). Figure 2.11 illustrates the functional interference among slow-start, congestion avoidance, and fast recovery phases with all expected probabilities.

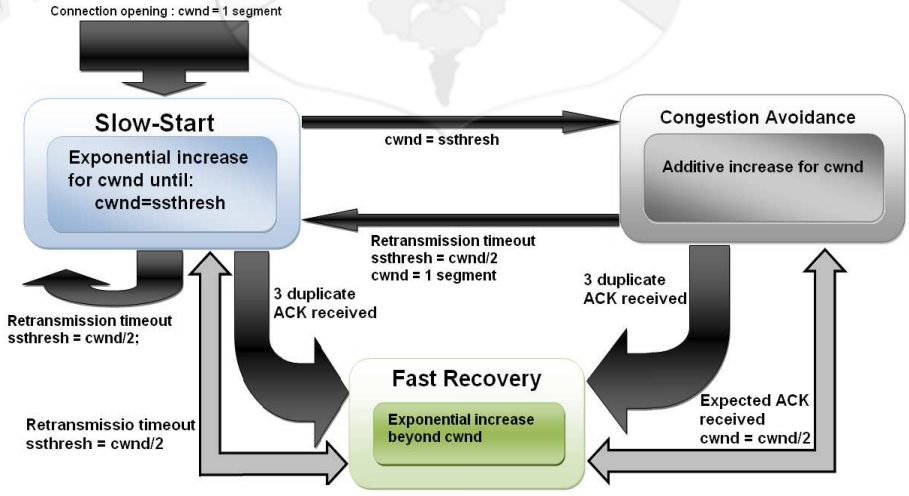


Figure 2.11 Functional interference of slow-start, congestion avoidance, and fast recovery phases

The fast recovery algorithm was developed in Newreno aimed at construing a fractional acknowledgment as a mark of additional lost packet, which will enhance the connection throughput when many data packets are lost in a single window. Therefore, the difference between Reno and Newreno is that, Reno waits for RTO for each packet then goes to slow-start, while Newreno deals with intelligent strategy, to enhance the recovery process (Floyd et al. 1999). In fact, the performance of Reno is affected when many packets are lost in the same window (Subedi et al. 2008). In addition, if segment reaches becomes out of sequence to the receiver, the host cannot distribute these segments to the end request and it needs to buffer these segments until the accurate sequence arrives.

2.4 CONGESTION CONTROL OF TCP SOURCE VARIANTS

Several of mechanisms were proposed to enhance the standard TCP versions including, Tahoe and Reno. In addition, many algorithms have been introduced for congestion control and to enhance the sharing of stability and bandwidth among flows over wired and wireless networks. The current TCP variants are not suitable for wireless networks due to radio channels effects. Therefore, the TCP needs additional modifications and developments such as, using split connections or reliable link layer approaches to run professionally in wireless links (Grieco & Mascolo 2004).

The computer networks and mobile cellular systems have evolved over the years; many computers and other equipment have become linked together via TCP as mutual protocol. Furthermore, it is hard to recognize the congestion control mechanisms that are applied in different devices in Internet, and the header of TCP does not deliver any facts to them. One more imperative problem is the manner that these mechanisms are employed in various operating systems (Moraru et al. 2003). Some successive variants of TCP grounded on the mechanisms of congestion control and avoidance have been proposed and established (Qureshi et al. 2009). The reason for the having different TCP versions is that, every version takes certain and specific ideologies and characteristics.

The evolution of computer networks and mobile cellular systems can be witnessed in the order of primary TCP (Tahoe), followed by Reno, which is supported by fast recovery, of late the newest retransmission algorithm was added to Reno to get new TCP, which is termed as Newreno. The TCP Fack is a Reno, but it uses forward acknowledgment technique. The usage of selective acknowledgment allows to the receiver, to specify some extra segment received during single duplicated ACK, where TCP Sack uses this technique. All previous variants are based on Jacobson concept, while the TCP Vegas employs its individual approaches for congestion control (Islam et al. 2009).

2.4.1 TCP Tahoe

TCP Tahoe was comprises three mechanisms to control the congestion, slow-start, congestion avoidance and fast retransmission. In Tahoe congestion control, the connections permanently begin slow-start phase for each losses in packets and when the size of window is large, the loss are infrequent. Then it will enable the connections to start the congestion avoidance phase; hence, it needs time to increase the size of the window from one segment in order to reach the value of *ssthresh* (Antila 2005). The general shape of congestion window for Tahoe is shown in Figure 2.12.

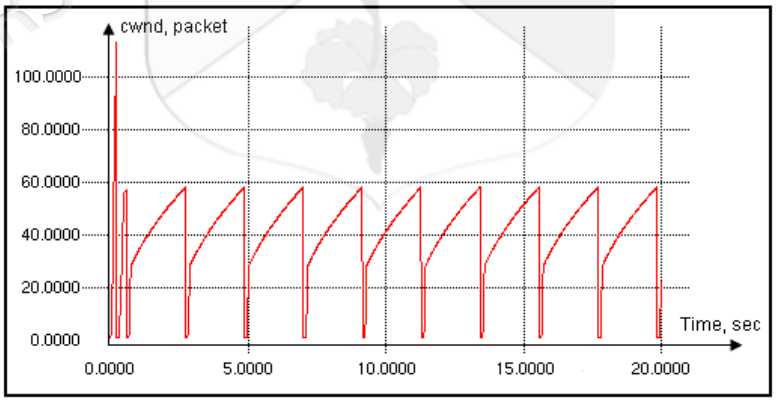


Figure 2.12 Congestion window behaviour of TCP Tahoe

The congestion window of Tahoe (and the other TCP variants) is drawn using NS-2 to observe the behaviour of *cwnd* for 20 s as a simulation time. In addition, the window size is set to 128 Kbytes and 1500 bytes as segment size.

As discussed earlier, Tahoe follows the congestion control proposed by Jacobson and is built on the 'packet conservation' concept. This means that, when the connection is established over the available bandwidth, the packet will not be hosted into the network path without taking out the previous packet. Tahoe implements this approach using ACKs to clock the departed segments, if the sender gets the required acknowledgement. The difficulty in implementing Tahoe congestion control is that, it needs to finish the timeout period to sense packet loss. Actually, in some applications, Tahoe takes more timeout interval, due to the coarse grain timeout. Moreover, Tahoe does not drive instant ACK's, but it tries to send accumulative acknowledgements. Hence, Tahoe needs to wait packet losses every time to sense timeout with an vacant network pipeline.

2.4.2 TCP Reno

In 1990, TCP Reno was released as an earlier TCP variant expanded with fast recovery. Currently, Reno is the greatest extensive TCP, but it does not perform well, if the connection is affected by the multiple data packets dropped in one window, as Reno needs to wait for timer expiration of retransmission before restarting flow of data (Fall & Floyd 1996). Reno has tried to use packet loss to determine the existing bandwidth capacity in the network. Reno begins the slow-start procedure along with TCP connection, which begins with timeouts. In this progression, it increases *cwnd* exponentially and linearly, when it reaches the *ssthresh* level to start congestion avoidance. When timeout occurs or if three duplicated ACKs are received, the fast retransmission and fast recovery are initiated. These algorithms enhance the performance of Reno by using the timeout interruption to indicate the congestion in network (Henna 2009). The congestion control of Reno does not decrease the transmission flow rate except if it notes a packet dropping, which happen only if network suffer from overload situations (Hughes 2006). As show in Figure 2.13, Reno uses two phases to increase the size of window; in slow-start and in congestion avoidance and supported by fast recovery and fast retransmit mechanisms.

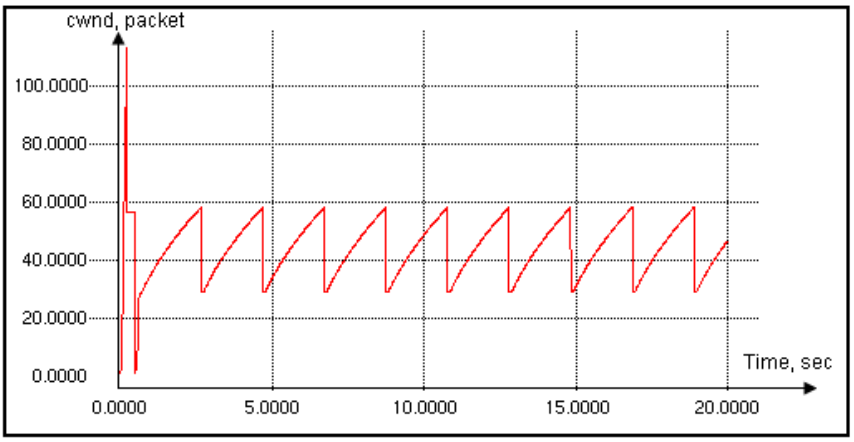


Figure 2.13 Congestion window behaviour of TCP Reno

To increase *cwnd*, the Reno will add one segment for every RTT for each ACK received and halving *cwnd* every loss occurrence for each RTT. Reno adjusts *cwnd* as following:

To increasing window size:

$$cwnd = cwnd + \frac{1}{cwnd} \tag{2.1}$$

To decreasing window size:

$$cwnd = cwnd - \frac{cwnd}{2} \tag{2.2}$$

If the bandwidth of the connection is constant, Reno repeats the procedures of increasing and decreasing the window size. Then, Reno sets *ssthresh* to half of *cwnd* and set *cwnd* to be equal current value of *ssthresh*. When duplicating the received ACK, the *cwnd* increases by one segment, and when *cwnd* value is larger than the packet amount in network pipeline, then it sends single segment, else remains in waiting mode. Reno recalls the characteristics of Tahoe, such as slow-starts and the timer of retransmission. It includes significant intelligence technique by early detecting the packets losses. Other significant change made by Reno is that, when a packet is lost it does not decrease *cwnd* to be equal one segment because that empties the pipeline, but it only decreases the flow rate and continue to the congestion avoidance, same as Tahoe.

The performance of Reno will be better if less number of packets is lost. If more packets are lost in one window, Reno starts to behave like Tahoe, because the Reno congestion control is able to recognize only one packet loss, while in multiple loss, the info for the other packet losses comes when the acknowledgment of the first retransmitted segment arrives to the sender after single RTT.

2.4.3 TCP Newreno

Newreno is an enhanced version of TCP Reno and was developed in 1996. According to Moraru et al (2003) is capable of addressing issues related to timeout in case of multiple packets loss in one window of data. Apart from solving the timeout problem, Newreno is capable of resending single packet for each RTT. The congestion window of Newreno TCP is illustrated in Figure 2.14.

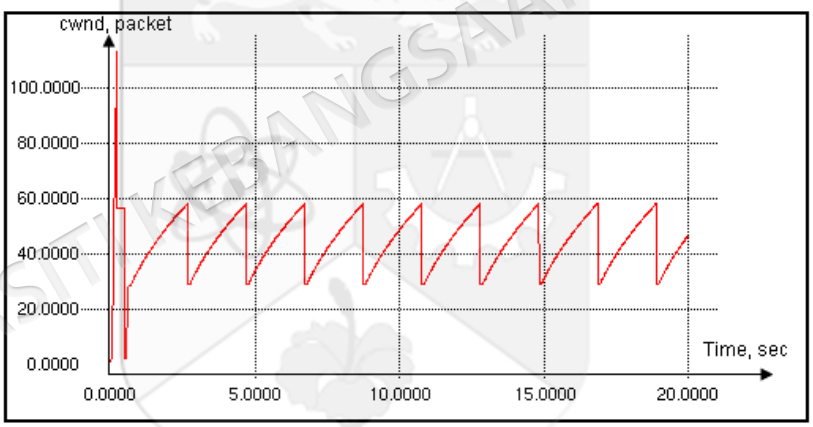


Figure 2.14 Congestion window behaviour of TCP Newreno

In Newreno, when the segment is lost, the congestion window will duplicate each RTT until it reaches *ssthresh* and the size of window will be improved by single segment for each RTT. In fast retransmit phase, the Newreno sender makes the following steps (Parvez et al. 2006):

1. The segment demanded by three duplicated acknowledgments is retransmitted.
2. *ssthresh* is set to *cwnd/2*.
3. The value of *cwnd* is set to the new value of *ssthresh* plus three segments.

During fast recovery phase, the TCP sender continuously increases the *cwnd* by a single segment for every received successive duplicated acknowledgment. It is noteworthy that in the fast recovery mechanism, the duplicated acknowledgments are capable of detecting several received segments via the receiver and this mechanism will initiate the transmission for new segment. However, similar to Tahoe, in full acknowledging the segments will be outstanding at the start of fast recovery, and then it sets the value of *cwnd* to *ssthresh* and goes to congestion avoidance phase. While the partial acknowledgment gets the subsequent segment in the pipeline that has been already missed and retransmits it, and sets the number of the received duplicated ACK's to zero (Afif et al. 2005; Nagamalai & Lee 2004).

When compared with Reno, Newreno is capable of sensing many packet losses when multiple packet losses occur in one window. Furthermore, Newreno will not leave the fast recovery phase until all the outstanding segments are acknowledged. However, the disadvantage of, Newreno is that, it needs one RTT period to detect packet loss, and it can recognize the segment lost only when the acknowledgment for primary retransmitted segment is received, then.

2.4.4 TCP Sack

TCP with selective acknowledgment (Sack) permits the receiver to openly acknowledge the out of sequence data that has been received by the sender. In Sack, the sender will not resend the data that has been "Sacked" in the period of loss recovery. Many studies have proved that the Sack technique enhances TCP throughput if the multiple packet are lost within the same window (Ekiz et al. 2011). Sack algorithm intermediates between the selective duplication resending strategy and the accumulative acknowledgment structure for TCP (Kettimuthu & Allcock 2004). Figure 2.15 shows the congestion window for TCP with Sack.

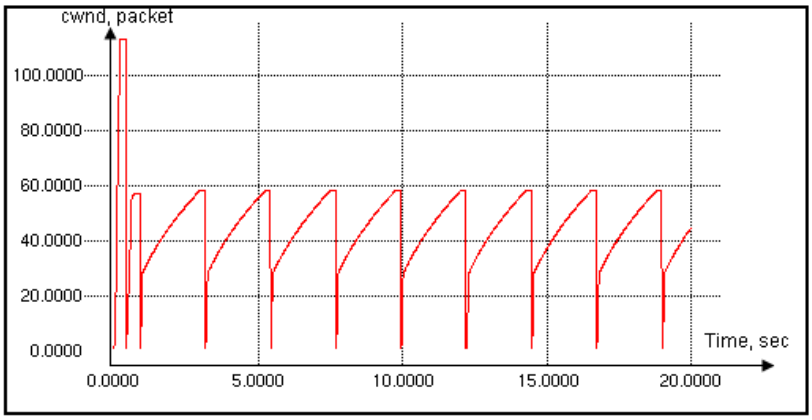


Figure 2.15 Congestion window behaviour of TCP Sack

It is easier to understand the TCP with Sack than the other two TCP algorithms such as, Tahoe and Reno. The TCP with Sack differs from Tahoe, in terms of the difficulties associated with slow-start and congestion avoidance, and differs with Reno, by performing consistently even at times of multiple dropping happens. The TCP Sack is simple and easier to implement (Floyd 1996). If Sack is not used with Reno, it performs dismally particularly if multiple packets are dropped in same window of data. These problems result from the necessity, to expect the expiration timer of retransmission before deciding to resend the data.

As Sack is an expansion of TCP's Reno and Newreno, it works along with the risks than these two variants when multiple packets losses happen. When the Reno and Newreno congestion control algorithms do not support the Sack, they are able to resend only one dropped packet per RTT. These characteristics are not included in Tahoe, as there is no limit to resend the single dropped packet for each RTT (Fall & Floyd 1996). The TCP Sack does not accumulatively, acknowledge packets, rather, it acknowledges the packets selectively. This is because every acknowledgment includes a block, which defines each acknowledged segment. Therefore, the TCP sender has an image of the acknowledged segments as well as the outstanding segments. Every time the TCP sender goes into fast recovery phase, it sets a mutable pipe, which determines the amount of data, which are still outstanding in the path of the network, and fix the congestion window to half of the recent value.

When the size of the pipeline becomes less than the size of congestion window, it detects the segments that are not still received and resends them. If there are no outstanding segments, then it will send new packets. Therefore, more than one segment loss will be sent within single RTT. The major problem with TCP Sack is that, presently the selective acknowledgement is not sent to the receiver, and it is not easy to implement the Sack.

2.4.5 TCP Fack

TCP with forward acknowledgement (Fack) is a different algorithm, which works on the upper options of TCP Sack. During recovery operation, the TCP Fack uses the info provided by the Sack to add accurate control to the data that have been injected into network pipeline. The fundamental concept of Fack mechanism depends on considering the greatest sequence number of forward selective acknowledgement, when the previous acknowledged segments are lost. This monitoring meaningfully improves the recovery process of packets losses. The Fack mechanism frequently improves the performance of TCP than the standard approaches. This mechanism performs well if the packets have been rearranged or been out of sequence in the network pipeline. This is because the holes of packets between data blocks in Sack do not take packets loss into account (Sarolahti 2002). The congestion window of TCP Fack is shown in Figure 2.16, where it can recognize different behaviour of congestion window.

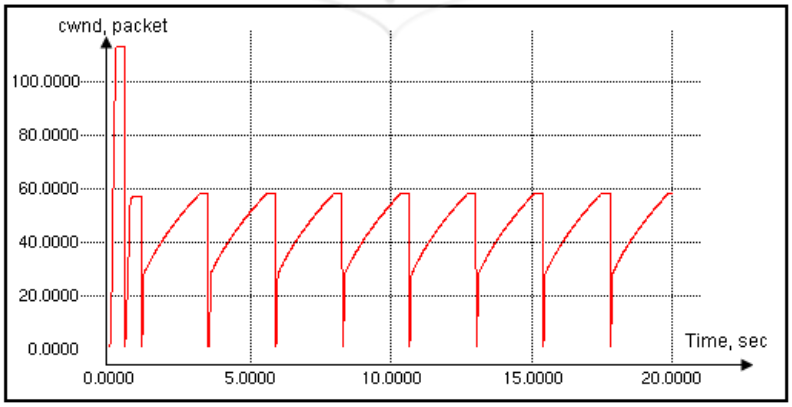


Figure 2.16 Congestion window behaviour of TCP Fack

Practically, the Fack behaves similar to Sack with some simple improvements, as it uses Sack approach to estimate the transferred data. The Fack presents a good technique to halve the size of window if congestion occurs. Therefore, if *cwnd* is instantly halved, the TCP sender stops transferring for a while and then restarts if the sufficient amount of packets reaches the destination. If the congestion happens, the *cwnd* must be halved, depending on the decrease of *cwnd*. The Fack sender recognizes congestion state after it happens, and after at least one RTT. Then, if it is in slow-start phase, the *cwnd* will duplicate as explained in slow-start section. However, even though the TCP Fack offers congestion avoidance and fast retransmits mechanisms, it still has many circumstances in recovery processes. Furthermore, it is not easy to implement the Fack over high performance networks (Tayade & Sharma 2011).

2.4.6 TCP Vegas

Brakmo et al. (1994) proposed TCP Vegas as new TCP version with novel technique in congestion avoidance phase (Jamal & Sultan 2008). The Vegas was developed with private approach, which differs from the other TCP source variants, and it performs better than Reno and can generate lower packet loss (La et al. 1999; Low et al. 2008). The Vegas is built on the estimation of RTT as a new algorithm, to detect the congestion within slow-start and congestion avoidance phases. Figure 2.17 shows the strange behaviour of Vegas congestion window, where the two main phases such as, slow-start and congestion avoidance employ different strategies to control the congestion.

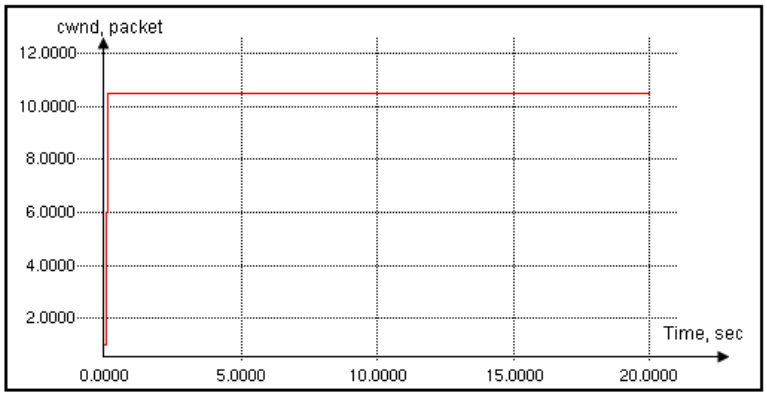


Figure 2.17 Congestion window behaviour of TCP Vegas

Vegas do not exclusively depend on the packet losses as a mark of congestion occurrence, but it discovers congestion state before the occurrence of loss. Furthermore, Vegas does not induce major changes in slow-start, retransmission, and congestion avoidance. Therefore, when a duplicated ACK is received, the Vegas will check whether the current time of the segment is larger than the RTT, then it directly resends the segment, and there is no need to wait for the three duplicated ACKs. In addition, the Vegas can detect a multiple losses in packets and then it decreases the *cwnd* only if the retransmitted segment was transmitted after the last decrement. Moreover, Vegas congestion control does not constantly increase the *cwnd* during congestion avoidance phase, but it attempts to detect the congestion early by associating the current throughput to the expected. On other hand, the development of slow-start mechanism includes the same detection process to the network congestion to decide if it needs to switch to congestion avoidance mode (Hengartner et al. 2000).

Many studies have proved that the TCP Vegas are capable of providing higher throughput than other TCP versions; however, it may be true in homogeneous network that works on top of Vegas. This is because in heterogeneous network, Vegas will be unable to achieve fair bandwidth sharing over network connections if compared with other TCP variants (Ong & Badlishah 2011).

2.4.7 Congestion Control Comparison for TCP Source Variants

Different congestion controls techniques are used with various TCP variants as explained in previous sections, and each variant comprises private congestion control mechanism. Nevertheless, some of these variants share some of the basic characteristics of congestion control, either in slow-start or in congestion avoidance or combined, to use same recovery process. Generally, TCP congestion control is grounded on Additive Increase Multiplicative Decrease (AIMD) algorithm, by halving the window size, when the congestion window suffers from packet loss, and then increases the window by one segment for each RTT. The second module of congestion control uses retransmission timer and contains the exponential increase then resets the retransmit timer, if the retransmitted packet is dropped.

The other congestion control mechanism depends on the ACK clocking, where the acknowledgments received by the sender are used to initiate the transmission for the next new packet.

TCP source variants had been discussed earlier in details, where all these variants share many features and characteristics, such as slow-start. However, the TCP Vegas does not share any features. Furthermore, all these variants follow the fundamentals of slow-start, AIMD, retransmit timers and clocking of ACK. Table 2.1 and Table 2.2, illustrates the functional structures and the actions used by six TCP variant. Moreover, the congestion control mechanism is compared with the other techniques used by the other variants. Indeed, the results of the comparison are tabulated in two tables for easier understanding.

Table 2.1 TCP congestion control algorithms for Tahoe, Reno, and Newreno

Action	**Tahoe**	**Reno**	**Newreno**
In slow-start, *cwnd* updated every ACK as:	*cwnd*+1	*cwnd*+1	*cwnd*+1
In congestion avoidance, *cwnd* updated every ACK:	*cwnd*+1/*cwnd*	*cwnd*+1/*cwnd*	*cwnd*+1/*cwnd*
Change from slow-start to congestion avoidance when:	*cwnd=ssthresh*	*cwnd=ssthresh*	same, but *ssthresh* may be estimated
Fast recovery	None	Terminate with partial or full ACK	Continue with partial ACK
ACK format	ACK	ACK	ACK

Table 2.2 TCP Congestion control algorithms for Sack, Fack, and Vegas

Action	Sack	Fack	Vegas
In slow-start, *cwnd* updated every ACK as:	*cwnd*+1	*cwnd*+1	Increase every other RTT
In congestion avoidance, *cwnd* updated every ACK:	*cwnd*+1/*cwnd*	*cwnd*+1/*cwnd*	Linear increase if expected - actual< α ; linear decrease if > β
Change from slow-start to congestion avoidance when:	*cwnd= ssthresh*	*cwnd= ssthresh*	Expected-actual < γ
Fast recovery	Continue with partial SACKs and send if pipe<*cwnd*	Send as long as out-standing data<*cwnd*	Retransmit with ACK if RTT > timeout
ACK format	SACK	SACK	ACK

From these tables, it is evident that, the slow-start phase is based on similar algorithm such as Tahoe, Reno, Newreno, Sack, and Fack, while the Vegas use its own slow-start technique. While in congestion avoidance, there are some variations and modifications, but all are based on Tahoe congestion avoidance, with some extra properties to improve the specific TCP over different networks and applications. The tables also explain the similarities and differences among these variants in terms of slow-start, congestion avoidance, switching from slow-start to congestion avoidance, fast recovery, and the acknowledging the format and process. Many proposals and trials are implemented to improve the performance of TCP. Furthermore, a number of studies have proved that the standard TCP's perform poorly over high-speed links (Mbarek et al. 2008).

2.4.8 Other TCP Congestion Control Techniques

Many TCP versions with own congestion control mechanism have been proposed to run on top of specific applications and platforms. The TCP Westwood is built on end-to-end bandwidth estimation (Mascolo et al. 2001). Particularly, the Westwood estimates the available connection bandwidth by calculating and filtering the data flow from returning acknowledgments. Then, after the congestion, it sets the congestion window and the slow-start threshold, depending on the available bandwidth (Grieco & Mascolo 2004). Some other protocols are designed with new congestion control mechanism, to work over high speed and wide area networks, such as High-Speed TCP (HSTCP) (De Souza & Agarwal 2003). The HSTCP uses the previous *cwnd* to estimate the new window size (Floyd 2003). Fast TCP (Jin et al. 2005) was proposed to maintain the stability by scaling down the sender response using RTT, to solve the problem of connection instability.

Scalable TCP (Kelly 2003) depends on AIMD algorithm. Scalable linearly increases and multiplicatively decreases the congestion window and attempts to increase the *cwnd* to the maximum size to exploit the full link bandwidth. The BIC TCP is the other new approach, which enhances the TCP performance (Xu et al. 2004). The BIC estimates the available bandwidth and sets maximum size for window to be the target window and keep the current *cwnd* as minimum window. The BIC applies a binary exploration to increase the minimum window to the average level between minimum and maximum windows for each new ACK.

The BIC TCP is further enhanced to make it less aggressive and this new TCP is called as CUBIC (Ha et al. 2008). The CUBIC TCP uses the delay among dropped packets, to regulate the *cwnd*. The same approach is based on the time delay among the dropped packets to regulate the size of congestion window used by Hamilton TCP (Leith & Shorten 2004). The H-TCP was proposed to decrease the *cwnd*, if it detects the congestion state, by using the relationship between minimum and maximum RTT, which facilitates to determine new *cwnd*.

The TCP Linux (Wei et al. 2006) was established for NS-2 platform, to provide TCP agents with Linux base. Linux is able to recover even with retransmission of packet loss, because NS-2 defaults TCP timeouts.

In addition to the previous TCP, extensions as discussed above, many other versions were designed and developed to run with different applications over wide networks variants. For example, Snoop (Chouly et al. 1993), Freeze (Goff et al. 2000), Eifel (Ludwig & Katz 2000), Hybla (Caini et al. 2006), and ESSE (Giordano et al. 2008). More and more congestion control mechanisms have been proposed to serve a specified task, but all are aimed to deliver high throughput and avoid congestion as much as possible.

2.5 TCP OVER WIRELESS NETWORKS

The expansion in wireless technologies and the extensive with the wide use of mobile devices have drawn the attention of research and technological communities towards wireless environments, such as Wireless Local Area Networks (WLANs), Wireless Wide Area Networks (WWANs), and mobile ad-hoc networks. In WLANs networks such as Wi-Fi technologies, or in WWANs such as 2.5G, 3G, 4G cellular systems, the hosts of mobile are connected by access point or by base stations. These elements are connected to the wired part of the network to provide wired-wireless platform. Unfortunately, wired and wireless networks are expressively different in terms of link reliability, bandwidth, and time of propagation delay (Chen et al. 2005). Additionally, several challenges are faced when mobile or wireless networks are associated with wired networks in transport layer side. This is because the wireless networks have various essential features, which will considerably weaken the performance of the protocol, if it is not modified. Fundamentally, these features involve mobility, links asymmetry, and channel error.

1. Connections asymmetry: As the mobiles have limited power, restricted processing capability and limited buffer space, the mobile terminal and the base station are interfaced with wireless asymmetric link.

2. Channels Errors: Generally, high error rate is expected in wireless channel due to the fading of multipath; the shadowing may drop packets in data transfer process, which leads towards the wide loss in segments or acknowledgments.

3. Mobility and handover: The cellular systems are characterized with handovers due to the user's mobility. Usually, the handovers cause short-term interruptions in connections, which in turn causes packet loss and add extra delay over networks connections.

Theoretically, the TCP is independent of the underlying layers, and if these layers reduce the reliability, then the TCP will be exposed to several shortcomings. On the other hand, wireless network is characterized with high probability of random errors and irregular connectivity of links. Commonly, when the IP packet is sent on wireless links, the IP layer observes the availability of one or more capacity, delay characteristics and loss amount, which differ with time. While the link layer employs, the accessible layer control techniques, such as retransmission scheduling, power controlling, and adapts flow rate, in order to balance the loss, latency, and capacity.

However, it is not easy to keep constant and suitable level of low latency, low losses, and high capacity, as these represent the individual characteristics of wired links. Therefore, any protocol that uses congestion control will suffer from packet losses and need to rebuild the congestion window frequently. As the result of packets dropping, some of the TCP versions decrease the *cwnd* and the flow rate, to maintain connection throughput (Mahmoodi 2009). The mechanism of congestion control for most TCP variants adapts the style and the requirements of wired networks. In wired network, the accessible bandwidth changes according to the cross traffic and the irregular routing of the network. In addition, the changing in capacity and delaying in wired links, initiate the constant values (Möller 2005).

If TCP is used over cellular infrastructure, the performance frequently reduced in both, the end-to-end and radio links employment. Researchers are focused with TCP over wireless channels in order to achieve good or at least acceptable level of end-to-end throughput.

The individualities of the actual nature and the requirements of the wireless networks are taken into account the strategies to employ TCP over wireless. For example, satellite networks involve large propagation delay, while the ad-hoc network lacks infrastructure. Moreover, the development of TCP for wireless networks need to obviously distinguish the reason behind packet loss. Researches are carried out to identify explicit techniques to notify the TCP senders, the reasons of dropping of packet, congestion, or random errors occurrences (Tian et al. 2005)

2.6 INTEGRATED APPROACHES TO IMPROVE TCP IN 4G SYSTEMS

Efficient protocols are needed to overcome the problems and the challenges of wireless networks, especially in the 4G systems as they expect high data rate. Various methodologies have been suggested to optimize the performance of TCP over 4G networks. These methodologies are classified as TCP improvement approaches or as layer limitations. The TCP improvement depends on end-to-end TCP variations, link level solution, or by split TCP connections with the help of intelligent agents. The following sections elucidate the possible solutions to develop TCP performance over 4G wireless networks.

2.6.1 Split TCP Connection Solution

The wired section of any wired-wireless network is more reliable than the wireless part in terms of connection bandwidth capacity and bit error rates. TCP performance can be improved by separating the connection in base station as shown in Figure 2.18.

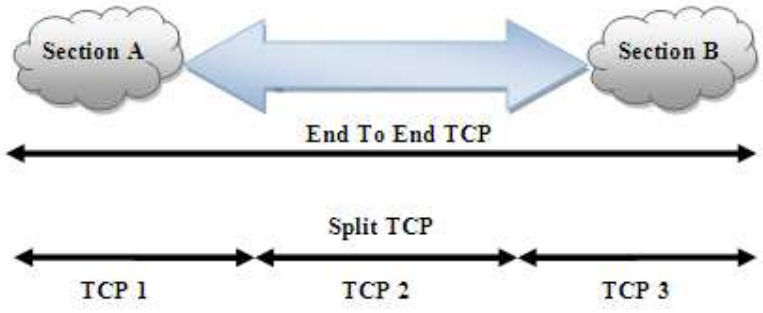


Figure 2.18 Splitting TCP connection

The idea of splitting TCP connection is to break an end-to-end TCP connection into two or three segments. Each segment is itself a complete TCP connection. Data streams are forwarded from one segment to another (buffering if necessary). For example, if the first and the last TCP segments span a low latency network, the TCP slow-start can speed up more quickly and the congestion window size will work fine. While the middle segment, should implement special features and use large windows to cope with extended latency.

Therefore, the performance of TCP can be improved only by implementing minor changes to application software. Furthermore, the splitting approach protects the wireless part from the wired part in network, by isolating the flow control in intermediate base stations from cellular networks (or routers). Hence, the wireless activities have the minimal effect and role is compared with the wired section in the network. The intermediate base stations act as terminals in wireless and fixed parts, and all end hosts are connected with this intermediate station and there is no need to acknowledge the other end (Tian et al. 2005).

Bakre & Badrinath (1995) have proposed an Indirect-TCP (I-TCP) to split TCP connection between any machines. The organization of I-TCP initially includes the connection between base station and mobile host in wireless section, and then between the fixed host and the base station in the fixed network, as illustrated in Figure 2.19.

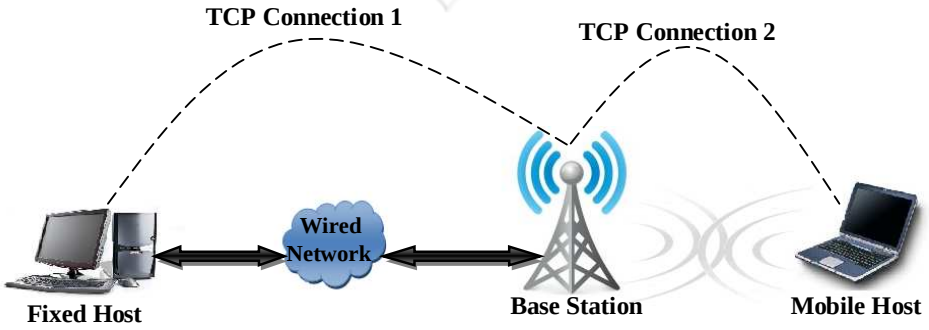


Figure 2.19 Splitting TCP connections in I-TCP

The base station initially receives the segment transmitted to mobile host and later it sends the ACK to the fixed host. After that, the segment is forwarded to the mobile host. When the mobile host moves to another cell, after connecting the establishment with fixed host, then the link info of connection is saved at the current base station and will be transferred to the next base station. Furthermore, when the end-to-end connection splits the TCP connections of the wireless part, it can benefit the link-aware TCP actions and help to handle errors of wireless links and may reduce handovers occurrence (Chen et al. 2005).

M-TCP is another approach, which is supported by the splitting concept, which divides the TCP connection into two sections. The first section lies between the base station and the fixed host, while the second section lies between the mobile host and the base station (Brown & Singh 1997). The M-TCP differs from I-TCP, because it preserves the end-to-end TCP characteristics. The M-TCP is proposed to operate on three-level hierarchy of architecture as shown in Figure 2.20.

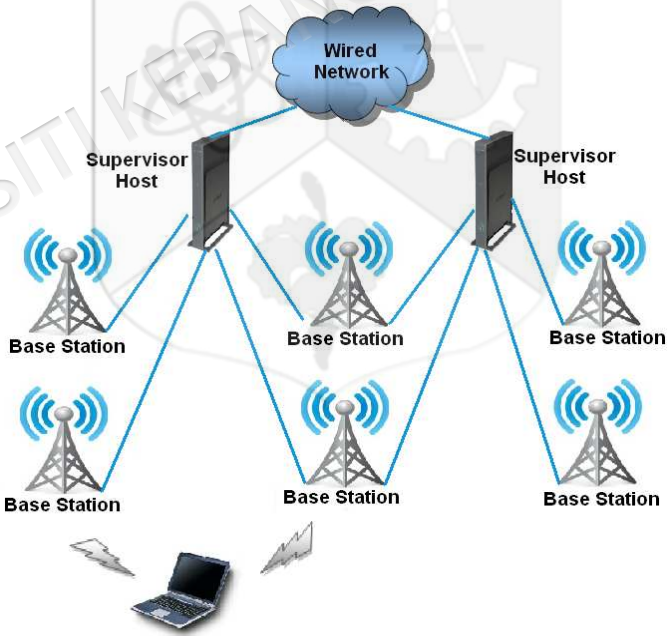


Figure 2.20 Underlying architecture of M-TCP

As shown in Figure 2.20 the mobile hosts are at the lowest level and they communicate with base station nodes in each cell. A supervisory host controls several base stations and each base station is connected to a wired network and handles most of the routing and other protocol details. It also maintains connections and handles flow control. When a wireless host moves from one cell into another, the two base stations need to perform state transfer, if the same base station controls them. The M-TCP is aimed to achieve two objectives. The first is the functionality in base station, which permits to use one supervisor host to control many base stations, which reduces the network cost. The second objective is to possibly reduce handovers rate.

2.6.2 Link Level Solution

The link layer solution provides more reliable link level, by making network transport layer shielded from wireless connection. The most popular models used in the link level are Forward Error Correction (FEC), Automatic Repeat Request (ARQ), and Hybrid ARQ (HARQ). The FEC technique can increase the probability of the packet delivery by accumulating various redundancies to the transmitted packet. While by natively retransmitting the lost packets, the ARQ technique can offer more reliability to the link layer. Therefore, reliability level at link layer depends on the persistency of ARQ and redundancy of FEC (Mahmoodi 2009).

Snoop protocol is one of the earlier link layer solutions, which introduces an extra snoop agent at the base station of the wireless medium (Balakrishnan et al. 1995). The protocol is named as Snoop as it adds snooping elements to the network layer, to observe each packet that passes through the base station in any route (Chen et al. 2005). The Snoop protocol provides a reliable solution by maintaining the end-to-end semantics of TCP, while locally recovering the wireless errors. The Snoop uses link level buffers at the base station for the following purposes:

1. To cache the packets that pass across the wireless link.
2. To retransmit the unacknowledged packets.
3. To avoid unnecessary timeouts.

In addition, the Snoop filters duplicate the acknowledgements to avoid duplicated packets. These functions are performed by two main routines such as, snoop-data and snoop-ACK (Vangala & Labrador 2002). Figure 2.21 illustrates the basic idea of Snoop protocol for transmitting data to the mobile host.

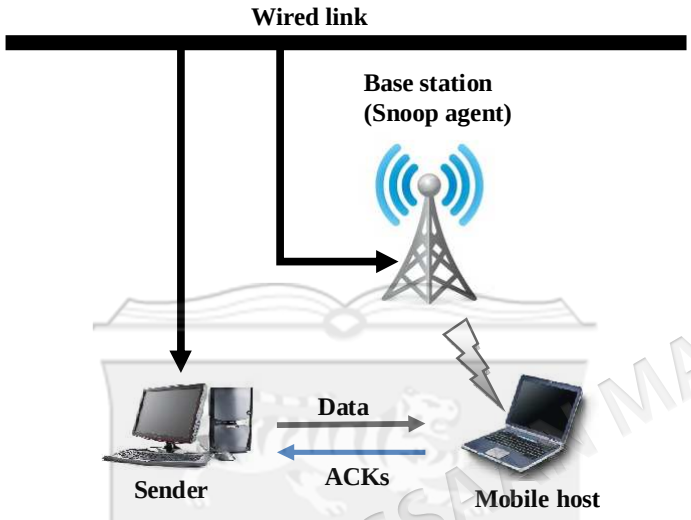


Figure 2.21 Snoop protocol model

Snoop agent can allocate timer to the sent packets, thus the lost packet may be noticed by the subsequent duplicated ACK number and by timer expiration. The main benefit of using link level protocol is to provide packet loss recovery. The link level protocol is capable of recovering packet loss, because this approach logically fits with the layering organization of network and it works independently with higher layers (Mahmoodi 2009). On the other hand, even the performance of TCP during handovers may be enhanced, but it suffers with multicast base station group and with transmission from base station to another.

2.6.3 End-to-End Solution

The end-to-end approaches can be also employed to improve TCP performance, to make the end hosts contribute in flow control. This modelling of this solution is shown in Figure 2.22.

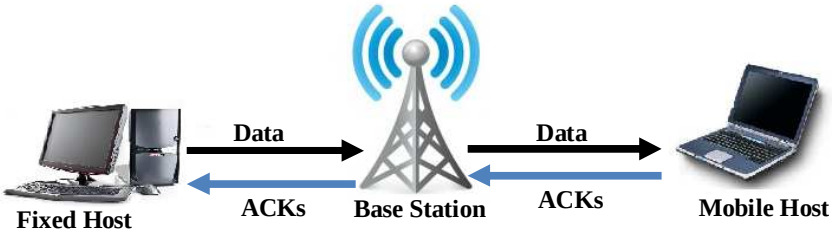


Figure 2.22 End-to-End connection of wireless TCP

The sender is responsible to regulate the congestion in network, while the receiver provides feedback to reflect network conditions. The end-to-end approach uses two congestion control mechanisms such as, proactive and reactive. In the proactive mechanism, the congestion control mechanism feedbacks from the network resources to avoid congestion. While in reactive mechanism, the sender rectifies the congestion window, if the situation of the network is congested, or exceeds the link threshold (Tian et al. 2005).

Explicit Congestion Notification (ECN) is a perfect example of end-to-end approach. The ECN uses routers, to report the congestion to TCP sender by using IP header. The ECN proposes dynamic queuing management arranged at the routers gateway and allows discovering congestion, either before losing packet or before the queue overflow (Ramakrishnan et al. 2001).

Another protocol under end-to-end strategy uses the advanced queuing management pattern at the routers (Biaz & Vaidya 2005). The main concept of this protocol is named as TCP-Casablanca, which de-randomizes the congestion loss by separating the congestion losses from the random losses form the wireless. Therefore, the TCP sender marks several sent packets and when the congestion occurs, the routers immediately drop those packets.

2.7 TCP AND SCTP OVER LTE/LTE-ADVANCED

LTE-Advanced is the continuation of 3GPP-LTE and targets to the advanced development of the requirements of LTE, in terms of throughput and coverage (Rong et al. 2010). Therefore, LTE-Advanced is not new as a radio access technology, but it is an advancement of LTE to enhance the performance. The driving force to further development of LTE towards LTE–Advanced is LTE Release-10, which provides inexpensive higher bit rates and at the same time completely fulfils the requirements set by ITU for IMT Advanced, also referred to as 4G. Developing LTE, instead of designing new radio access technology is imperative for the operators (Parkvall et al. 2011). The operators can install Release-8 (LTE) network and in future when the necessity ascends, they can upgrade to Release-10 (LTE-Advanced) (Wannstrom 2012 ; Nakamura 2009).

Fundamentally, the earlier release of LTE terminal must always be capable to access the functionality of the supporting carrier of Release-10 and to employ all the features of Release-10 of this carrier (Dahlman et al. 2011). The LTE-Advanced was accomplished in late of 2010 and it enhanced the LTE spectrum flexibility over carrier aggregation. Furthermore, it extended the multi antenna broadcast, introduced a supporting platform for the relaying, and provided an enhancement in the part of inter-cell interference coordination in heterogeneous network utilizations. Of late, the LTE-Advanced network is a promising candidate for 4G cellular systems, to run into top rates of data reaches to 100 Mbps with high mobility and 1 Gbps with low mobility. These are wanted in 4G systems, and LTE-Advanced must be capable to keep the broader of bandwidth higher than it provided by LTE (Zhang et al. 2010).

2.7.1 Architecture of LTE-Advanced

In Release 8, 3GPP identifies the requirements and the features of the architecture of (EPC) to serve as the base for the next generation systems. This identification specifies two main work objects, LTE and System Architecture Evolution (SAE).

In addition, EPC combines with Evolved Universal Terrestrial Radio Access Network (E-UTRAN) and Evolved Universal Terrestrial Radio Access (E-UTRA) to construct the network core. This core is responsible to arrange the system air interface and the radio access of the network link and terminals. The EPC provides IP connection between the external packet data network and the User Equipment (UE) using E-UTRAN. In the environment of 4G systems, the radio access network and the air interface are actuality improved, while the architecture of core network (EPC) does not need large modifications from the previous architecture of SAE. Figure 2.23 shows E-UTRAN and EPC architecture of LTE-Advanced networks (3GPP 2009b).

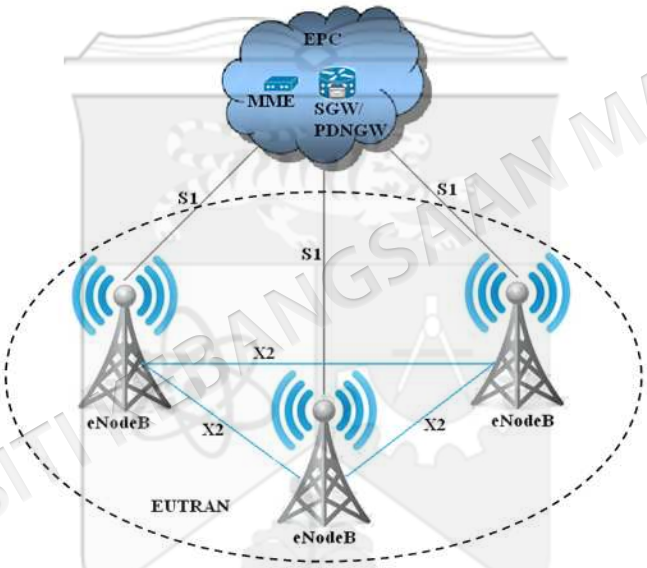


Figure 2.23 E-UTRAN and EPC architecture of LTE-Advanced

The main part in the architecture of E-UTRAN is the enhanced base station Node B (eNB or eNodeB). This provides the air interface between the control plane protocol terminations and the user plane towards the user equipment. The eNodeBs are logical element that can serve one or more E-UTRAN cells. The interfaces of LTE-Advanced network are completely built on IP protocols. The link interfacing between the eNodeBs is termed as X2 interface. The eNodeBs are connected by X2 interface and to the Mobility Management Entity/Gateway (MME/GW) by S1 interface as illustrated in Figure 2.23.

The interface S1 supports many relationships between eNodeBs and MME/GW. The functions splitting between MME/GW and eNodeB are shown in Figure 2.24 (Khan 2009).

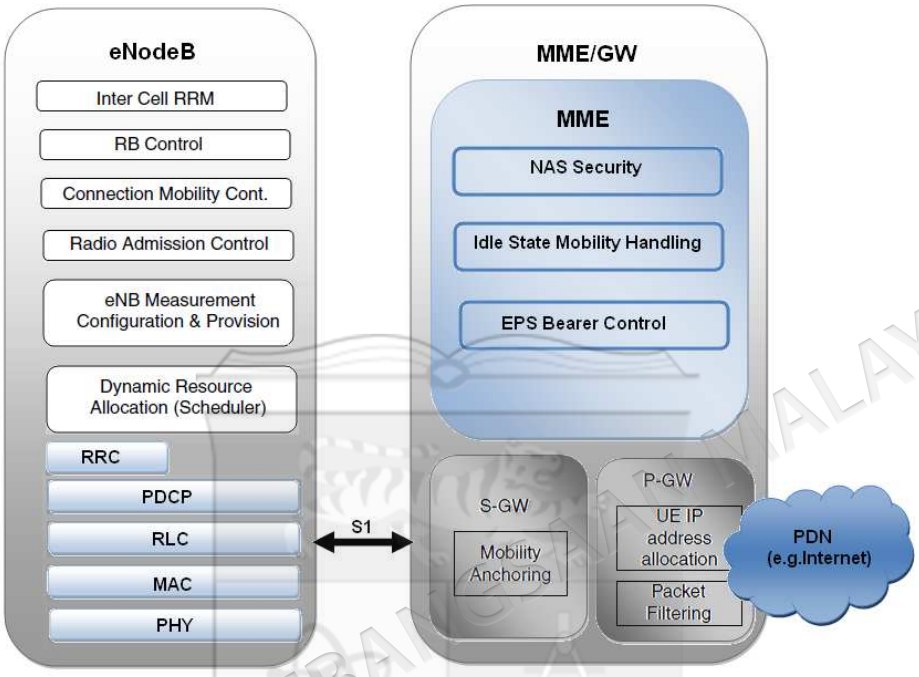


Figure 2.24 Functions splitting between MME/GW and eNodeB

The two entities of the logical gateway are Serving Gateway (S-GW) and Packet Data Network Gateway (P-GW). The Serving Gateway (S-GW) acts as limited anchor for the mobility service, and used to receive and forward packets from and to eNodeB, and this operation is used to serve the UE. The P-GW is an interfacing element used to the linking with exterior Packet Data Networks (PDNs), and this represents Internet Multimedia Server (IMS) or Internet. In addition, P-GW provides other IP functions such as, packet filtering, routing, policy statement, and address allocation. The main benefit of separating network entities is to indicate if the capacity of network for traffic and signalling can independently grow. The main tasks of MME are to idle the reachability of UE and to control the retransmission of paging, roaming, authorization, P-GW/S-GW selection, tracking area list management, bearer management and bearer establishment, authentication, security negotiations and signalling of Non-Access Stratum (NAS) (Khan 2009).

The eNodeBs implement the functions of eNodeB along with the protocols that are usually applied in Radio Network Controller (RNC).

The eNodeB functions include ciphering, packet reliable delivery, and header compression, while in controlling side, the functions of eNodeB are (3GPP 2009):

1. Radio resource management (radio bearer control, radio admission, connection mobility control, and dynamic scheduling).

2. Routing user plane data towards SAE Gateway.

The stack of user plane protocol is shown in Figure 2.25. The Radio Link Control (RLC) and the Packet Data Convergence Protocol (PDCP) layers are usually included in RNC on the network side, but now they are included in eNodeB side (3GPP 2009).

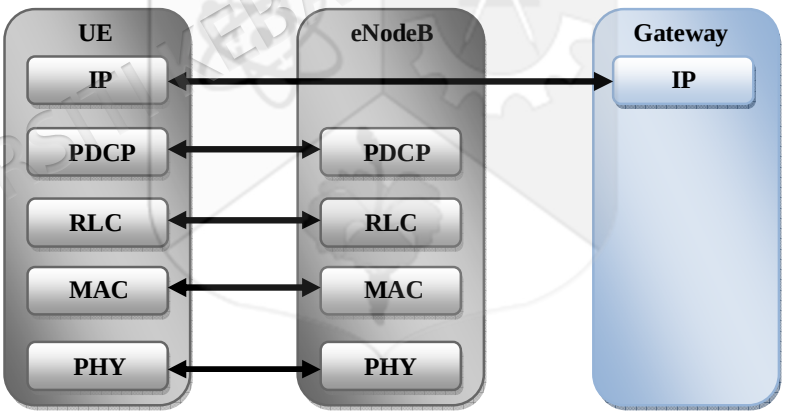


Figure 2.25 User plane protocol

The control plane protocol stack is illustrated in Figure 2.26. The functionality of Radio Resource Control (RRC) is conventionally applied in RNC and is already integrated in eNodeB. The layers of Medium Access Control (MAC) and RLC implement the similar roles to user plane (3GPP 2009).

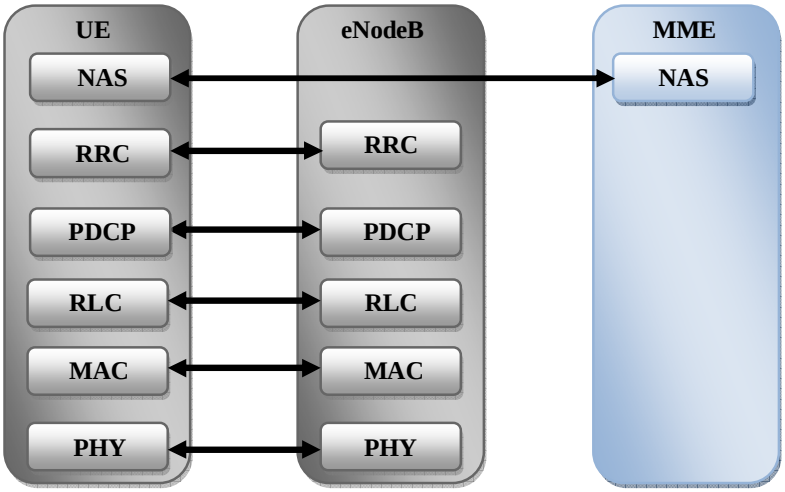


Figure 2.26 Control plane protocol architecture

The functions of RRC include paging, system information broadcast, radio bearer control, and connection management for RRC, measurement reporting to UE, and mobility functions. In the MME network side, the NAS protocol is terminated, while in the terminal side, the UE executes functions similar to EPS such as, authentication, security control, and bearer management. Figures 2.27 and Figure 2.28, illustrate the interface protocol stacks S1 and X2.

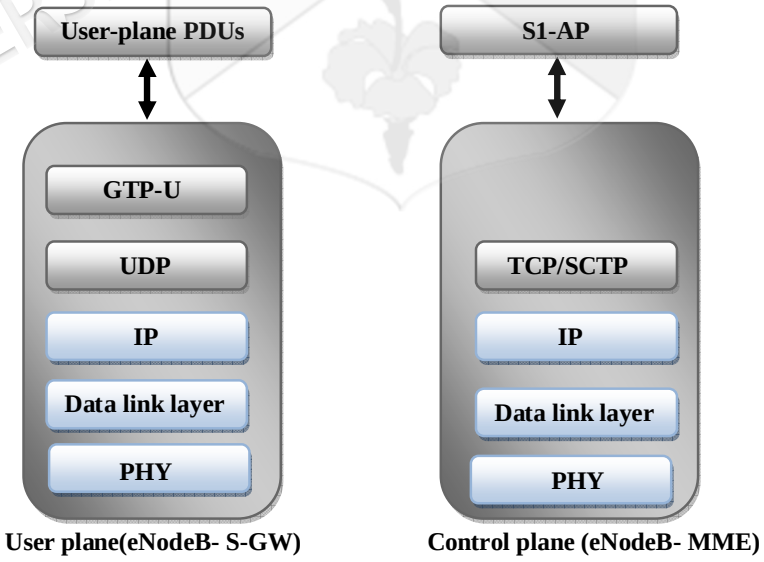


Figure 2.27 S1 interface user and control planes

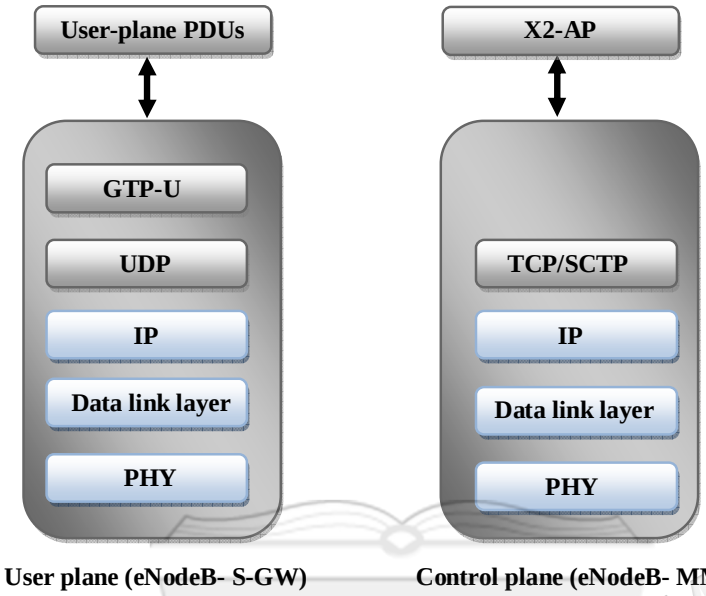


Figure 2.28 X2 interface user and control planes

The interface between S-GW and eNodeB are interconnected by S1 user plane interface (S1-U).This interface uses General Packet Radio Service (GPRS) and Tunnelling Protocol-User Data Tunnelling (GTP-U) over User Datagram Protocol (UDP) UDP/IP transport. In addition, it provides a nonguaranteed delivery to the user plane PDUs between S-GW and eNodeB (Khan 2009). The GTP-U is a simple IP based tunnelling protocol that allows many tunnels between ends sets.

Furthermore, S1 interface separates the EPC and the E-UTRAN into two interfaces. The first is S1-U, which transfers data between S-GW and the eNodeBs. The second interface is S1-MME, which signals the interface between the MME and eNodeB. On other hand, X2 is the interface among the eNodeBs and has two interfaces. The X2-C uses the control plane to interface among eNodeBs, while X2-U is used to interface the user plane interface among eNodeBs. It is supposed that always there is an X2 interface between eNodeBs to provide the required communication between each other (Ghosh et al. 2010).

S1-MME represents the S1 control plane interfacing between MME and eNodeB. Similarly, the transport network layer and user plane are based on IP transport and the probable reliable transport; the TCP or SCTP is applied over the top of IP. The signalling protocol in application layer is mentioned as X2 application protocol (X2-AP) and S1 application protocol (S1-AP) for X2 and S1 interfaces control planes. In S1 interface (and the same applies to the X2 interface), SCTP or TCP is used over the usual IP network layer. There is only one association for S1 interface (or eNodeB to MME relation) and over this association, one SCTP (or TCP) stream is used for all common procedures, such as, the paging between two equipment's and the procedures are supported over limited number of SCTP streams (Lescuyer & Lucidarme 2008).

2.7.2 LTE-Advanced/SAE Reference Architecture

The proposed model of LTE-Advanced is based on the structure of LTE/SAE, which probably might face some challenges in the wireless broadband. The LTE/SAE presents an advanced radio interfacing with main upcoming improvement by using of Orthogonal Frequency Division Multiplexing (OFDM) with compound antenna technique (Qiu et al. 2009). These technologies were previously proposed in WiMAX as itemized in IEEE 802.16. Besides the advanced radio interfacing, the LTE/SAE states the development in the network architecture. The design is packet based and contains less network components to decrease the deployment costs, protocol processing, and the latency in the network.

As illustrated before, the X2 interface between eNodeBs carries the traffic of the user plane (X2-U) and control plane (X2-C). The core network contains control plane elements MME, with S1 control plane (S1-C) traffic and user plane gateways S-GW, with S1 user plane (S1-U) traffic. The X2 interfaces directly among eNodeBs. The LTE-Advanced specifications include strict latency requirements, to apply the features. The cooperative Multiple Input Multiple Output (MIMO) is an example of these features requirements.

All these requirements are required to consider in the range of 10 ms. In LTE-Advanced systems, different base stations are connected together using the X2 logical interface. Furthermore, there is no continuous guarantee to achieve straight link among cooperating entities (Stencel et al. 2010). Therefore, the structure of the network has to be adjusted by the operator, to effectively apply the schemes of Coordinated Multipoint (CoMP) in practice. The CoMP is a technique built on the cooperation among different base stations via fast backhaul network and used in data reception and transmission. This technique meaningfully assists to enhance the interference conditions and consequently the whole system performance.

2.7.3 LTE-Advanced Relaying

LTE-Advanced spreads the coverage approaches into in-band and out-band relaying. LTE-Advanced adopts a relay technology to achieve self-backhauling of the radio signal between the mobile station and the base station. This technology targets to enhance the received signal, improve the cell interface, and to enhance the throughput. In this manner, the radio signal can be broadcasted more competently and the cell coverage is extended at the edge of the cell (Iwamura et al. 2009). The Relay Node (RN) is linked by Uu interface and connected to a contributor cell (eNodeB) through the Un interface as shown in Figure 2.29.

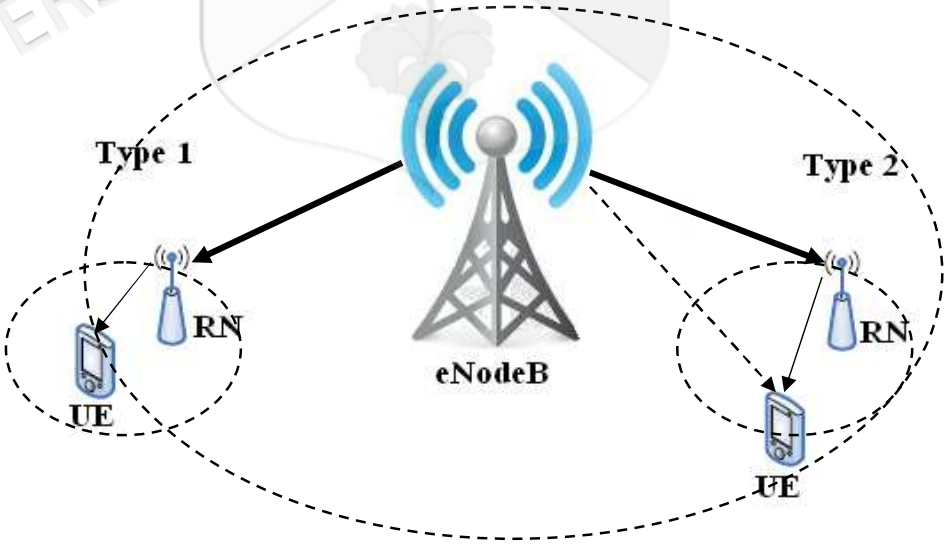


Figure 2.29 Relaying architecture and types

RN separates the cell with individual physical cell ID and provides synchronization to the channels. Which means that the wholly Uu control plane and the user plane protocols are established in the RN (Stencel et al. 2010). Two types of relaying have been defined in 3GPP LTE-Advanced standards, Type 1 and Type 2, or can be classified as transparency and non-transparency (Yang et al. 2009). Type 1 is non-transparency relaying and can assist the remote UE locates far-off from an eNodeB to contact the eNodeB. Thus, Type 1 requires to send general reference signal to adjust the information for eNodeB.

Hence, the main objective is to provide coverage service and that can help to contribute to the capacity of the system by enabling extra communication services and data transmission for the isolated UEs (Yang et al. 2009). On the other type (Type 2), the transparency relaying helps the local UEs located within the coverage of an eNodeB and have a straight connection with the eNodeB, to enhance the link capacity and the service quality. Therefore, Type 2 cannot send a common reference signal or information control, but it assists to expand the capacity of the network by realizing the multipath range for the local UEs.

2.7.4 TCP versus SCTP

Both LTE and LTE-Advanced use either TCP or SCTP, to send messages between two peers. Since both protocols are connection-oriented, the connection between two peers has to be established before sending any command (Olsson et al. 2009). The SCTP is a transport protocol identified in RFC 2960 (Stewart et al. 2004), which works at an equivalent level in the stack as TCP and UDP. Compared to TCP and UDP, the SCTP is superior in functionality and more robust against the failures in the network connections. The purpose in employing SCTP is to provide an efficient and reliable signalling bearer. To achieve this, the SCTP provides appropriate congestion control techniques with fast retransmission in the case of packet loss and enhanced reliability. Furthermore, it provides extra security against blind attacks and enhances security feature, when connected on top of UMTS and other 3G systems with different operators (Kaaranen 2005).

When the functionalities of TCP and SCTP are compared, it is evident that the later provides two key features, multi-streaming and multi-homing, which lacks in TCP. In the SCTP domain, a stream is a unidirectional sequence of user packets to be distributed to upper layers. Consequently, bi-directional communication between two entities includes at least a pair of streams, one in each direction. The multi-streaming is the feature, from which the name of STCP is actually derived. It permits setting up several independent streams between two peers. In such a case, when a transmission error happens on one stream, it does not affect the transmission on the other streams.

In contrast, TCP only provides one stream for a given connection between IP peers, which may cause additional data transmission delay when a packets dropped. When a transmission loss happens on a TCP connection, the packet delivery is suspended until the missing parts are restored (Lescuyer & Lucidarme 2008). SCTP provides new services and features for IP communication. However, the TCP provides reliable communication service and the UDP provides unreliable service but neither TCP nor UDP can handle multi-homing or have the ability to send information to an alternate address, if the primary becomes unreachable. Many of the features found in TCP and UDP can be found in SCTP. Ccomparative results between SCTP, TCP and UDP are provided in Table 2.3 (Stewart et al. 2008).

Table 2.3 Comparison among TCP, UDP, and SCTP

Feature	**TCP**	**UDP**	**SCTP**
Connection oriented	Y	N	Y
Reliable transport	Y	N	Y
Preserve message boundary	N	Y	Y
In-order delivery	Y	N	Y
Un-order deliver	N	Y	Y
Data checksum	Y(16-bit)	Y(16-bit)	Y(32-bit)
Flow and congestion control	Y	N	Y
Multiple streams within a session	N	N	Y
Multi-homing support	N	N	Y

Y-Yes, N-No

Based on the Table 2.3, it can understand that, the UDP is quickly ruled out as unfit as it is not reliable. From a high-level perspective, the SCTP and TCP are quite close to each other, as they both, support reliable and ordered data delivery, and they have congestion control, to regulate network data flow. Nevertheless, the SCTP is message-oriented and frames individual messages as against the TCP. In SCTP, the messages are transmitted as a whole set of bytes (provided the maximum length is not reached), which helps to improve the transmission efficiency.

The SCTP was first developed to professionally transfer the telephony signalling data across the Internet, but its features make it more attractive for other applications too (Welzl 2005). The SCTP is well known for its advanced features inherited from TCP that ensure the required reliable delivery of the signalling messages. Furthermore, the enhanced features such as, the handling of multi-streams to easily implement transport network redundancy add more value to the SCTP. The 3GPP working groups debate on the suitable transport protocol for the controlling plane in LTE/LTE-Advanced, to support the signalling message exchange between network entities.

The SCTP can handle congestion and packet drop, better than the standard protocols. In fact, the SCTP implements congestion and flow control such as, TCP. In addition, the SCTP association is distributed into a number of streams, where the SCTP associations are similar to the TCP connection, except that, it is able to provide multiple IP addresses for both ends. The SCTP is similar to TCP in using message acknowledgement and retransmission mechanism, to ensure the segments delivery to the corresponding hosts (Bates 2002). Moreover, as in TCP, the SCTP has three main phases in congestion control such as: slow-start, congestion avoidance, and fast retransmit.

As mentioned above the SCTP uses multi-streaming, therefore if the transmission to the primary path is carried out in the congestion avoidance mode, the implementation may still use slow-start for the other paths (Ali 2010).

The congestion control for TCP and SCTP shares many features in slow-start and congestion avoidance phases. Therefore, when using either TCP or SCTP, the congestion control is affected by the parameters and the limitations of the LTE/LTE-Advanced network. The significant objective for LTE and LTE-Advanced is to decrease the network latency (Dahlman 2008). The network latency is the time taken by a packet to travel from a client to a server and back again and that represents a direct limiter on performance of TCP. The congestion control mechanism, especially the slow-start, is sensitive to the link latency in the network and it is expected to delay the data transmissions.

Therefore, the protocol used should guarantee to prevent packet from going beyond the network capacity, even when the number of transferred packets is huge. The congestion control of SCTP follows the same congestion window reduction mechanism used by TCP. In addition, the SCTP might behave similar to TCP in the event of multiple packet losses (Nagamalai & Lee 2005). The SCTP applies window-based congestion control similar to TCP, and uses AIMD technique to control the window size according to the network conditions. Therefore, even though the congestion control in this study is proposed to execute over TCP protocol, it is also possible to apply the enhanced congestion control over SCTP, especially the proposed congestion avoidance is grounded on AIMD, which will be explained in Chapter IV.

2.8 SUMMARY

This chapter had presented the fundamentals and analysed the functional structure for TCP protocol. Besides, this chapter had focused on the congestion control mechanism used by standard TCP. It had reviewed the factors and the parameters that affect the behaviour and the TCP performance, and had illustrated the design of the congestion control of six TCP source variants in addition to some other TCP's. The techniques used to improve the congestion control were also presented. The comparison of congestion control phases and ability for six TCP variants are briefed in Table 2.4.

Table 2.4 Comparison of TCP source variants

Feature	Tahoe	Reno	Newreno	Sack	Fack	Vegas
Slow-start	NE	NE	NE	NE	NE	EV
Congestion avoidance	Y	Y	Y	Y	Y	EV
Fast Recovery	N	Y	EV	EV	EV	Y
Fast retransmit	Y	Y	Y	Y	Y	Y
Multiple packet flow	N	N	N	N	N	N
Recover Multiple loss	Y	N	Y	N	N	N
Retransmission	N	N	N	N	N	NM
Congestion control	N	N	N	N	NM	NM
Selective ACK	N	N	N	Y	Y	N

Y-Yes, N-No, NE-Not Efficient, NM-New Mechanism, EV-Enhanced Version

Moreover, the possibility to improve the performance of TCP over wireless and 4G systems were reviewed too. Based on previous researches, the employment of TCP and SCTP protocols over LTE-Advanced network was discussed with the explanations to the downside and the limitation in using TCP protocol over large-bandwidth low-latency networks. The architecture of LTE-Advanced network studied and illustrated in side of traffics, links, components, and the distribution of network elements. Moreover, the role of each component clarified with the effect on the whole performance. Additionally, the comparisons between TCP and SCTP protocols were presented in this chapter to investigate the distinctive employment of TCP and SCTP over LTE/LTE-Advanced networks with the key features of SCTP protocol.

CHAPTER III

ENHANCED TCP SLOW-START MECHANISMS

3.1 INTRODUCTION

With the rapid growth towards the next generation networks utilising TCP protocol, the TCP congestion control of TCP has become the key factor in determining the performance of the networks. The congestion happens when the resources demanded by the network are greater than the resources that can be supplied, which consequently leads to dropping of several packets. These dropped packets will then be retransmitted (Hassan 2005). When several hosts send the packets through a common link in the network, and if these packets are sent by hosts out of control, the link will be unable to handle these packets and the network nodes will reject several packets. To avoid packets dropping, the hosts need to slow down the packet transmission if the acknowledgments are missing. If any of those acknowledgments is missing, the host retransmits the packet that has failed to arrive by supposing that the loss in congestion may occur but will slow down the flow rate of the packet. If the host gets the required acknowledgments in a timely routine, it gradually surges the transmission of packets to reach a maximum rate available in the network. This arrangement is termed congestion control.

The congestion control allows TCP to repeatedly regulate the packets transmission rate and the links of network are completely exploited although there is limitation of the congestion (Walrand & Parekh 2010). The main function of the congestion control is to reduce packets flow in the network traffics and to relieve the congestion at a certain point in the network.

In addition, the congestion control redirects the flows to other paths that are far from the nodes that suffer from congestion (Puzmanova 2001). Therefore, TCP congestion control plays an important role for the applications requesting and also it can limit the services provided to the users over various networks.

3.2 STANDARD SLOW-START MECHANISM

The implementation of TCP congestion control mechanism is not a straight-forward operation. If the TCP senders are ready for packets transmission, the mechanism initiates the increment of congestion window in the attempt to increase the rate of transmitted packets. The window size exponentially rises until threshold level called slow-start threshold (*ssthresh*) and this phase is termed as "slow-start" phase as shown in Figure 3.1.

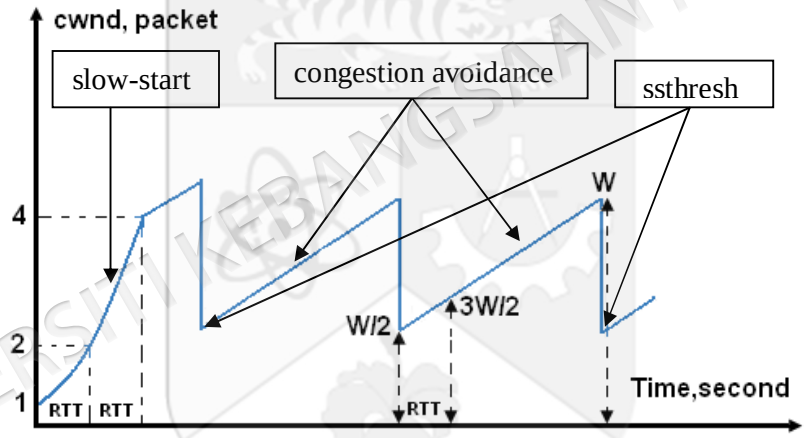


Figure 3.1 Congestion window in slow-start and congestion avoidance

In slow-start phase, the TCP sender doubles the size congestion window to increase the transmitted rate for each RTT. When *cwnd* size reaches *ssthresh*, TCP moves to the next phase called "congestion avoidance" and then window size moderately increases in slower rate. TCP sender immediately detects packet loss by observing the full number of the duplicated acknowledgements from the receiver. As explained in Section 2.3.6, if the TCP sender obtains a duplicate acknowledgement from previous sent packet, it can recognise the packet loss and it automatically decreases the size of congestion window. Then, the packet rate will also decrease and it sets a new level to *ssthresh* on the new congestion window.

These phases are called as fast retransmit and fast recovery, respectively, where the TCP sender arrives to these phases by passing the slow-start phase and congestion avoidance phase.

If the congestion situations are simple (easy to detect congestion) or not recognised by the sender, TCP congestion control will be unable to detect the packet loss. Hence, TCP depends on other technique called "retransmission timeout" that produces retransmission for the lost packets. Moreover, TCP decreases *cwnd* to one segment and tries to retransmit the lost packet. On the other hand, the TCP doubles the interval of retransmission timeout when the sender keeps transmitting the same packets and fails to recognise any acknowledgement for its sent packet.

TCP sender starts by setting the congestion window to one segment (*cwnd = 1*). The size of window is exponentially enlarged in every RTT and continues until the window reaches *ssthresh*. Once it reaches *ssthresh*, TCP reduces the increase of *cwnd*. Then, after time interval, if the transmission rate of TCP exceeds the capacity of the network link, packet loss occurs. When TCP detects the packet loss, the size of congestion window decreases to half of its value. If the congestion conditions are cleared, TCP sender enters the phase of congestion avoidance. Then, congestion window is linearly improved by one segment in every RTT.

As illustrated in Figure 3.2, in steady state, TCP oscillates between the values of window *W* and *W/2*, where "*W*" depends on the capacity of the network link and the number of active TCP connections (Vallamsundar et al. 2007). TCP slow-start algorithm applies two stages over the congestion window, the first with the initial transmission, while the other is in the middle of transmission interval. These two stages are due to the congestion control in some protocols such as Tahoe that continuously uses slow-start after each timeout, while in Reno and Newreno; it is used only during early starting point.

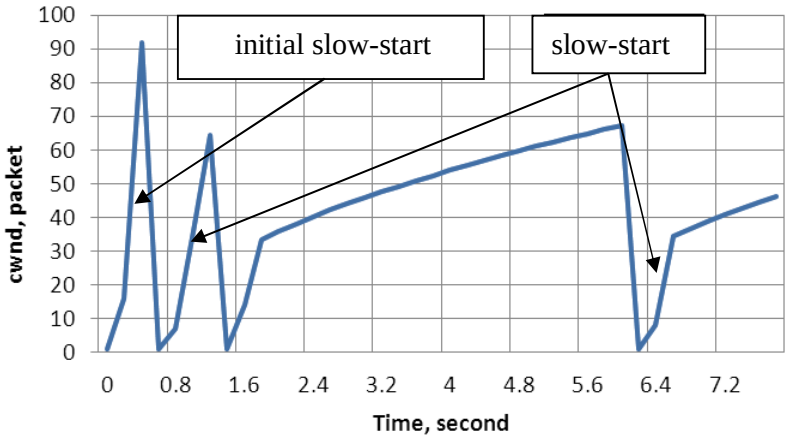


Figure 3.2 Difference between initial slow-start and slow-start for TCP Tahoe

The reasons behind this difference are due to the fast recovery and fast retransmit algorithm used over congestion control in Reno, which are not available in Tahoe. Hence, although both Tahoe and Reno start with the same initial slow-start mechanism, in the next RTTs, only Tahoe uses slow-start while Reno compensates this situation by fast retransmission. However, after connection has been established, the congestion window value is still unknown and TCP congestion control uses slow-start to distinguish the congestion window value. . In addition, the congestion control will limit the number of the unacknowledged packets and inject the packets into the pipeline of network. On the other hand, the bandwidth and the delay of the network links will indicate the perfect size of the congestion window.

Initially, the congestion window size is set to be one segment as in Equation (3.1). If TCP sender receives the ACK, the window size increases by one segment again.

$$cwnd\ (t) = 1 \tag{3.1}$$

The congestion window in standard slow-start increases in each ACK received and with new RTT. Consequently, for each new RTT, the current size of congestion window is doubled until it reaches the maximum size that represents the exponential growth of the congestion window as in Equation (3.2).

$$cwnd(t+\tau) = \begin{cases} 2 * cwnd\ (t) & \text{if } cwnd(t) < ssthresh/2 \\ Congestion\ avoidance & \text{elsewhere} \end{cases} \quad (3.2)$$

where *cwnd(t)* represents the current size of the congestion window, while *cwnd(t+ τ)* denotes the next *cwnd* size after one RTT. This exponential increment can end if one of two conditions happens. If the congestion window reaches a determined *ssthresh*, then TCP moves to the congestion avoidance mode. Otherwise, the TCP experiences loss in packet, which frequently occurs when timeout happens, and then TCP will set the new value of *ssthresh* to be *cwnd/2* and move to the congestion avoidance mode.

The other increment in congestion window happens when each successful ACK is received by the sender as shown in Equation (3.3).

$$cwnd(t+\tau) = \begin{cases} cwnd\ (t) + 1 & \text{if } cwnd(t) < ssthresh \\ Congestion\ avoidance & \text{elsewhere} \end{cases} \quad (3.3)$$

where *cwnd(t)* represents the current size of the congestion window, while *cwnd(t+ τ)* denotes to the next *cwnd* size after ACK. Figure 3.3 explains the initial start-up of standard TCP slow-start with maximum window size of 64 packets.

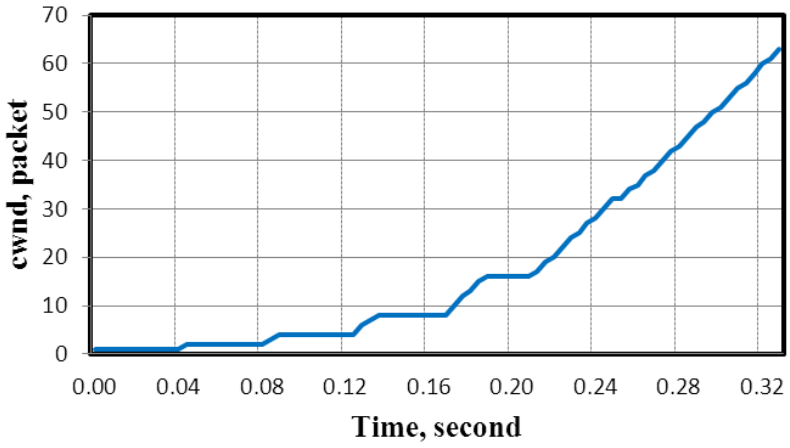


Figure 3.3 Initial slow-start phase of standard TCP

3.3 SLOW-START MECHANISM WITH LINEAR INTERPOLATION INCREMENTS

Following to the standard slow-start algorithm in traditional TCP congestion control, this section explains a new slow-start mechanism to provide smooth and fast increment to the slow-start of congestion window that can reach *ssthresh* during shorter time. The new proposed algorithm based on congestion window size increments (normally start with initial window size of one segment) by duplicating the window size instead of increasing it by one segment for each ACK received.

When connection is established over the network, the congestion control triggers the slow-start phase to start sending packet from TCP source to destination. Then, *cwnd* size starts from segment 1 and grows to 2, 4, 8, and so on, until it reaches *ssthresh/2* for every new RTT by sending the double value of the last size of *cwnd*. This increment style allows TCP sender to send a large number of packets. Besides, the linear increment of *cwnd* will occur in very short period due to the increment in this scheme depends on *cwnd* size duplication instead of increasing it by one segment for each ACK received. The standard formula is illustrated in Equation (3.3) where *cwnd* linearly increases, while the new formula shown in Equation (3.4) is proposed to increase *cwnd* by multiplication factor of two to achieve faster growth.

Every new ACK:

$$2 * cwnd\ (t) \rightarrow cwnd\ (t + \tau) \tag{3.4}$$

This procedure repeats for each ACK received and forces *cwnd* to reach the maximum network limit in reduced period as shown in Figure 3.4.

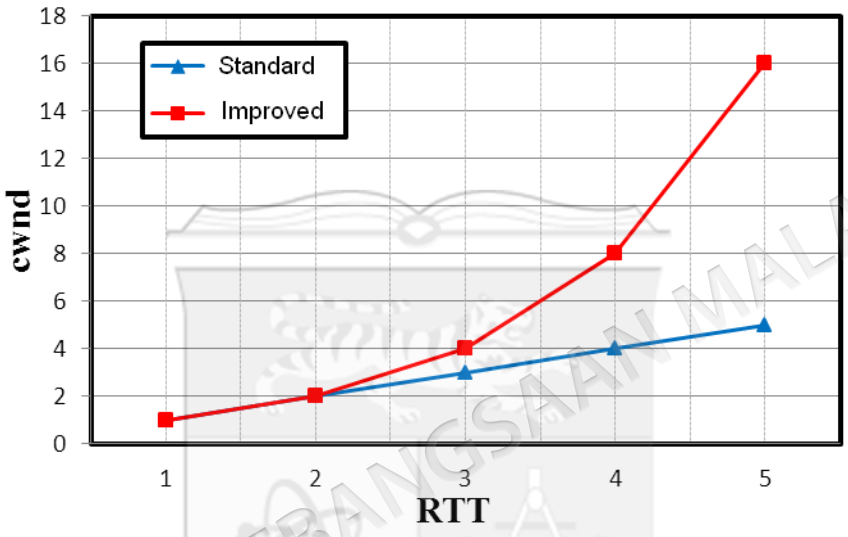


Figure 3.4 Comparison between linear and doubling increasing

The example shown in Figure 3.4 assumes five RTTs (these RTTs mean one acknowledging cycle without packet loss). It is noted that in standard growing of *cwnd*, it reaches size equal to 6 packets after five RTTs, while in doubling method, it reaches to *cwnd* size equal to 8 packets after three RTTs only and this difference will continuously increase for large window size. Numerically, during 5 RTTs, the standard *cwnd* increases from one segment to 5 segments size only, while the proposed technique is supposed to increase *cwnd* from one segment to 2, 4, 8, and then to 16 segments during same period (5 RTTs).

In standard slow-start mechanism, the congestion control algorithm can check if *cwnd* size is equal or larger than *ssthresh* to start the congestion avoidance. On the other side, in proposed method (doubling increment), the congestion control algorithm needs to control the increment step size before it exceeds *ssthresh*. The practical problem happens when the duplicated value of *cwnd* is larger than the *ssthresh*.

In other words, the new technique should stop duplicating *cwnd* when the last value of *cwnd* is larger than *ssthresh/2* and less than *ssthresh*. In this situation, it is not suitable to duplicate *cwnd* due to this situation make *cwnd* exceeds *ssthresh* of the network. For this reason, the new mechanism is supported by other algorithm to continue increasing the window but in small and predicted steps when *2*cwnd > ssthresh/2*. This additional technique is based on the linear interpolation formula to join the last value of *cwnd* with the window limit (*ssthresh*) to reach *ssthresh* smoothly. To explain the function of this controller on the behaviour of the slow-start increment, the mathematical representation of the linear interpolation is illustrated in next section.

3.3.1 Linear Interpolation Algorithm

Interpolation technique is used to find new points of data in a range of identified data points. The linear interpolation is the simple routine to obtain values at a position in between other points of data. These points are simply combined within straight line of segments, where a line that relates two data points is used to predict the halfway values. Figure 3.5 explains the assumption of simple interpolation routine (Meijering 2002).

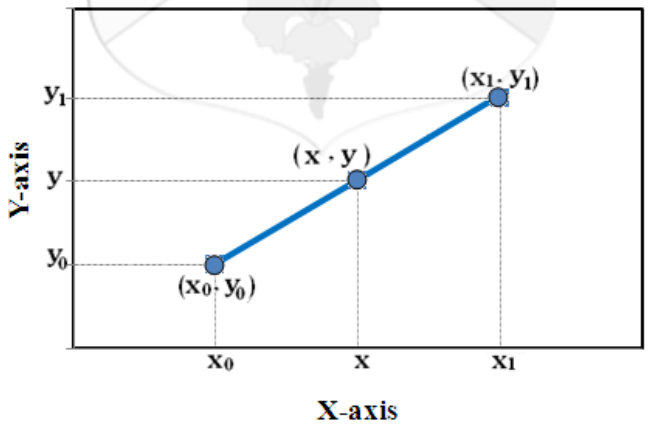


Figure 3.5 Linear interpolation

The procedure in Figure 3.5 consists of the coordinates (x_0,y_0) and (x_1,y_1) with the expectation to obtain the points on this line by knowing either an x or y in the interval $(x_0 \ldots x_1, y_0 \ldots y_1)$. For example, it is possible to increase x by a constant factor from x_0 up to x_1 and get the corresponding y values. The expected relationship obtained from the Figure 3.5 can be summarised as follows:

$$\frac{y - y_0}{x - x_0} = \frac{y_1 - y_0}{x_1 - x_0} \tag{3.5}$$

By solving this equation for y, then can obtain:

$$y = y_0 + (x - x_0)\frac{y_1 - y_0}{x_1 - x_0} \tag{3.6}$$

Equation (3.6) can be rewritten as follows:

$$y = y_0 + y_1\frac{(x - x_0)}{(x_1 - x_0)} - y_0\frac{(x - x_0)}{(x_1 - x_0)} \tag{3.7a}$$

$$y = y_0\left[1 - \frac{(x - x_0)}{(x_1 - x_0)}\right] + y_1\left[\frac{(x - x_0)}{(x_1 - x_0)}\right] \tag{3.7b}$$

This is the formula for linear interpolation in the interval (x_0,x_1) and when it is outside this interval, the formula is identical to linear extrapolation. This formula can also be understood as a weighted average. The weights are inversely related to the distance from the ends to the unknown point and the closer point has more influence than the farther point. Thus, the weights are:

$$\frac{x - x_0}{x_1 - x_0} \quad \text{and} \quad \frac{x_1 - x}{x_1 - x_0}$$

where these are normalised distances between the unknown point and the end points.

3.3.2 Formulation of Interpolated Slow-Start Algorithm

Now, the interpolation is used over the relation between *cwnd* and *ssthresh* to get a straight relation between them by assuming variables and constant parameters. As shown in Figure 3.6, the interpolation can be implemented in terms of *ssthresh*, *cwnd*(t), and *cwnd*(t+τ) and the time differences (ACK period) required to estimate the coefficients are taken into account.

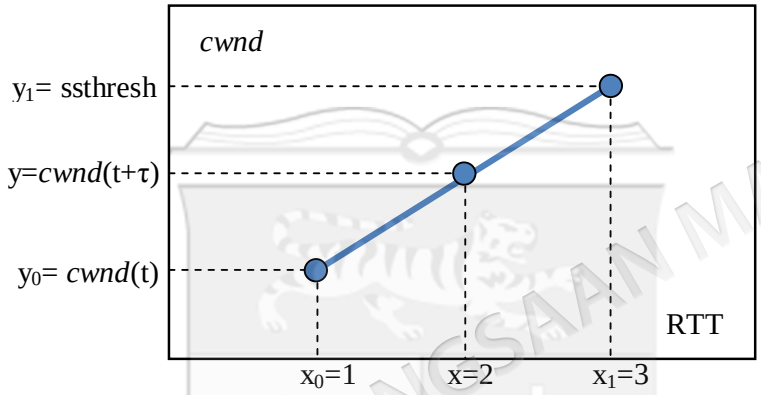


Figure 3.6 Linear interpolation in term of *cwnd* and *ssthresh*

From Figure 3.6, it can obtain the following:

$$cwnd\ (t+\tau) \rightarrow y$$
$$cwnd\ (t) \rightarrow y_0$$
$$ssthresh\ \rightarrow y_1$$

In addition, because *cwnd* always increases for every one ACK received, then:

$$x_1 - x_0 = 2$$
$$x - x_0 = 1$$

After substitution, the new variables and parameters are as shown in Equation (3.7b), then the equation can be expressed as follows:

$$cwnd\ (t+\tau) = cwnd\ (t)\left[1-\frac{1}{2}\right] + ssthresh\ \left[\frac{1}{2}\right]$$

Then:

$$cwnd\ \ (t+\tau) = \frac{1}{2} cwnd\ \ (t) + ssthresh\ \ \frac{1}{2}$$

After simplification:

$$cwnd\ \ (t+\tau) = \frac{cwnd\ \ (t) + ssthresh}{2} \tag{3.8}$$

Equation (3.8) represents the linear interpolation formula to increase *cwnd* size when the congestion control algorithm cannot double the *cwnd* anymore. Therefore, the new slow-start mechanism for each ACK is as shown in Equation (3.9):

$$cwnd(t+\tau) = \begin{cases} 2 * cwnd\ \ (t) & \text{if } cwnd(\text{t}) \leq \text{ssthresh/2} \\ \dfrac{cwnd(t) + ssthresh}{2} & \text{if (ssthresh/2} > cwnd(\text{t}) \ \&\& \\ & \quad cwnd(\text{t}) < \text{ssthresh)} \\ Congestion\ avoidance & \text{elsewhere} \end{cases} \tag{3.9}$$

The enhanced slow-start algorithm is separated into two phases as shown in Equation (3.9). The first phase occurs when *cwnd* is less than *ssthresh/2*; and when it exceeds this condition, the algorithm enters to the second phase to increase *cwnd* size in slow mode by using interpolation method.

The main goal to improve the slow-start algorithm is to propose a robust adaptive congestion control mechanism for TCP. The concept of the new slow-start algorithm is derived from the linear interpolation to approximate *cwnd* in slow-start after duplication happens and when *cwnd* is less than *ssthresh/2*.

The pseudo code of the proposed slow-start mechanism is implemented on tcp.cc file (Appendix A) in the source of NS-2 simulator by changing the standard subroutine by proposing an algorithm as follows:

```
   /* Slow-Start phase*/
if (cwnd_ <= ssthresh_/2){
       /* Duplicating cwnd */
     cwnd_ = 2*cwnd_ ;
        }
  if (cwnd_ > (ssthresh_/2) && cwnd_ < ssthresh_){
       /* Linear interpolation */
     cwnd_ = (ssthresh_ + cwnd_)/2 ;
        }
```

Observation of the behaviour of the new mechanism is used by setting a maximum window size with 128 packets and by comparing the improved TCP (TCP with new slow-start mechanism) with the standard TCP (TCP Tahoe is used in this comparison) as shown in Figure 3.7.

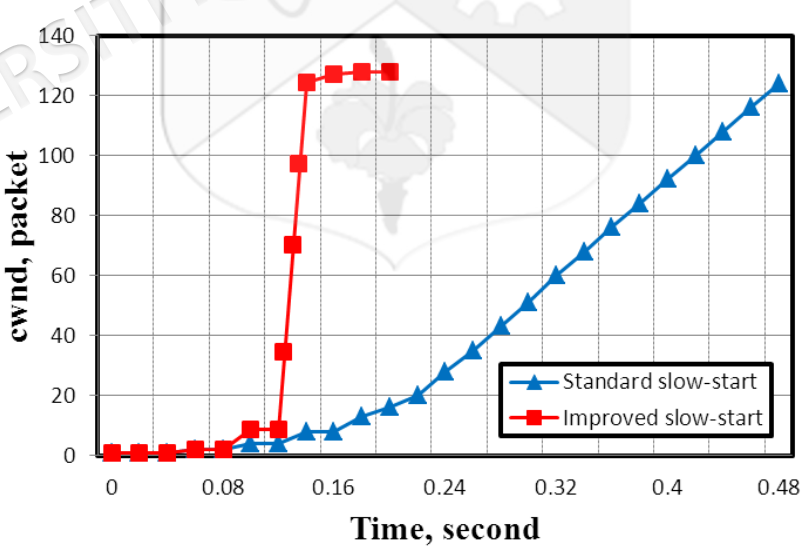


Figure 3.7 Initial slow-starts for standard and improved mechanisms

The slow-start of standard and improved algorithms are presented in Figure 3.7, where the standard method needs 0.487 s to reach the initial *ssthresh* while the improved method only needs 0.192 s to reach the *ssthresh*. The proposed algorithm is flexible and the increment step in second stage can be changed to produce better slow-start algorithms. Hence, the second stage can be written in other form as follows:

if ($cwnd < ssthresh / f$)

then $cwnd = f * cwnd$

end if ;

where f is the scaling factor to adjust the start-up of *cwnd*. Hence, *cwnd* increases by multiplying *cwnd* by f for every ACK received in first growing stage. In second stage, it is important to change the condition of the interpolation area to be between *ssthresh/f* and *ssthresh*. Therefore, this stage should be written in the following form:

if ($cwnd \geq ssthresh / f$ && $cwnd < ssthresh$)

then $cwnd = (cwnd + (f - 1) * ssthresh) / f$

end if ;

This scheme shows that it is necessary to keep the value of f greater than or equal 2 according to the behaviour and the required time of *cwnd* start-up. Table 3.1 illustrates the required period for *cwnd* start-up with different scaling factor values.

Table 3.1 Scaling factor f and start-up time

Scaling factor *f*	**Start-up time (second)**
2	0.192
3	0.195
4	0.224
5	0.250
10	0.299

In order to test the start-up phase of the proposed slow-start algorithm and to monitor the effect of scaling factor *f*, a comparison proceeds when maximum window size is set to 128 packets. Two is the default value of *f* and gives the faster start-up than other *f* values. Moreover, other values provide reasonable start-up period because the difference in start-up time is when *f* is equal to 2 and 3, not exceeding 0.003 second, while the gap is increased when *f* has large values, such as 10. Figure 3.8 shows the effect of scaling factor on the behaviour of initial slow-start phase.

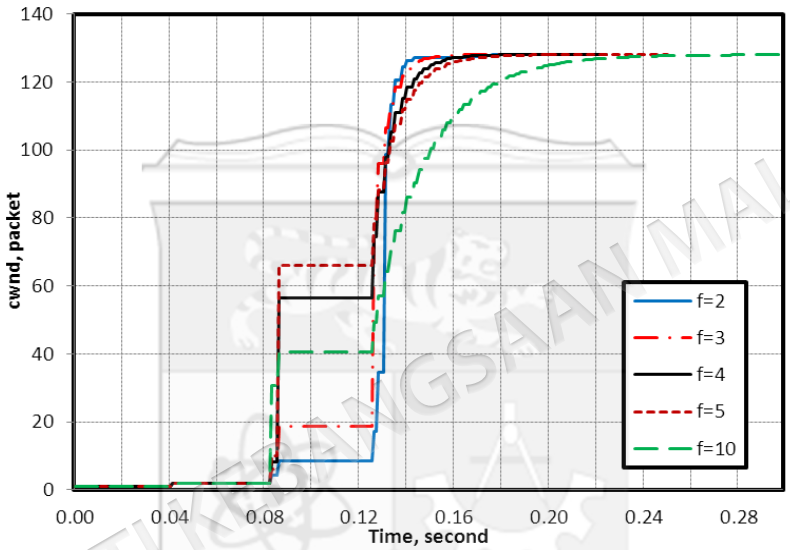


Figure 3.8 TCP start-up in slow-start phase with different scaling factors *f*

All scaling factors show adjacent behaviour. In fact, when *f*=2, the delay in *cwnd* grows but it reaches *ssthresh* faster than others. On other side, when *f*=10, the start-up becomes smoother but it needs longer time to reach *ssthresh*. Therefore, it can be concluded that when *f*=2 represents the finale scaling factor to the proposed slow-start algorithm, the algorithm will keep the same scheme when it is applied later with the proposed congestion avoidance algorithm.

3.3.3 Performance Evaluation of Interpolated Slow-Start Mechanism

To evaluate the performance of the proposed mechanism and to observe the behaviour of the congestion window, a heterogeneous network topology is proposed to experiment the new congestion control scheme.

The proposed network topology is constructed from three TCP sources that are connected to the destination side through bottlenecks using four routers namely R1, R2, R3, and R4 as shown in Figure 3.9.

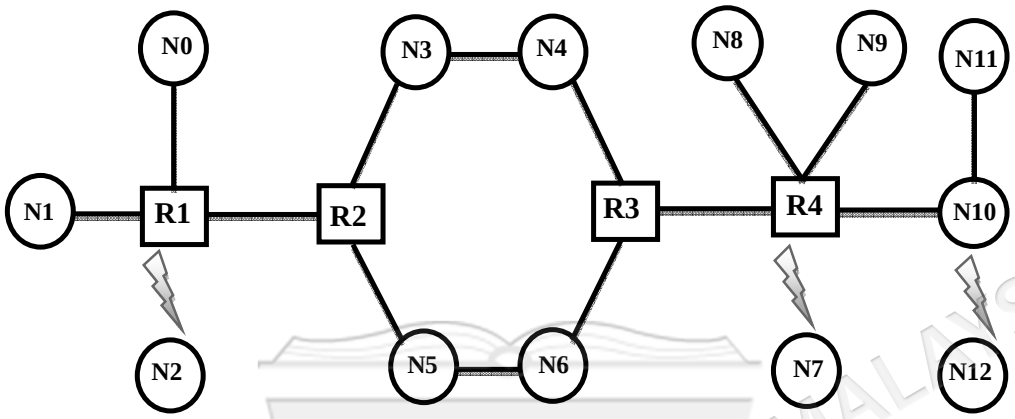


Figure 3.9 Proposed network topology

The proposed topology also contains three wireless nodes namely N2, N7, and N12 with two cascaded bottlenecks and one crosslink to introduce sudden re-routing to the packet flow from source nodes to destination nodes. The experimentation of proposed mechanism includes three situations namely packet losses, three cascaded bottlenecks, and re-routing, and these conditions provide high level of packet congestion to the network topology and to the simulation scenario.

For this experiment, the length sizes of queues are set to 12 packets to monitor the packet flow over the links between R1-R2 and R3-R4. All links connections are Drop-Tail type (Patil et al. 2011) with maximum congestion window of 20 packets and packet size of 1460 bytes. In fact, the proposed topology is separated into two sides: sender and receiver, and the scenario is based on the use of the nodes N0, N1, and N2 as TCP sources while the nodes N11, N10, and N12 as TCP destinations. Other parameters of the proposed topology with links bandwidths and delay are illustrated in Table 3.2.

Table 3.2 Network links parameters

Link	Bandwidth Mbps	Delay ms
N0-R1	10	2
N1-R1	10	2
N2-R1	0.5	2
R1-R2	4	6
R2-N3	8	2
R2-N5	8	2
N3-N4	8	2
N5-N6	8	2
R3-N4	8	2
R3-N6	8	2
R3-R4	4	2
R4-N7	0.5	2
R4-N8	10	2
R4-N9	10	2
R4-N10	10	2
N10-N11	10	2
N10-N12	0.5	2

TCP Tahoe is used to test the new slow-start technique since it does not support fast retransmission approach while TCP Reno and Newreno have fast recovery and fast retransmission. This approach will assist to watch the slow-start phase frequently during the simulation session and the effect of the proposed technique clearly appears because Tahoe uses slow-start with each transmission cycle. Figure 3.10 demonstrates the behaviour of congestion window before and after using the interpolated slow-start scheme but without congestion events. The differences in bandwidth of the proposed links are to provide variety congestion situations and to make fast rerouting and bottleneck links to the data transferred for source nodes to the destination nodes. Moreover, all links used small latency except the link R1-R2 due to the queue measurement requires larger delay for that it set to be 6 s instead of 2 s.

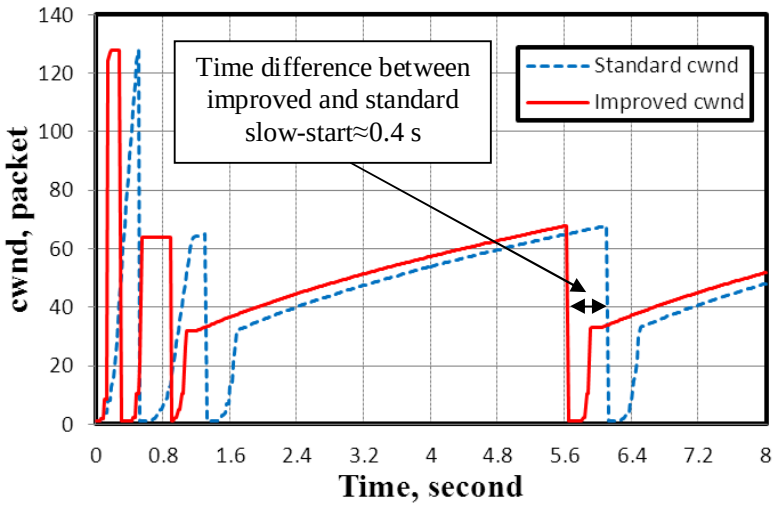


Figure 3.10 Congestion window comparisons for standard and improved Tahoe without congestion

Figure 3.10 shows the fast initialisation of *cwnd* when interpolated slow-start mechanism is used. In addition, *cwnd* only requires 5.7 s to finish the completed window cycle while the standard congestion control requires 6.1 s to complete the same cycle. In addition, the gap in time between standard and proposed mechanisms continuously increases because the collected time variation is obtained after each slow-start phase.

This fast increase of the congestion window assists to reduce the congestion over the network pipelines or at least to delay the congestion event that is expected to occur when the network has large bandwidth and low latency. As a result, this will reduce the packet loss, avoid reasonable amount of packets retransmissions, and help to avoid the TCP buffer of the destination to enter overflow situation.

a. Congestion window comparison

The behaviour of congestion window shown in Figure 3.10 represents an ideal congestion window because no congestion, no re-routing, and no packet loss are assumed. Figure 3.11 shows the congestion window behaviour when practical network events are injected to monitor the performance and the behaviour of the proposed slow-start algorithm, and compared with the standard slow-start.

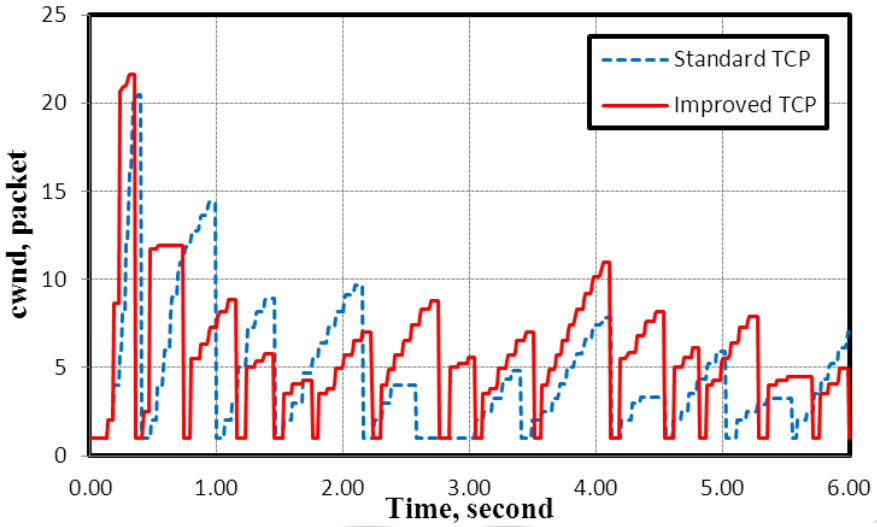


Figure 3.11 Congestion window comparisons between standard and improved Tahoe over congested network

TCP used in N0, N1, and N2 implements the slow-start algorithm and applies the standard congestion avoidance algorithm. Additionally, it has a maximum window size of 20 packets. In fact, N0, N1, and N2 cannot buffer at initial 20 packets burst, so the expected loss comes after transmission starts by 0.1 s. After about 0.57 s, it begins receiving duplicated ACKs, and finally after 1 s the timeout is expired and then the lost packet is retransmitted. When the acknowledgement reaches around 1.2 s, the congestion window is set to half size of last window and continues by sending 20 packets, which is too much for R1-R2 link.

b. Queue size comparison

Generally, the effect of the new slow-start mechanism on the behaviour and performance of congestion window is distinguished. It can easily note the number of windows clocking within 6 s and compare it with the standard mechanism. Figure 3.12 shows the progression of packets number over time where this metrics represents the TCP queue size evaluation. The time (in seconds) is on the x-axis and the relative sequence number of packets is on the y-axis.

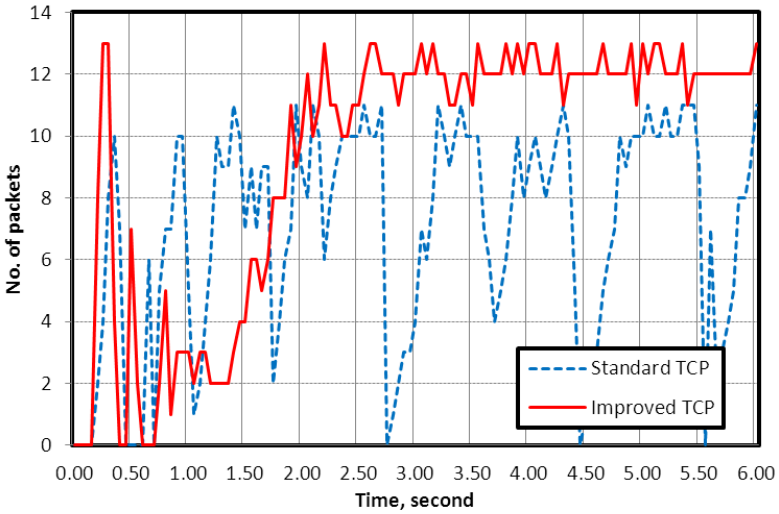


Figure 3.12 Queue size comparisons between standard and improved Tahoe over congested network

The queue size at a bottleneck would affect the performance of TCP protocols, especially with single TCP flow and large bandwidth path. There are many studies on TCP/RTT determined by evaluating packet traces, but it is not efficient if the maximum RTT is used to estimate the bottleneck queue size. In this test, the maximum values of one-way delays are immediately measured when a packet loss occurs and leads to queue overflow.

To estimate the number of queued packets for each RTT interval in the slow-start phase, the following formula in Equation (3.10) is used (Hirabaru 2006).

$$N = \frac{W * RTT(R_i - R_o)}{BW} \tag{3.10}$$

where N is the number of queued packets, W is the window size in packet, BW is the link bandwidth capacity, R_o is the bottleneck rate (in this experiment, 4 Mbps), W_i is the initial window size (assumed to be equal 2), and R_i is the burst rate at which the TCP source generates the traffic.

In Equation (3.11), the standard TCP increases the windows size by two when ACK is received so that R_i is limited to 2*Ro* (Hirabaru 2006).

$$W = \frac{W_i * 2 * Time}{RTT} \tag{3.11}$$

The standard TCP cannot properly calculate the bandwidth when bottleneck queue is small. In Figure 3.12, when sender starts transmitting the packets, the two TCPs (standard and improved) give the same performance and the queue size is approximately similar in shape. However, after one second, the improved slow-start forces the queue to be near the maximum level (14 packets) while the standard algorithm is below 12 packets. This is due to the fast growth of the window when duplicating window size is used instead of using the slow method via linear increase. The low queuing showed in the period between 0.5 s and 2 s is because of the improved congestion control still not finished the completed window of *cwnd* and the improved slow-start is slow-down to the initial *cwnd* level earlier than the standard.

c. Packet loss comparison

The queue size achieves better performance, which means the number of expected packet losses is also minimised as the number of retransmissions are decreased as shown in Figure 3.13. Mostly, in slow-start phase, several packets will drop when standard algorithm is used and this continues during the increase of window until packet loss occurs before starting the congestion avoidance phase.

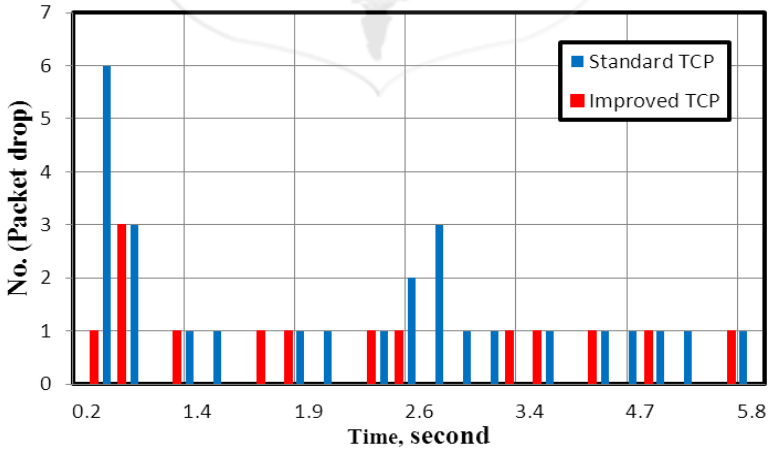


Figure 3.13 Packet loss comparisons between standard and improved Tahoe over congested network

There is no perfect mechanism that is able to detect loss before it occurs and for each slow-star period, at least one packet should drop to trigger the congestion control to enter congestion avoidance phase. In fact, TCP Tahoe (used in this experiment) generally gives higher loss but when the improved slow-start scheme is implemented, the number of packet loss is also minimised to half as shown in Figure 3.13. As a result, the expected throughput for the network should be better than the performance when standard slow-start algorithm is used.

d. Throughput comparison

TCP throughput rate assumed here is built on the Maximum Segment Size (MSS), RTT used for the experiment, and to the Probability of Packet Loss (PPL). The basic formula to estimate the throughput is shown in Equation (3.12). MSS value sets to 1460 bytes and PPL sets to 10^{-8} (this value must not exceed 10^{-8}) (Mathis et al. 1997).

$$Throughput \leq \left[\frac{MSS}{RTT} * \frac{1}{\sqrt{PPL}} * \sqrt{\frac{3}{2}} \right] \tag{3.12}$$

Equation (3.12) is proposed by Mathis et al. (1997) and termed as Mathis Equation. It summarises what occurs if TCP experiences loss in the path. Figure 3.14 shows the throughput comparison between the standard and improved slow-start.

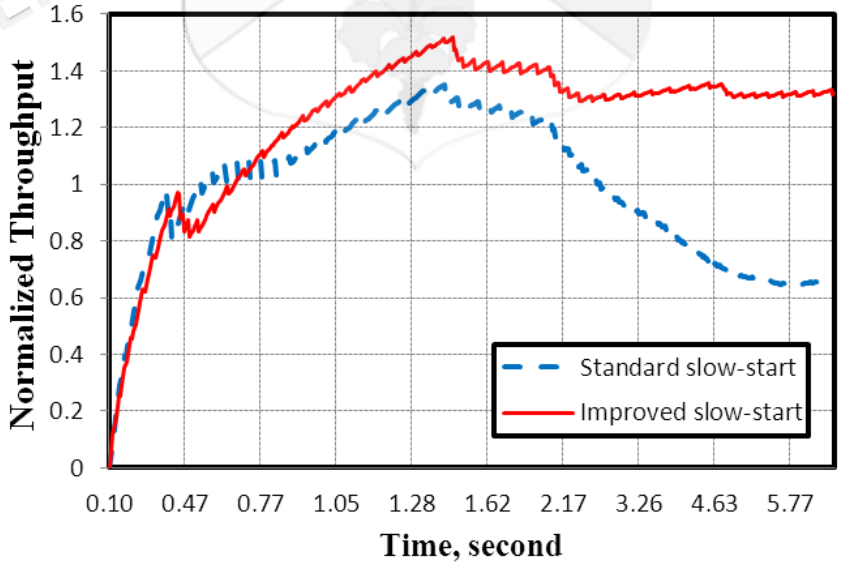


Figure 3.14 Throughput comparisons between standard and improved Tahoe over congested network

The fine performance of the new algorithm shown in Figure 3.14 is because of the small RTT used in the network topology. Thus, the short RTT allows TCP source to send a high packet rate to the destination. That means, when the slow-start time is shorter, it will lead to making difference in number of transmitted packets during the same RTT numbers. Besides, it will delay the congestion point of the connection, so it can send larger amount of packet within the same period as it can start-up *cwnd* faster than the standard slow-start. Essentially, the throughput measured and compared in this research based on the normalized throughput and not includes and measurements units to achieve better visibility and comparisons between different throughputs.

3.4 SLOW-START MECHANISM WITH LAGRANGE INTERPOLATION INCREMENTS

The linear increment of TCP slow-start mechanism leads to very long period to start up the window that causes low exploitation to the available bandwidth of the network path. In addition, in initial slow-start phase, the sender nodes are not able to recognise the available network bandwidth because it uses the default values of *ssthresh* (Wang & Yuan 2008). In previous slow-start technique explained in Section 3.3, multiple packets are sent through one sending window to force the window to grow faster and to control the packet dropping by using interpolation increment to avoid from exceeding the *ssthresh*. Although this technique incremented by more than one packet in the same transmitting window, it cannot always perform well. This is due to the characteristic of TCP that makes the performance radically degraded. Furthermore, the increasing phase via standard linear interpolation scheme needs extra improvement to get smooth slow-start. Therefore, it may be an operative technique that can defeat the failure of the TCP congestion control in slow-start phase by using other specialised methodologies to obtain an intelligent increment to the congestion window. The proposed algorithm is built on duplicating the congestion window size for each ACK and when *cwnd* is less than *ssthresh*/*2*. The duplicating process continues until the double size of *cwnd* becomes larger than *ssthresh/2*. At this point, the congestion window increment goes to increase *cwnd* value by precise steps to avoid from exceeding the *ssthresh* of the network path. In slow-start mechanism as introduced in Section 3.3, the linear interpolation algorithm is used to perform this operation.

In fact, the linear approximation to the window size takes long time and much more iteration when the window size is large. For example, if the window size reaches 64 packets while the current *ssthresh* is 120 packets, that means, the required iterations to reach *ssthresh* can exceed 22 steps and it requires time and processing by congestion control. Therefore, it is necessary to minimise the number of iteration used to approximate the window size when the new mechanism cannot duplicate the window anymore.

More intelligent and complex technique is used in the congestion window increment when *cwnd* > *ssthresh/2*. Instead of using two sequenced points to approximate the next window size used in linear interpolation, the new approach applies new technique to approximate the next window size with accurate value. This operation is performed using Lagrange interpolating polynomial to interpolate the window size.

3.4.1 Lagrange Interpolating Polynomial

Interpolation is a fundamental topic in the numerical analysis that is based on constructing a function that goes through a given set of data points. The idea of interpolation is to find a function of a specified form, which passes through a given list of points. In some applications, these data points are obtained by sampling a function or process. Subsequently, the values of the function can be used to construct an "Interpolant", which must agree with the interpolated function at the data points (Saniee 2007). The simplest kind of interpolation, in which most development has been made, is an interpolation by "univariate" polynomials. Multiple formulae for polynomial interpolation have been given, such as Newton (Ron et al. 2010) and Lagrange (Gasca & Sauer 2000).

Polynomial interpolation involves finding a polynomial of order *n* that passes through the *n+1*data points. One of the methods used to find this polynomial is called the Lagrange Interpolating Polynomial (Hildebrand 1987). As shown in Figure 3.15, Lagrange's interpolation uses polynomials.

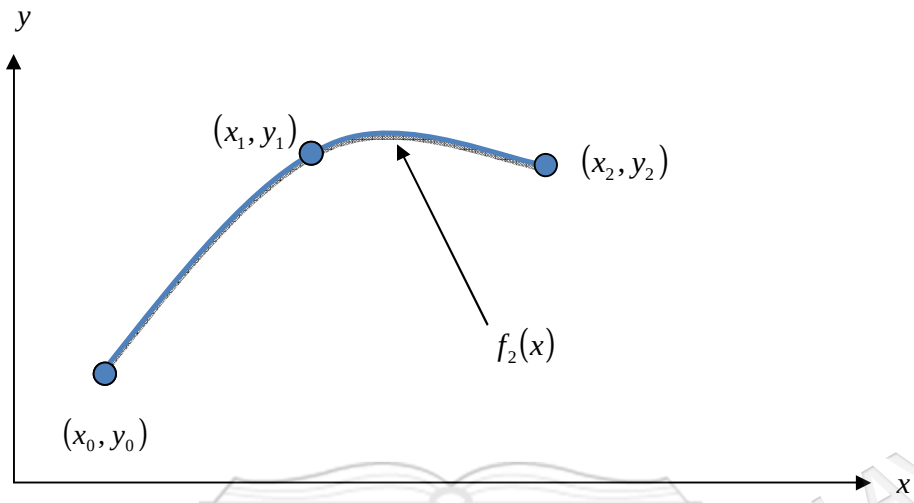


Figure 3.15 Quadratic interpolation

When three points, (x_1,y_1), (x_2,y_2), and (x_3,y_3) with different x-coordinates, either the three points lie in a line, or exactly lie on parabola through the three points. The second-degree polynomial can be represented by $y = ax^2 + bx + c$, and this represents a second order polynomial interpolation called Quadratic Interpolation.

If the three points lie in a line, there is a line $y = bx + c$ that passes through these points. Besides, it can be said that in any case, there is a polynomial of degree at most two passing through the three points. The Lagrange interpolating polynomial is the polynomial degree$\leq n-1$ that passes through n points. The Lagrange interpolating polynomial is given by:

$$f_n(x) = \sum_{i=0}^{n} L_i(x) f(x_i) \tag{3.13}$$

where n in $f_n(x)$ stands for the n^{th} order polynomial that approximates the function $y=f(x)$ given at $n+1$ data points as (x_0,y_0), (x_1,y_1),……, (x_{n-1},y_{n-1}), (x_n,y_n).

Then:

$$L_i(x) = \prod_{\substack{j=0 \\ j \neq i}}^{n} \frac{x - x_j}{x_i - x_j} \tag{3.14}$$

where $L_i(x)$ is a weighting function that includes a product of $n-1$ terms with terms of $j=i$ are omitted. For three points (x_0,y_0), (x_1,y_1), and (x_2,y_2), the quadratic polynomial $y=L(x)$ is equal to 1 at x_0 and equal to 0 at x_1 and x_2.

$$L_0(x) = \frac{(x - x_1)(x - x_2)}{(x_0 - x_1)(x_0 - x_2)} \tag{3.15}$$

Similarly, it can find the quadratic polynomials $L_1(x)$ and $L_2(x)$ such that:

$$L_1(x) = \frac{(x - x_0)(x - x_2)}{(x_1 - x_0)(x_1 - x_2)} \tag{3.16}$$

and,

$$L_2(x) = \frac{(x - x_0)(x - x_1)}{(x_2 - x_0)(x_2 - x_1)} \tag{3.17}$$

Then:

$$f(x) = y_0 * L_0(x) + y_1 * L_1(x) + y_2 * L_2(x) \tag{3.18}$$

The Lagrange form illustrated in Equation (3.18) represents a polynomial of degree at most three, which passes through the points (x_0,y_0), (x_1,y_1), and (x_2,y_2). When constructing interpolating polynomials, there is a trade-off between having a better fit and having a smooth well-behaved fitting function. The extra data points used in the interpolation will increase the degree of the resulting polynomial, and therefore, the greater oscillation it will exhibit between the data points.

3.4.2 Formulation of Proposed Slow-Start Algorithm

To apply Lagrange interpolation in slow-start algorithm, we need to assume the candidate point s to get the next value of the congestion window. To simplify the required relationship between *cwnd* and *ssthresh*, it can be writing:

$$y = A.x^2 + B.x + C \tag{3.19}$$

where *y* is the value of *cwnd* after ACK and *x* represents the current *cwnd*.

Then:

$$cwnd\ (t+\tau) \rightarrow y$$
$$cwnd\ (t) \rightarrow x$$

The coefficient *C* is neglected to avoid the shift in *cwnd* level while other coefficients, *A* and *B* can be calculated using the relationship between the value of *ssthresh* and the time differences for RTT period. Then, after applying these parameters to estimate the coefficients using Equation (3.19), the result is as follows:

$$cwnd\ (t+\tau) = C + B.cwnd\ (t) + A.(cwnd\ (t))^2 \tag{3.20}$$

The derived and estimated values of *A* and *B* explained in details in Appendix B, where the implementation of quadratic Lagrangian interpolation gave the following:

$$A = \frac{-1}{ssthresh}$$
$$B = 2$$

Then, Equation (3.20) can written in other form after add the coefficients as shown below:

$$cwnd\ (t+\tau) = 0 + (2) * cwnd\ (t) + (\frac{-1}{ssthresh}) * (cwnd\ (t))^2$$

Therefore, the final formula will be:

$$cwnd\ (t+\tau) = 2\,cwnd\ (t) - \frac{1}{ssthresh}\left(cwnd\ (t)\right)^2 \tag{3.21}$$

Equation (3.21) shows the required formula to increase the congestion window size when double size of *cwnd* is larger than *ssthresh/2*. The function of Lagrange interpolating polynomial is to achieve accurate increment to the congestion window when duplicating window approach is used. Therefore, the final scheme to the proposed slow-start algorithm for each ACK is as shown in Equation (3.22):

$$cwnd(t+\tau) = \begin{cases} 2 * cwnd\ (t) & \text{if } cwnd(t) \le \text{ ssthresh}/2 \\ 2cwnd(t) - \dfrac{1}{ssthresh}\left(cwnd(t)\right)^2 & \text{if (ssthresh}/2 > cwnd \;\&\& \\ & \quad cwnd(t) < \text{ssthresh)} \\ Congestion\ avoidance & \text{elsewhere} \end{cases} \tag{3.22}$$

This algorithm can provide faster growth to the congestion window for TCP sender compared with the previous technique because it reduced the required iterations to reach the *ssthresh*. To explain the difference in using linear and quadratic interpolations, a simple example proposed in Appendix C.

3.4.3 Performance Evaluation of Proposed Slow-Start Mechanism

The evaluation of the proposed slow-start mechanism is based on the same network topology in Section 3.3.3. The new slow-start mechanism is applied over TCP Tahoe by removing the previous standard algorithm, inserting the new algorithm, and making the required modifications to the main source file **tcp.cc**.

The pseudo code of the proposed algorithm is arranged as follows:

```
        /* Slow-Start phase*/
if (cwnd_ <= ssthresh_/2){
       /* Duplicating cwnd */
     cwnd_ = 2*cwnd_ ;
         }
if (cwnd_ >= (ssthresh_/2) && cwnd_ < ssthresh_){
        /* Quadratic interpolation */
 cwnd_= 2*cwnd_-(cwnd_*cwnd_)/ssthresh_ ;
```

As shown in Figure 3.16, the slow-starts of standard and improved algorithms are presented. The standard growth needs 0.621 s to complete the initial slow-start phase and reaches *ssthresh* (128 packets) while the improved TCP reaches *ssthresh* level within 0.203 s. That means the new slow-start scheme is faster than the typical scheme and gives reasonable period to start up *cwnd*.

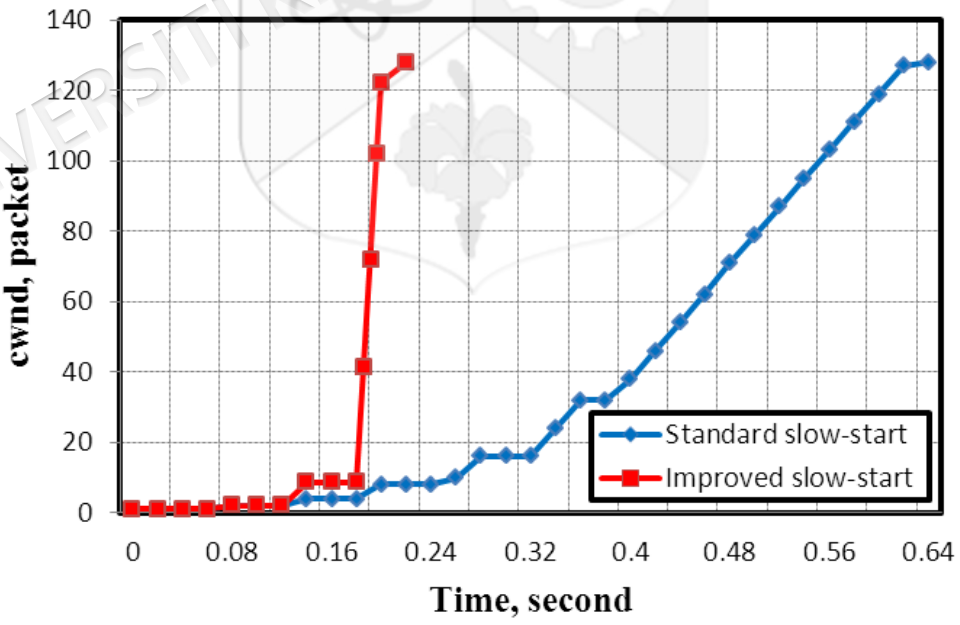


Figure 3.16 Initial slow-start phases for standard and improved TCP

The effect of the proposed mechanism will appear with large bandwidth network or the congestion window size becomes large. Therefore, the proposed technique allows the congestion window to increase within short period and start the next phase while the standard algorithm with linear-exponential increase needs longer time to reach the same level because it is based on single increment.

a. Congestion window comparison

The congestion window experimented with maximum *ssthresh* of 128 packets is as shown in Figure 3.17 but without congestion events. Two TCPs (standard and improved) are tested, so it is easy to detect the wide gap in terms of time between them. The first congestion window phase of the standard TCP is completed after 8.2 s, while the improved TCP is only requires 7.3 s to complete the same interval.

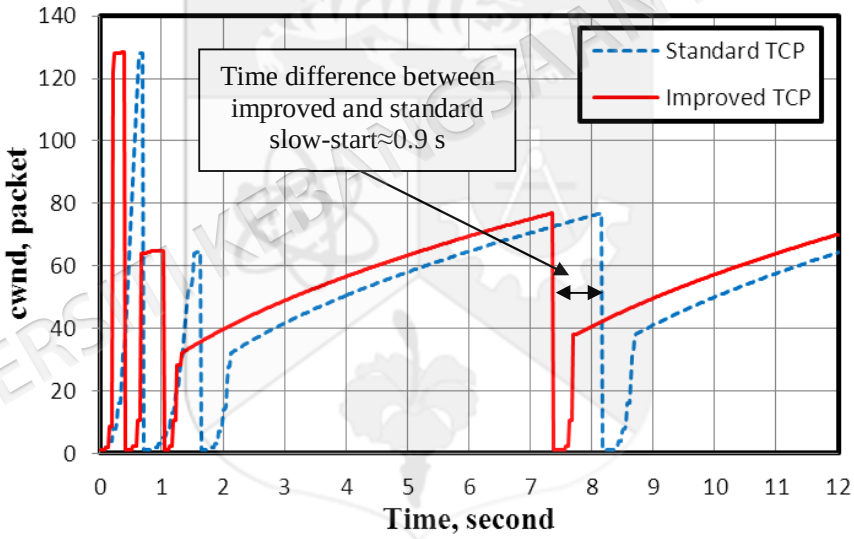


Figure 3.17 Behaviour comparisons of *cwnd* for standard and improved TCP without congestion events

In this experiment, the fast growth in congestion window is performed by the new slow-start mechanism, which allows the TCP sender to transmit larger amount of packets exceeding the ability of the standard TCP. Moreover, the alterations in performance between two approaches (standard and improved) should be increased when the network path is still full and when the TCP sender tries to inject new packets through the network pipeline.

Other experiment is used to test the new and standard slow-start mechanism with early loss event in network with 20 packets as maximum window and 1460 bytes as a packet size. As shown in Figure 3.18, the congestion window of new congestion control still performs well with loss and congested network path. Confidently, the fast window growth assists TCP sender to send more packets because it needs shorter period to send the packets and to receive the acknowledgment. Furthermore, even though the loss occurs, there is enough time (compared to standard slow-start algorithm) to retransmit the lost packet again.

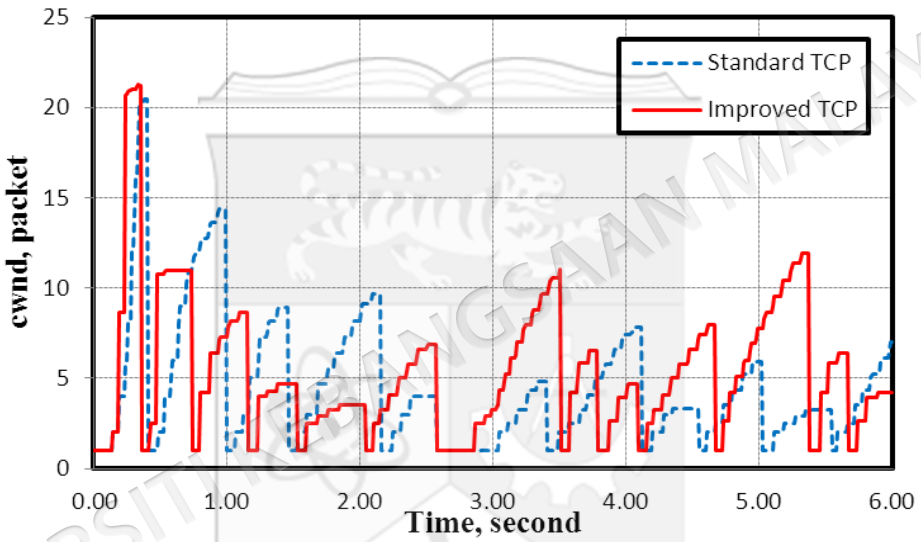


Figure 3.18 Congestion window comparisons of standard and improved TCP with high-congested connections

b. Queue size comparison

The monitoring of queue size management of the bottleneck pipe R1-R2 is shown in Figure 3.19. The behaviour in Figure 3.19 proves that the proposed mechanism can open the queue length to be near the maximum size (14 packets) while the standard mechanism keeps the queue size under 11 packets.

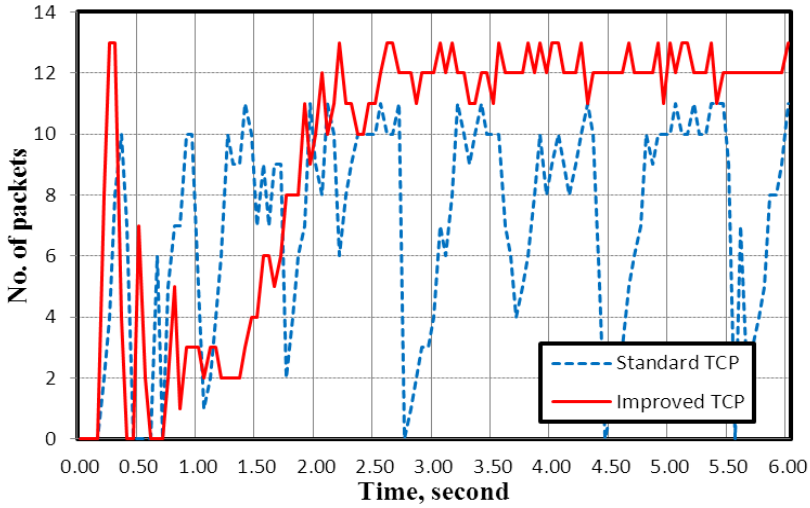


Figure 3.19 Queue size management comparisons of standard and improved TCP

In fact, the queue size starts to be stable after 3 s because the improved technique still gives loss in the early intervals and a loss level must be expected when the packet drop is observed. The good behaviour of new algorithm in queue management is due to the available time to resend the lost packets and because the TCP Tahoe used in this experiment has poor loss recovery mechanism.

c. Packet loss comparison

The comparison of packet loss between the standard and improved slow-start is illustrated in Figure 3.20.

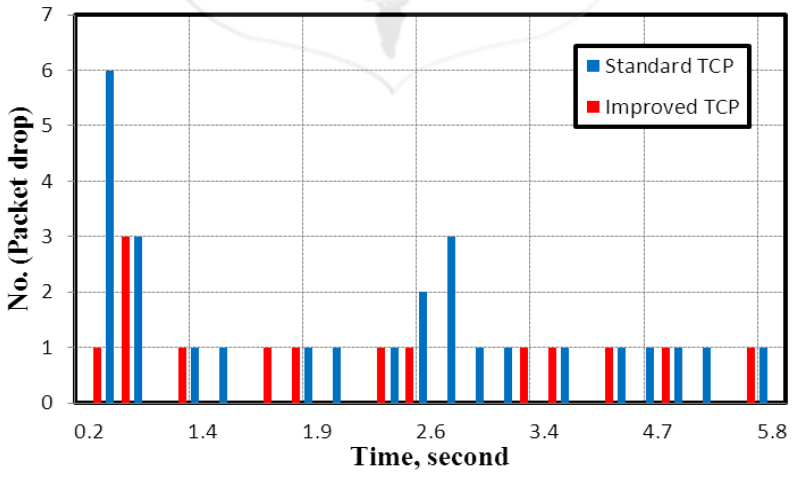


Figure 3.20 Packet loss comparisons of standard and improved TCP

As shown in Figure 3.20, the large packet loss happens before the third second and then it is minimised to normal level while the standard TCP continuously runs but with high packet drop.

d. Throughput comparison

When the throughputs for the standard and improved TCP are tested, the performance of the improved TCP is obviously better than the standard and the variance between the two throughputs continuously increase over the network simulation session. MSS value is set to 1460 bytes, while the probability of loss is set to 10^{-8}. In the early network session, throughput comparisons show that the two TCPs give low throughput, but within next proceedings, the performance of the new TCP begins to perform well and the number of the acknowledged packets increases. However, TCP Tahoe cannot give high performance and this is because of the classic mechanism used to control the congestion over the network connections. Essentially, the reason behind the improved performance of Tahoe in Figure 3.21 is the result of the enhanced slow-start mechanism used.

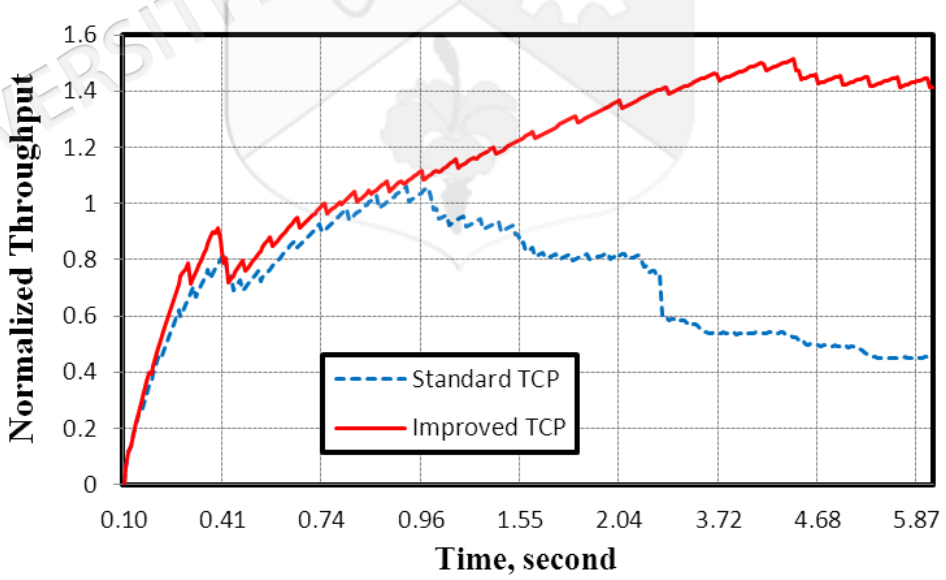


Figure 3.21 Throughput comparisons of standard and improved TCP

The enhanced slow-start allows Tahoe sender to transmit larger number of packets within shorter interval. Moreover, the proposed slow-start technique allows to delay the congestion state and to authorise Tahoe sender to send extra packets that exceed the packets amount transmitted by the standard Tahoe during the same session.

3.5 SUMMARY

In this chapter, two slow-start models are presented to improve the performance of TCP congestion control over highly congested networks. The two models minimise the required period to more than half of the standard slow-start algorithm. Therefore, the new TCP, which is based on new slow-start scheme, can give quick start-up to the congestion window during window initialisation or in other congestion window stages. Both mechanisms can send higher number of packets than the linear-exponential slow-start used in standard TCP source variants. The first proposed slow-start model is built on duplicating the window size when *cwnd* is less than the half of *ssthresh* and then linear interpolation is used to increase the window until the *ssthresh* is reached. The second model gives faster congestion window start-up and then new approximation approach is used based on Lagrange interpolating polynomials to achieve rapid growth in congestion window before it reaches the *ssthresh* with minimum number of iterations. Numerically, the enhanced slow-start mechanisms provided shorter increase and reduced the slow-start period in initialization phase by 60% for linear mechanism and 67% for quadratic mechanism compared to standard slow-start. Additionally, both slow-start schemes are experimented with highly congested network scenario involving many terminal nodes with different bandwidth and propagation delay links. The proposed topology also involves two cascaded bottlenecks with re-routing scenario to give more reality to the experiment. In these experiments, the proposed mechanisms are added to the TCP Tahoe, and the performance evaluation of congestion control window proves that the improved algorithm performs better than the standard algorithm and gives higher throughput and lower packet losses with suitable queuing management.

CHAPTER IV

ENHANCED TCP CONGESTION AVOIDANCE MECHANISM

4.1 INTRODUCTION

Congestion avoidance phase starts when TCP sender detects packet drop in the network path and then sets the value of *ssthresh* to be equal to one-half of the current *cwnd*. The sender decreases the transmission rate by returning to the slow-start mode, but in this case, it exponentially increases the packets rate until *cwnd* becomes equal to *ssthresh*. In this case, TCP sender linearly increases *cwnd* size (which typically increases one segment every RTT). If the size of *cwnd* is equal or less than to *ssthresh* value, then sender is in slow-start phase, and if *cwnd* is greater than *ssthresh* that means the TCP sender is in congestion avoidance phase.

The combination between slow-start and congestion avoidance forces TCP to decrease *cwnd* size by half for each time it detecting packet losses. When congestion detects packet loss it continues for a period and the traffic injects certain packets amount into network and the retransmission rate in the sender side decrease exponentially. Unfortunately, TCP does not perform well with high-congested networks or when the link suffers from high latency due to the simultaneous packet losses. This is because the TCP congestion window in the congestion control has been reduced to half of its current size. Therefore, the large congestion window will take a long time to reach to its present position. Then, the proposed congestion control techniques are based on the developed new mechanism, which takes in to account the large bandwidth available over high-speed networks.

As discussed in Chapter II, the performances of standard TCP variants with traditional congestion control algorithms are significantly reduced if they are used over wireless networks. Furthermore, the standard TCP does not perform properly over high speed and wide area networks. In order to explain this problem, a simple case study is illustrated here. If the target is to obtain a throughput of 7 Gbps when RTT is 100 ms and 1500 bytes as packet size, the formula in Equation (3.12) is used to estimate the steady-state throughput (Welzl 2010 ; Nabeshima 2005) and as explained in Section 3.3.3 d, the throughput can be estimated as following:

$$Throughput = \frac{MSS}{RTT\ \sqrt{PPL}} * C$$

where C is constant and equivalent to $\sqrt{3/2}$ (Mathis et al. 1997). Then, the packet loss probability (PPL) can be estimated, as follows:

$$7 \times 10^{9} = \frac{1500 * 8}{100 \times 10^{-3} \sqrt{PPL}} * \sqrt{3/2}$$

Alternatively, it can be written in term of PPL, as shown below:

$$\sqrt{PPL} = \frac{1500 * 8}{100 \times 10^{-3} * 7 \times 10^{9}} * \sqrt{3/2}$$

$$= 2.09956 \times 10^{-5}$$

$$PPL = 4.408 \times 10^{-10}$$

This means that the expected probability of loss should not reach 4.408 x 10^{-10}, whereby it represents inaccessible error rates. However, this value should not exceed 10^{-8} that was mentioned in Section 3.3.3 d. In addition, TCP needs a huge number of RTT's to recover one packet loss and this will cause TCP to be completely unemployed by the existing link capacity.

In order to develop TCP performance over high-speed wireless networks, several congestion control techniques, such as TCP Veno (Fu & Liew 2003) and TCP Westwood (Mascolo et al. 2001), have been proposed. Although these congestion control techniques have succeeded in their individual objectives, the access to the optimum congestion control over wireless networks or high-speed networks remains a big challenge. Additionally, with the utilization of high-speed wired-wireless networks, such as LTE, LTE-Advanced, and WiMAX, it is necessary for the required congestion control algorithm to handle the wireless links losses besides the typical congestion in the wired networks (Wang et al. 2011). In Section 4.3, an enhanced congestion avoidance mechanism is proposed to form an innovative TCP congestion control, which is operated stably for large bandwidth, low latency, wired-wireless networks.

4.2 ADDITIVE INCREASE MULTIPLE DECREASE ALGORITHM

The TCP protocol uses *ssthresh* to ensure that *cwnd* does not increase forever. Theoretically, *ssthresh* detects the appropriate size of the congestion window corresponding to the existing network capacity. Once it exceeds *ssthresh*, TCP enters into the congestion avoidance phase. In additive increase phase and for every ACK received, *cwnd* is improved by (*1/cwnd*) segments. This is roughly equivalent to growing *cwnd* by one segment for each RTT (Floyd & Jacobson 1991). In multiple decreases phase and when congestion is detected, the transmitter will decrease the transmission rate by a multiplicative factor; for example, the congestion window is cut in half after loss. The result is a saw-tooth behaviour that represents a probe for bandwidth. The TCP source experiences congestion in the network if the period goes out for ACK. Then, *ssthresh* is set as the maximum value between two segments and between the minimum of the halved *cwnd* and the receiver window or the advertised receiver window (rwnd) (Demers et al. 1989). This approach is called the AIMD algorithm, where the proposed congestion avoidance technique is based on. Principally, when TCP uses the AIMD algorithm, the transmitted packets are controlled by halving the congestion window for each window of the data, including a packet loss and incremented by one segment for each acknowledged packet.

AIMD is also a feedback control technique that is well-known for its use in TCP congestion avoidance. AIMD associates a linear increase of *cwnd* with an exponential decrease if any congestion occurs (Chiu et al. 1989). Later, AIMD is complemented with two more processes, termed as fast recovery and fast retransmit, to improve the performance in certain states.

The AIMD congestion control algorithm is based on the scheme demonstrated by Floyd et al. (2000). AIMD algorithm proposes that a packet is lost when *cwnd* reaches *w* packets size, as illustrated in Figure 4.1 below. On the other hand, the stochastic TCP model proposed by Floyd et al. (2000) considers that the packets are lost with random probability *PPL* and TCP retransmit timeouts, which enable an accurate TCP model to be generated.

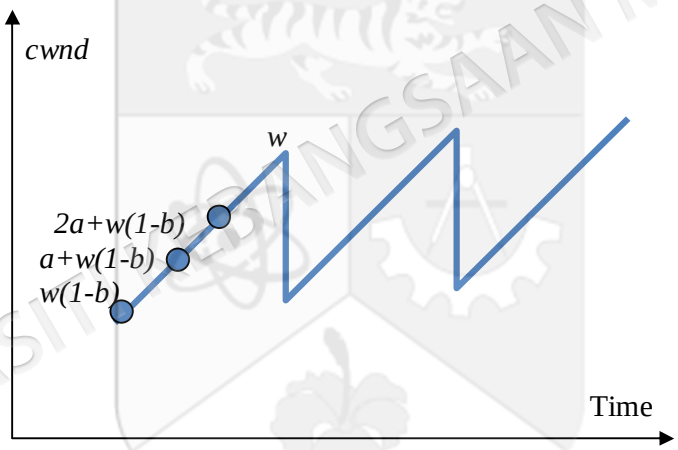


Figure 4.1 Congestion window based on AIMD

In AIMD modelling, the congestion epoch begins with the *cwnd* of *w(1-b)* packets. Then, *cwnd* is uniformly increased by *a* packets for each RTT, up to *w* window size when a packet loss occurs. At that time, *cwnd* is reduced multiplicatively back to *w(1-b)*, where each congestion epoch involves *w(b/a)+1* of RTT. Meanwhile, if *S* denotes sending packets for each RTT, and *T* denotes the sending packets for each second, the average sending rate for one congestion period in the packets per RTT (Floyd et al. 2000) is as follows:

$$S = w\frac{2-b}{2} \tag{4.1}$$

In addition, the average sending rate, T for one congestion period in packets per RTT is:

$$T = w\frac{2-b}{2RTT} \tag{4.2}$$

The total number of packets for each congestion epoch can be given as:

$$S\left(w\frac{b}{a}+1\right) = w\frac{2-b}{2}\left(w\frac{b}{a}+1\right) \tag{4.3}$$

$$= w\frac{2-b}{2}\left(w\frac{b}{a}+1\right)$$

$$\approx \frac{w^2(2-b)b}{2a} \tag{4.4}$$

At the end of the congestion epoch and with one packet loss, the rate of packets drop p becomes:

$$p = \frac{2a}{w^2(2-b)b + w(2-b)a} \tag{4.5}$$

When $a < 1$and $b < ½$, the line can be extended as shown in Figure 4.1 using an approximation for Equation (4.5) (Floyd et al. 2000a):

$$p \approx \frac{2a}{w^2(2-b)b} \tag{4.6}$$

After approximated Equation (4.6) has corresponded to *cwnd*, the formula can be simplified as follows:

$$w \approx \sqrt{\frac{2a}{p(2-b)b}} \tag{4.7}$$

Now, by substituting Equation (4.7) in Equation (4.2) to estimate the sending rate *S* in packets per second, the following can be obtained:

$$S = \frac{\sqrt{a}\sqrt{(2-b)}}{RTT\ \sqrt{p}\sqrt{2b}} \tag{4.8}$$

Assuming that a=1 and b=0.5, putting that in Equation (4.7) will get the following:

$$S = \frac{\sqrt{1.5}}{RTT\ \sqrt{p}} \tag{4.9}$$

When the relationship between the AIMD parameters a and b has the same long-term sending rate to the rate of packets loss, when a=1 and b=0.5, then the response function of both AIMD will become:

$$\frac{\sqrt{1.5}}{RTT\ \sqrt{p}} = \frac{\sqrt{a}\sqrt{(2-b)}}{RTT\ \sqrt{p}\sqrt{2b}} \tag{4.10}$$

Equivalently, Equation (4.10) can be written as follows:

$$a = \frac{3b}{2-b} \text{ and } b = \frac{2a}{3+a} \tag{4.11}$$

Equation (4.11) represents the approximated resolution of AIMD deterministic model which can provide a suggestion for (*a, b)* values such as (1/5,1/8) and (3/7, 1/4) where each value should compete fairly reasonable with AIMD (1, ½). The windowing of AIMD algorithm represents a perfect choice for many present applications due to its demanding usage of the available bandwidth. On the other hand, for several real-time applications such as multimedia over TCP/HTTP, the necessity for moderately smooth modifications of the transmitting rate is more imperative than the capability of increasing the capacity of the available bandwidth.

Moreover, in some applications, the main purpose is not to directly use the congestion control of TCP, but to avoid sudden halving of the transmitting rate in the reaction, even to single packet loss.

4.3 ENHANCED CONGESTION AVOIDANCE ALGORITHM

AIMD algorithm used to adjust the congestion window and it involves linear increment of *cwnd* with the exponential decrease when packet loss occurs. In congestion avoidance stage, if TCP sender *j* receives an ACK, it will increment *cwnd*, depending on the additive increment rule in Equation (4.12) while in a loss event, it will decrease *cwnd* according to Equation (4.13).

For each RTT:

$$cwnd_j = cwnd_j + (a_j / cwnd_j) \tag{4.12}$$

For each loss event:

$$cwnd_j = cwnd_j - b_j * cwnd_j \tag{4.13}$$

In Reno TCP the parameters pair in AIMD is proposed to be a_j =1 and b_j =0.5. Accordingly, the TCP sender steadily triggers the packets transmission and when a loss is detected, the sender will enter the fast recovery phase and the dropped packets are retransmitted. To describe the proposed congestion avoidance algorithm in detail, let assume that each physical session involves *k* number of virtual TCP Reno sessions. Then, *cwnd* for each virtual TCP Reno session is regulated using the AIMD scheme, as follows:

For each RTT:

$$cwnd = cwnd + (1 / cwnd) \tag{4.14}$$

For each loss event:

$$cwnd = cwnd - \frac{cwnd}{2} \tag{4.15}$$

Consequently, the *cwnd* of Reno with virtual session's *k* can be stated as follows:

For each RTT:

$$cwnd = cwnd + (k / cwnd) \quad (4.16)$$

For each loss event:

$$cwnd = cwnd - \frac{cwnd}{2k} \quad (4.17)$$

In this case, the *cwnd* of Reno represents the congestion window for the standard Reno, including *k* number of the virtual session of packet flows.

As mentioned previously, the proposed congestion avoidance algorithm is built over general AIMD concept to achieve additive increase and multiplicative decrease using the parameters *a* and *b* respectively. The proposed congestion avoidance algorithm regulates the values of *k* according to the network status and the network bandwidth. When associating the parameter *k* with the control factors *a* and *b* in Equation (4.11) obtained from the general AIMD algorithm, the final congestion window will then be multiplied by *k*, and this will increase of the *k* packets for each RTT.

In Equations (4.16) and (4.17), the AIMD algorithm with the parameters *a* and *b* can be rewritten by substituting the factor *k* as a number of virtual TCP sessions to obtain the new AIMD model as a function of *k*. In the AIMD algorithm, when the loss packet is detected, the *cwnd* is decreased by (1-*b*) times of the current *cwnd*, while a number of packets increase it for each RTT. Now, to implement factor *k* over the AIMD algorithm, it can be said that when *a=k=1*, the congestion window will increase by a single packet on every RTT. In the decreasing mode, when one packet loss occurs, the *cwnd* in Equation (4.17) will decrease by *b*, if *b*=½*k*.

4.3.1 Limitations of Standard AIMD

The implementation of AIMD (*k*, ½*k*) algorithm over TCP congestion control is still limited due to its difficulties in achieving the *k*-TCP virtual flows instead of a single flow. AIMD (*k*, ½*k*) cannot provide throughput as *k* times the throughput is provided by a single TCP. In order to solve this problem, Crowcroft and Oechslin (1998) proposed a parallel protocol using AIMD (*k*, ½*k*) to emulate the behaviour of *k* TCP flows, which is termed as multiple TCP (MulTCP). MulTCP differs from the standard TCP variants because it permits data stream to flow over the available link bandwidth equivalent to *k* times that of a standard TCP stream (Mulroy et al. 2009).

In standard TCP, *cwnd* increases by single packet per RTT or by *1/cwnd* for each ACK, while in MulTCP, *cwnd* increases by *k* packets per RTT or by *k/cwnd* for each ACK. On the other hand, when the standard TCP detects congestion, it halves *cwnd*, while MulTCP halves one k^{th} of the *cwnd* and scales it by *(k – ½)/k* or (1-*b*).

It is important to note that in MulTCP, the losing procedure is detected by one flow only and it is not dependent on the loss of all flows (Singh et al. 2004). Therefore, there is no guarantee if the flow of MulTCP will remain performing like *k*-TCP, especially when *k* is large. Additionally, Crowcroft and Oechslin (1998) have noted that the flow of MulTCP cannot be precisely affected like the *k*-TCP flows. This due to the timeout of packet loss forces MulTCP *cwnd* to slow down to one segment size. While in the *k*-TCP flows, the packet loss is slowed down by only one window of TCP connection to the size of one packet, while keeping the other window sizes the same. The other limitation is that the number of flows, *k* values, for MulTCP cannot reach the available bandwidth as *k*-TCP (*k*=4 is the recommended) (Gevros et al. 1999). In addition, despite the fact that MulTCP can give reasonable throughput over high-speed networks, the current version is still not the best choice for LTE or LTE-Advanced networks. This is because the performance of MulTCP is more sensitive in term of the standard TCP features and characteristics than the standard TCP due to the necessity to address the fairness issue in large bandwidth networks.

The direct implementation of AIMD (*k*, ½*k*) over MulTCP may decrease the performance of MulTCP, since the value of *k* required is to be set by the user and there is no mechanism to control *k* values. Besides, the modified slow-start algorithm of MulTCP causes problem when the advertised window of TCP receiver is not large enough as compared to MSS assigned to the link (Gevros et al. 1999). The other challenge involved in MulTCP is the throughput that it suffers from loss during timeouts, and thus, effective throughput will be reduced and it will not match the expected weight, *k*.

On other hand, MulTCP is certainly bursty than the standard TCP since the same number of ACK received can send more back-to-back packets. However, it will lead to extra delays in the router queue and the buffer remains in overflow. Besides, MulTCP has to be fulfilling during very short period and this may lead to packet loss again and may generally degrade the system performance.

The analysis and observation of MulTCP congestion control assist to monitor the behaviour of AIMD (*k*, ½*k*) algorithm and try to correct or avoid the failures. Then, it will be easy to improve the interrelation between the parameters, *a* and *b*, or by adding new techniques to make AIMD (*k*, ½*k*) that can perform better over high speed networks.

4.3.2 Formulation an Improved Relationship between *a* and *b*

The proposed solution to improve AIMD (*k*, ½*k*) algorithm will focus only on two major challenges. The first challenge is to estimate the optimum value of *k* dynamically by using the end-to-end congestion conditions of the network to regulate *cwnd*. The second challenge focuses on the ability to obtain the *k*-TCP flows in practice instead of using the standard TCP over the same connection. Then, it is important to achieve the main goal by obtaining the *k* flow level throughput provided by MulTCP or AIMD (*k*, ½*k*). The congestion control mechanism of the standard TCP versions use the AIMD algorithm to increase and decrease *cwnd* via the parameters pairs, *a* and *b*. The congestion window is reduced by (1-*b*) of the current size, if a packet loss event is occurred, and it is increased by *a* for each RTT.

Currently, the standard TCP uses AIMD (1, ½), besides the retransmitting timer and the exponential back-off of the retransmitting timer in high congestion links (Allman 2003). The estimation of *a* and *b* values for standard TCP is based on the relationship shown in Equation (4.11). For example, when TCP congestion control presumed a=1, then b=½.

AIMD (k, ½k) is used in MulTCP instead of AIMD (1,½) which is used in the standard TCP. MulTCP aims to provide multiple flows and can share the existing bandwidth along the congested connection by assuming a weighting factor k to the aggregated flows. The congestion control mechanism of MulTCP adopts the congestion control of the standard TCP to emulate the behaviour of k-TCP links by using the same congestion window loop. The investigation of the performance of AIMD (k, ½k) can be explained into two phases (Kuo & Fu 2003):

1. When a=k, *cwnd* of the single TCP flow is increased by single packet for each RTT in standard TCP, while in MulTCP *cwnd* of k flows of TCP is increased by k packets for each RTT.

2. When b=½k and during packet loss, one of the k TCP flows halves *cwnd*, while in MulTCP, the congestion window becomes k (1–½k) of its current value.

Nonetheless, the throughput provided by MulTCP with AIMD (k, ½k) is not a guarantee to achieve exactly k times of standard TCP, especially when k has a constant value over different congestion conditions of the network (Wang et al. 2011; Reale et al. 2012). Therefore, it is necessary to find an improved method to achieve a better throughput than the one provided by MulTCP with AIMD (k, ½k). Nabeshima (2005) proposed significant modifications for MulTCP by introducing new relationship between a and b. This new relationship operates in packet drop situation and is permitted to obtain an improved *cwnd* as compared to the typical MulTCP.

For this reason, the proposed congestion avoidance will be based on the improved MulTCP proposed by Nabeshima (2005) model to enhance the relationship between *a* and *b*, instead of using the standard relationship illustrated in Equation (4.11).

The main target of using the new relationship between *a* and *b* is to enable the parameter pair to tune to a wide range of preferred transmission rate. This relationship can be used by merging the parameters pair *(a,b)* with *k* so as to produce the required relative transmission rate. Firstly, for *cwnd* in the steady state, the average of *k* flows can be estimated, as shown in Equation (4.18):

$$cwnd_{k} = k * cwnd_{stnd} = \frac{k\sqrt{3/2}}{\sqrt{p}} \tag{4.18}$$

where $cwnd_k$ and $cwnd_{stnd}$ represent the congestion window of flow *k* and the standard *cwnd*, respectively, and *p* is the ration of packet loss. The formula used to calculate the average *cwnd* according to *p* is expressed as follow (Floyd et al. 2000):

$$cwnd = \sqrt{\frac{2a}{p(2-b)b}} \tag{4.19}$$

If *T* denotes the sending rate, and *cwnd* is estimated at the end of the congestion epoch, then:

$$T_{(a,b,p,RTT)} = cwnd\left(\frac{2-b}{2RTT}\right) \tag{4.20}$$

The congestion epoch is defined as a period beginning with a congestion window of *(1-b)w* packet (refer to Figure 4.1). Theoretically, when the throughput of the new TCP is *k* times of the traditional TCP, it means that:

:

$$T_{(a,b,p,RTT)} = k * T_{(1,1/2,p,RTT)} \tag{4.21}$$

By substituting a=1 and b=½ in Equations (4.19) and (4.20), the throughput of the standard TCP flow will then be estimated as follows:

$$T_{(a,b,p,RTT)} = \sqrt{\frac{2a}{p(2-b)b}}\left(\frac{2-b}{2RTT}\right) \tag{4.22a}$$

$$T_{(1,1/2,p,RTT)} = \sqrt{\frac{2}{p(1.5)0.5}}\left(\frac{1.5}{2RTT}\right) \tag{4.22b}$$

$$T_{(1,1/2,p,RTT)} = \sqrt{\frac{1.5}{p}} * \frac{1}{RTT} \tag{4.23}$$

By substituting Equation (4.23) in Equation (4.21), the following will be obtained:

$$k * \sqrt{\frac{1.5}{p}} * \frac{1}{RTT} = \sqrt{\frac{2a}{p(2-b)b}} * \frac{2-b}{2RTT} \tag{4.24}$$

where, p and RTT are the same in both sides, and after the simplification these become:

$$k\sqrt{1.5} = \sqrt{\frac{a}{b}} * \frac{\sqrt{2-b}}{\sqrt{2}}$$

$$k^2\frac{3}{2} = \frac{a}{b} * \frac{2-b}{2}$$

$$3k^2 = \frac{a(2-b)}{b}$$

Hence:

$$a = \frac{3bk^2}{2-b} \tag{4.25}$$

and:

$$b = \frac{2a}{a+3k^2} \tag{4.26}$$

This relationship between a and b can offer a wide range of probable values of a and b to accommodate the desired sending rate over different networks applications and conditions. For example, using $a=\sqrt{k}$, $b=2/(1+3k^{3/2})$ will then give a better performance compared to when $a=k$ is used over high-speed and wide area networks. This because when k is set to a large value, more packets will then be sent out during single RTT or may be lost (Nabeshima 2005). When $a=k$, it can estimate b to be as the following:

$$b = \frac{2}{1 + 3k} \tag{4.27}$$

The value of a is set to be the number of virtualized flows, while the value of b is estimated as expressed in Equation (4.27). This relationship represents the default values setting hereafter in the enhanced congestion avoidance algorithm. Therefore, according to the enhanced relationship between the parameters pair a and b, the estimation of the congestion window for each RTT, and the packet loss can then be rewritten as follows:

For each RTT:

$$cwnd = cwnd + (k / cwnd) \tag{4.28}$$

For each loss event:

$$cwnd = cwnd \left(1 - \frac{2}{3k + 1}\right) \tag{4.29}$$

In Table 4.1, some of the calculated values of a and b, which were estimated according to Equation (4.27) where $a=k$ and k are varied between 1 and 10 as a number of virtual flows.

Table 4.1 The estimated values of the parameters pair (*a, b*)

a=k	***b=2/(1+3k)***
1	1/2
2	2/7
3	1/5
4	2/13
5	1/8
6	2/19
7	1/11
8	2/25
9	1/14
10	2/31

In order to monitor the difference in the *cwnd* behaviour over the same network conditions between TCP(*k*, ½*k*) and the improved version TCP(*k*,2/(3*k*+1)), TCP Reno was used with the maximum segment size of 1460 bytes. In the first case, Reno was tested when *k*=1, and this represented the standard Reno (1,½). In the second test, when *k* was set to be equal to 2, and where the parameters pair became (2,2/7), 2 virtual flows were obtained for the improved TCP.

As shown in Figure 4.2, the enhanced Reno performed better than the standard Reno despite the fact that the improved TCP still used the standard slow-start (linear-exponential increase). In the other experiment, TCP Tahoe was used to test the congestion window development when *k*=4.

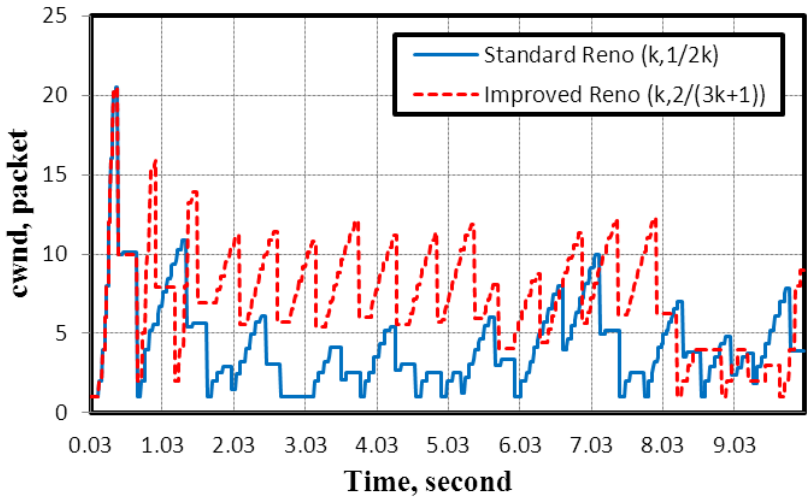


Figure 4.2 Behaviour of *cwnd* for Standard and improved TCP Reno when *k*=2

The purpose of using other TCP variant was to prove whether the improved scheme could work over any TCP version instead of the standard congestion avoidance algorithm. Although the improved congestion algorithm varied from one TCP version to another, the enhancements in *cwnd* were still maintained. In Figure 4.3, the *cwnd* of the improved TCP excluded fast retransmit mechanism because the Tahoe congestion control was not included in this phase. In other words, the effect of 4 virtual flows of the improved congestion control could clearly be distinguished and the random congestion events used in the simulation scenario shows the irregular *cwnd* behaviour due to the fast re-routing as shown in the seconds 0.8 and 7.

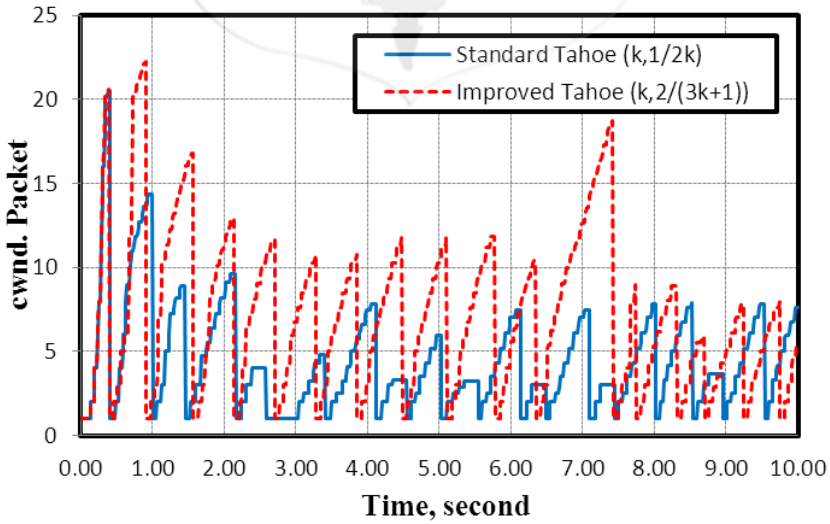


Figure 4.3 Behaviour of *cwnd* for standard and improved TCP Tahoe when *k*=4

The network topology used in these experiments is similar to the topology proposed in Section 3.3.3 and contains a reasonable amount of packet losses and re-routing traffic with cascaded bottleneck. All these specifications emulate the real scenario of highly congested network. The effect of the proposed congestion avoidance can clearly be observed from the throughput comparisons shown in Figure 4.4 below.

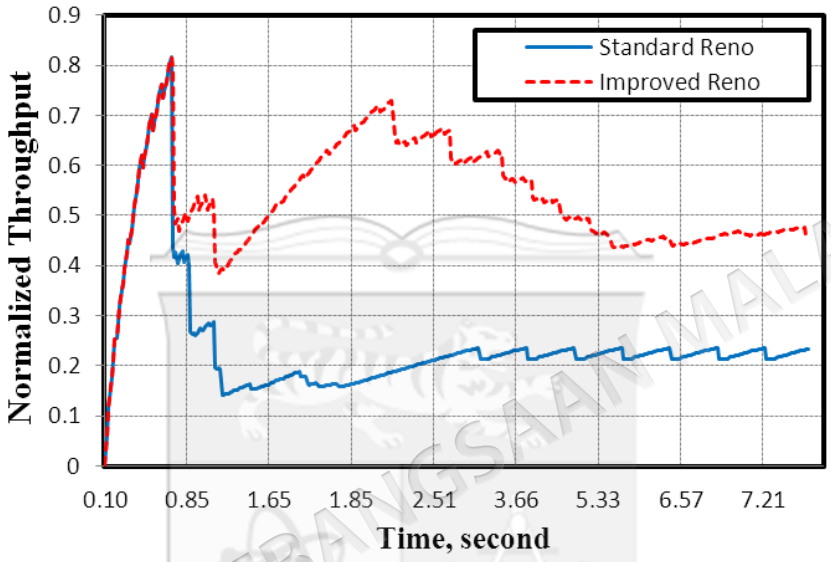


Figure 4.4 Throughput comparison of standard Reno and improved Reno when k=4

The rational throughput comparisons for the standard and improved Reno were estimated when k=4. Initially, both Reno had kept the same throughput, but when the network connection suffers from congestion, the improved Reno delivered better throughput. This is because the extra flows improved Reno throughput, while the proposed flows number increased the number of transmission rate over the same link and the standard Reno remained the normal throughput. At the end of the experiment, the throughput of the improved Reno degraded since the number of flows used in this experiment did not fulfil the network requirements. Hence, some packets would be lost, and this particularly happened when the flow number was of high values, such as 4. The selection of the k values is still set by user and there is no practical solution to dynamically choose and set k, depending on the network congestion state. In the next section, a practical solution to estimate the optimum number of TCP virtual flows k in a dynamic method corresponding to the network congestion situation will be taken into consideration.

4.3.3 Adaptive Approach to Estimate the Optimum Number of TCP Flows

TCP with cascaded flows give several benefits as compared to single flow TCP. When the end host opens the *k* of TCP flows to the corresponding destination, *cwnd* will increase *k* times faster than the *cwnd* of single flow TCP. Therefore, the achievable throughput of TCP with multiple flows is significantly higher than that gained by TCP with single flow, when the probability of packet loss is kept at the same level. The proposed technique uses the modified *cwnd* behaviour during the increasing and decreasing phases to estimate the value of *k*. Meanwhile, in the slow-start phase, *cwnd* increases by one segment for each ACK received until congestion is detected or when *cwnd* reaches *ssthresh* as shown in Equation (4.30).

$$cwnd_{t+\tau} = cwnd_{t} + 1 \tag{4.30}$$

where $cwnd_{t+\tau}$ denotes *cwnd* after one ACK which takes τ period.

Conversely, if *k* TCP flows are permanent in the slow-start stage, *cwnd* will then increase completely and becomes *k* times faster than single TCP flow. Therefore, for the same set of TCP flows, the congestion window for each flow, $cwnd^{j}$ must increase by 1/*k* for each ACK, as shown in Equation (4.31).

$$cwnd^{j}_{t+\tau} = cwnd^{j}_{t} + \frac{1}{k} \tag{4.31}$$

The sum of the *cwnd* size at time *t* for each TCP flow can then be estimated as follows:

$$cwnd_{t} = \sum_{j=1}^{k} cwnd^{j}_{t} \tag{4.32}$$

Moreover, when each flow with set *k* increases *cwnd* by one segment, the sum of increasing becomes *k* times of the previous *cwnd*.

In order to match the increment in the *cwnd* of *k* TCP flows with single flow, the received *cwnd* will become:

$$cwnd \quad = \sum_{i=0}^{k-1} cwnd \quad + i \tag{4.33}$$

To adjust the fairness for all the flows of TCP, the distribution of unduplicated ACKs is required for all the flows. Thus, each TCP flow needs to receive *cwnd/k* before increasing its *cwnd* by one segment. Therefore, each flow included in each set of *k* flows should increase its *cwnd* by one segment after receiving a number of unduplicated ACKs which are equivalent to:

$$\frac{\sum_{i=0}^{k-1} cwnd \quad * \left(\sum_{j=1}^{k} cwnd \quad ^{j} + i \right)}{k} \tag{4.34}$$

In the decrease mode, the single flow TCP reduces its *cwnd* by half if congestion is detected, whereas in the multipath TCP, only one flow halves its *cwnd* for each occurrence of congestion.

Assuming that the total *cwnd* becomes half of the previous window size for *k* flows after the congestion, the total congestion window will then by decreased by amount, according to the formula shown below:

$$\sum_{j=1}^{k} cwnd \quad _{t+\tau}^{j} = \frac{1}{2} \sum_{j=1}^{k} cwnd \quad _{t}^{j} \tag{4.35}$$

As a result, the decrease in the *cwnd* of *k* flows is equivalent to (2*k*-1)/2*k* of the current size of *cwnd*, and the final equation is formulated to be as follows:

$$cwnd \quad _{t+\tau} = \frac{2k-1}{2k} cwnd \quad _{t} \tag{4.36}$$

In order to obtain the value of *k* according to the current and previous sizes of the congestion window, Equation (4.36) can be rewritten as:

$$k = \frac{1}{2}\left[\frac{cwnd_{\ t}}{cwnd_{\ t} - cwnd_{\ t+\tau}}\right] \tag{4.37}$$

Equation (4.37) estimates the number of connections for any parallel TCP in a simple mechanism according to the loss events. Additionally, this formula estimates the value of *k*, regardless to the previous value of *k*. This mechanism only depends on the status of the congestion window for the previous and the current *cwnd* sizes. The implementation of this relationship over the real parameters of *cwnd* shows the smooth increase and decrease in the number of TCP flows.

For example, if the previous *cwnd* size is estimated to be 30 packets and the current *cwnd* is 20 packets, this means the expected number of TCP flows will be 2.5. In special case, when the current *cwnd* becomes equal to the half of the previous size, *k* will be then equivalent to one and that denote that the connection suffer from high congestion. Therefore, it is necessary to set the number of flows to one in order to avoid packet dropping.

The performance evaluation of the proposed mechanism of dynamic estimation to the flows number is based on the congestion window and the throughput. Figure 4.5 shows the congestion window behaviour of the TCP Reno using the proposed mechanism with three options. The first option when the number of Reno flow is set to 2, followed by setting it to 4, while the third option is based on using the dynamic estimation to the number of virtual flows.

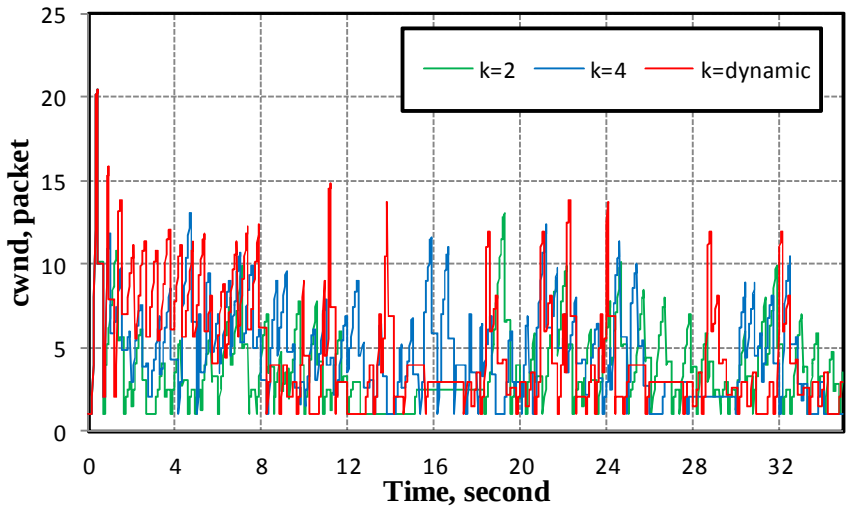
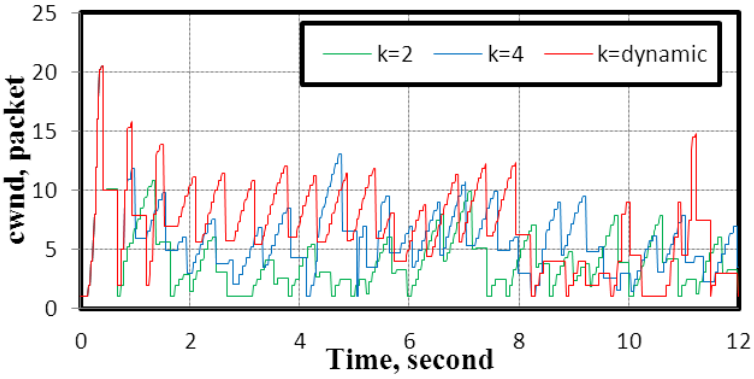


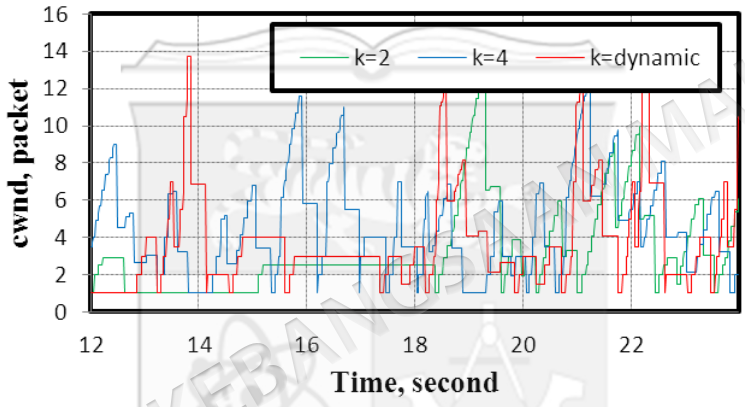
Figure 4.5 Behaviour of *cwnd* of TCP Reno with multiple flows when *k*=2,4, and dynamic estimation

When the number of flows is set to 4, it gives a better performance than when it is set two flows, while the congestion window reaches a reasonable size. However, it seemed to be unstable when the number of flows was set to be equivalent to two. This could be due to the fact that when *k* was set to a constant value as it caused the congestion control to be unable to recognize the high congestion and to continue sending high packet rate, and even the network pipe would be suffering as well.

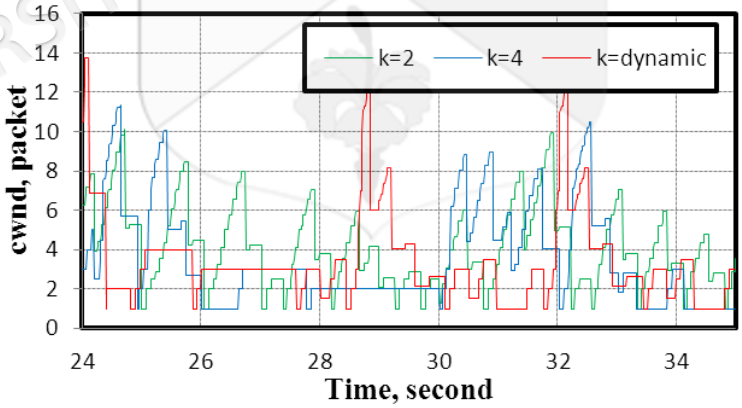
However, when *k* was estimated according to Equation (4.37) the number of flows would be controlled, depending on the variations in *cwnd*. When the connection begins to enter the congestion mode and senses a packet loss, the modified congestion avoidance mechanism will decrease the number of flows to avoid more losses, and then return to increase the number of flows when it senses that there is no more loss. Figure 4.5 does not clearly illustrate the behaviour of the congestion window, so the enlarged graph is separated into three parts, as shown in Figure 4.6.



(a)



(b)



(c)

Figure 4.6 Enlarged *cwnd* of TCP Reno with multiple flows when *k*=2,4, and dynamic estimation
(a) 0-12 seconds (b) 12-24 seconds (c) 24-35 seconds

The effect of the amount of packet loss for the three options could not be clearly observed in monitoring the behaviour of congestion window. Therefore, the throughput measurement was carried out to validate the performance of the proposed mechanism. As shown in Figure 4.7, the throughput comparisons between the three experiments included *k*=2, 4, and dynamic estimation.

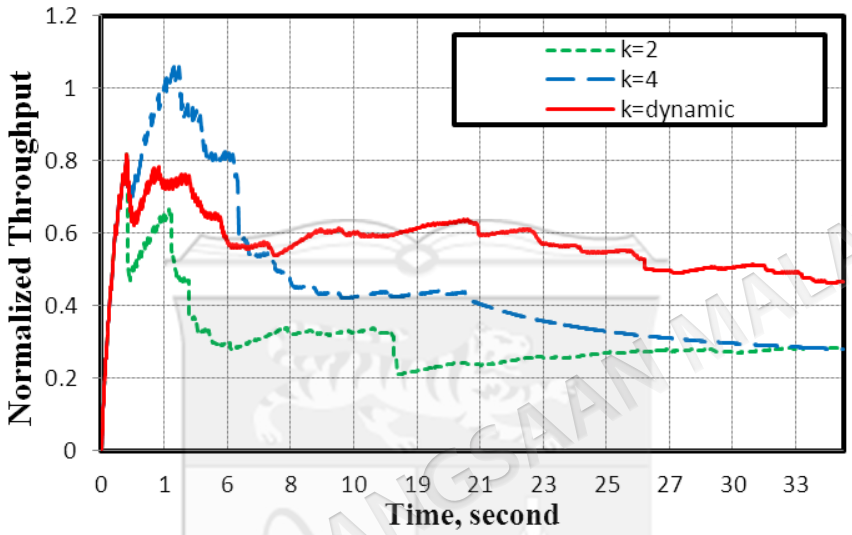


Figure 4.7 Comparative throughput of TCP Reno with multiple flows when using *k*=2,4, and variable dynamically

The comparative throughput demonstrated in Figure 4.7 proves that the variable value of *k* could give better performance. This is because the variations in virtual flows number can efficiently regulate *cwnd* to avoid expand the packet rate out of bandwidth limit. Moreover, in Figure 4.7, the performance of TCP Reno with 4 flows is better in beginning. Since the huge amount of packet transferred from source to destination over four connections. After 5 seconds, the performance is degraded due to the retransmission and recovery routines used to recover the lost packets. On other side, for the adaptive setting, the flows number started with low throughput compared with Reno (*k*=4), but it kept the throughput in good level thought rest of simulation period. The stability in throughput shown in Figure 4.7 resulted from the fluctuations in the number of optimum flows, which estimated depending to the congestion events of the network connections.

The setting of virtual flow connection, either manually by user or dynamically by special algorithms still represents a major challenge to the parallel TCP protocols and for multipath TCP's too. In addition, it is not acceptable to assume the number of virtual flows with one set of connections to be large because of the probability in unfairness between the parallel flows always exists. Despite the weakness of using a modified congestion control, multipath TCP's remains the best choice in high-speed networks when the available bandwidth reaches to hundreds of Mbps.

4.4 SUMMARY

In this chapter, three enhancements to congestion avoidance mechanism are presented to achieve the high performance of congestion avoidance mechanism. These enhancements involved improving the relationship between the parameters pair *a* and *b* of the AIMD algorithm. In addition, the improved AIMD is supported by multiple flows technique, and the number of these flows is controlled using an adaptive method based on detecting the congestion window variations during congestion events. Firstly, the proposed mechanism uses the general AIMD algorithm to controlling the increments in the congestion window for each RTT and when packet loss occurs. The new relationship between the parameters pair *a* and *b* can provide a wide range of desired values for *a* and *b* according to the number of the virtual path assumed by the user. Practically, the setting of the flow number by user can cause a reasonable amount of loss and additional recovering periods. For this reason, the proposed congestion avoidance algorithm adopted a new approach to estimate the optimum virtual flows available over the network. This particular technique gave the algorithm more reliability to run in stable mode over different congestion conditions. Moreover, this technique will also assist in adjusting TCP flows during sense loss events by comparing the current congestion window size with the previous size and then estimate the candidate value of the flows.

CHAPTER V

IMPLEMENTATION OF TCP RAPID AGENT IN NS-2

5.1 INTRODUCTION

Despite its wide employment in wired and wireless networks, the TCP is still affected some practical problems. One of these problems is that, the congestion control mechanism does not permit the data stream to get complete bandwidth from the network links. To solve this problem, many TCP protocols have been proposed. However, until now, there is no final agreement in terms of the high-speed TCP that must be adopted. Therefore, the operators may require employing different TCPs to transmit data over the same-shared network path. The works proposed in Chapter III and Chapter IV have used the NS-2 simulator to examine the performance and to monitor the behaviour of the proposed slow-start and congestion avoidance mechanisms.

For implementing the proposed congestion control mechanism with the current TCP variant, the source code files of the TCP agents in NS-2 have to be modified, by merging the new algorithms within the developed TCP. This chapter presents an enhanced congestion control mechanism, which consists the slow-start phase, congestion avoidance phase, and fast retransmit phase. The slow-start algorithm that had been proposed in Section 3.4 has been further modified to improve its performance. In addition, the enhanced congestion avoidance algorithm that had been introduced in Chapter IV has also been simultaneously involved with the slow-start algorithm to deliver better TCP performance in high-speed networks.

5.2 PROPOSED CONGESTION CONTROL MECHANISM

The implementation of new TCP agent in NS-2 involves the development of new congestion control mechanism or at least the improvement of the slow-start or the congestion avoidance phases, by modifying the parameters that control and adjust the congestion window such as *ssthresh*. As explained in previous chapters, the existing and future generation networks use high-speed links and they mostly experience low propagation delay. Therefore, that is important to deliver a new group of protocols that can give reasonable performance over these quick links. Then, the traditional techniques used in TCP source variants are not be able to perform well over wired-wireless high-speed networks, and the standard algorithm, such as standard slow-start, should quickly increase the *cwnd*. These reasons force the developers to propose new mechanisms with high performance, which will enable to complete the transmission operation in a shorter period of time and with lesser packet loss.

Even though the congestion control enhancement has been mainly achieved by the improvement in either slow-start or congestion avoidance phases, but several improved versions of TCP have also been built on new congestion control by modifying both the phases. The second approach is more efficient and more reliable in the controlling of congestion window, because the slow-start and congestion avoidance practically work together and the operation interference between them controls the congestion window. Thus, the proposed TCP includes a wide modification and enhancement in these key algorithms to achieve high-speed protocol with proficient congestion control.

In Chapter III, it had proposed two slow-start mechanisms, to improve the start-up of congestion window. Both the slow-start mechanisms used an interpolating scheme to force the window to grow faster. The proposed mechanism with quadratic interpolation polynomial had been proposed in Section 3.4 can reach the *ssthresh* with lower numbers of iterations and for this reason it will be chose to implemented with the proposed TCP.

5.2.1 Representation of Slow-Start Mechanism

TCP Tahoe supports the slow-start algorithm, which is the first phase in congestion control. If a packet is transmitted over network links with unidentified environments, it moderately needs the TCP to regulate the accessible network bandwidth to avoid network congestion. The slow-start procedure is used to solve this problem at the opening of transfer, or when the retransmission timer distinguishes packet loss. Through the slow-start, the TCP increases *cwnd* mostly by MSS bytes for each ACK received and that will acknowledge the new packet to be transmitted. The slow-start phase ends when the *cwnd* exceeds *ssthresh* level or when the congestion is noticed. The slow-start algorithm uses *cwnd* as a main parameter, where the TCP sender could not transmit the packets with unacknowledged data into network pipeline, which exceed the *cwnd* size. The slow-start algorithm has two objectives:

1. To determine the *cwnd* value, which is not identified when the new connection is established (Liu et al. 2008).
2. To initialize new packets by using the acknowledged packets as a clock as the TCP is a self-clocking protocol (Jacobson 1988).

In the proposed slow-start, the algorithm is divided into two cascaded stages; the first phase is used to quickly increase the congestion window, while the other phase is used to approach *ssthresh* in small and accurate increments. The deriving and numerical model of the proposed slow-start had been illustrated in details in Section 3.4.2 where the Equation (3.22) represented the proposed slow-start mechanism. The proposed algorithm is a collection of two increment schemes, of which, one uses the duplicating increments if the *cwnd* is less than or equal *ssthresh*/2, while the other increment is used in Lagrange interpolation polynomial. The second stage of the proposed slow-start is used when the first stage is not able to further duplicate the *cwnd*. For a deeper look on proposed slow-start mechanism, that first stage starts when the *cwnd* has initial value and begins increasing if the *cwnd* is less than or equal *ssthresh/2*. However when the duplicating of *cwnd* exceeds the *ssthresh*, the second stage begins to control the increasing limit by using interpolation approach, to approximate the next value of *cwnd*.

In this technique, the *cwnd* is always increased for each ACK received, but the increment step becomes smaller, when the *cwnd* value reaches near *ssthresh*. These two phases are stopped at the point, when the *cwnd* reaches *ssthresh* and then the congestion control enters the congestion avoidance phase. The sequence of the algorithm is as follows:

Subroutine a:

Step 1: Initialization: Set: *cwnd*=1, initial *ssthresh* = *ssthresh* (initial value),

Step 2: Send packets using the window size *cwnd*.

Step 3: for each new received ACK, Go to step 4;

Step 4: if ($cwnd \leq ssthresh/2$), then $cwnd = 2 \times cwnd$ then Go to step 2;

Step 5: if ($ssthresh/2 < cwnd$ && $cwnd < ssthresh$),

then $cwnd = 2 \times cwnd - (cwnd)^2 / ssthresh$ then Go to step 2;

Step 6: if retransmission timeout occurs or receive three DUBACKs,

then *cwnd=ssthresh* and Go to step 2;

Step 7 if ($cwnd > ssthresh$), then enter the congestion avoidance phase (Subroutine c);

It is noteworthy that, when the TCP connection is initiated in slow-start phase, the duplicated increase of *cwnd* begins until it reaches *ssthresh*, and once the *cwnd* becomes larger than *ssthresh*, the TCP switches to congestion avoidance phase. When the TCP senses any loss in any of the segments, this is a robust sign of network congestion; therefore, as a corrective action, the TCP reduces its segment flow by decreasing the *cwnd*, then it goes back to the slow-start mode again. When three DUBACKs are received within this phase, the TCP changes to the fast retransmit mode, without waiting for the retransmit timeout.

For example, let *ssthresh* be equal to 65535 bytes as initial value (this is default value) and *cwnd* be equal to 1460 bytes, which represents the segment size and means *cwnd=1*. If *cwnd* < *ssthresh*, the TCP goes to slow-start mode and *cwnd* increases from 1460 bytes to 23360 bytes in first increasing stage, furthermore it has been presumed that there is no loss in segments within slow-start. Through the slow-start, the *cwnd* increases by *2cwnd* segment for each successful acknowledgment received.

As soon as the *cwnd* reaches 2 x 23360 > 32767, the slow-start algorithm changes to the second stage by using other formula, to increase the *cwnd* based on Lagrange polynomial interpolation ($2cwnd–cwnd^2/ssthresh$), to efficiently increase the *cwnd* until it becomes equal to the *ssthresh*. After that, the TCP enters the congestion avoidance phase.

As mentioned in chapter III, the proposed slow-start mechanism comprises a lot of advantages due to the fast growth of *cwnd* such as, reducing the number of segment losses and also minimizing the burst traffic over the network path. In addition, the fast performance of the proposed algorithm will help to delay the expected congestion time.

5.2.2 Representation of Congestion Avoidance Mechanism

The congestion control mechanism of the new TCP includes the congestion avoidance algorithm as introduced in Chapter IV. This algorithm consists of three stages, the first stage is based on general AIMD algorithm to adjust the increment/decrement of the congestion window for each RTT or when the packet is lost. The relation between *a* and *b* of general AIMD is formulated to obtain high performance of these algorithm. The relationship of the parameters pair *a*,*b* is enhanced by deriving new relationship between them that can provide extra virtual paths over the same TCP connection. In addition, the new relationship gives a wide range of desired values for *a* and *b* according to the number of virtual path assumed by the user. The relationship between *a* and *b* is based on the number of TCP virtual flows, but these flows are not always estimated dynamically, but they are set by users.

The third stage in enhanced congestion avoidance is focused to design and implement new technique to estimate the number of flows that are available in the network pipe, to get maximum data transfer from the source to the destination using *k*-TCP flows. The number of flows *k* depends on the network conditions and the congestion state. When these conditions are set to 2, 3, 4, or more, it is similar as using multiple TCP connections over same link.

To arrange the final congestion avoidance algorithm, and before implementing AIMD, it is necessary to estimate the optimum number of virtual TCP flows by using Equation (4.37) according to the congestion window size for the current and the previous size. The value of *k* is determined when detecting the congestion event or exactly after each packet losses. The steps of *k* estimation algorithm are as follows:

Subroutine b:

Initialization, k=1 and f=0;

Step 1: When packet loss occurs, enter *k* estimation routine, Go to Step 2.

Step 2: k=0.5 (f/(f-*cwnd*)) ;

Step 3: keep the last value of *cwnd*, f=*cwnd* ;

Step 4: Stop *k* estimation routine and Go to congestion avoidance (Subroutine c);

where *k* is the number of virtual flows and *f* represents a temporary variable to store the last window size after last packet lost.

After estimating the number of TCP virtual flows *k*, and when $a=k$, then it can determine the value of *b* based to Equation (4.27). The implementation of AIMD (*a*, *b*) to control congestion window increments is based on the estimation of the virtualized flows and the relationship between parameters pair *a* and *b*. Now, when $a=k$, the congestion window increases by *k/cwnd* per RTT and the total size of *cwnd* is estimated according to Equation (4.28). If loss occurs, the congestion is window adjusted by the *b* parameter, where *b* calculated from the improved relationship between *a* and *b* and limited by the number of virtual flows *k* as expressed in Equation (4.29).

Corresponding to the last two formulae, *k* estimated with each packet loss to control the congestion window size and if there is no loss, the congestion window is continually increased by *k/cwnd* for each RTT. The construction of the congestion avoidance algorithm includes the following procedures:

Subroutine c:

Step 1: For each RTT, *cwnd* = *cwnd*+ (*k*/*cwnd*) ; Go to step 2;

Step 2: If loss occur, *cwnd* = *cwnd* – (2/(3*k*+1)) *cwnd* ; Go to Step 3;

Step 3: Estimate new value of *k* ; Go to step 1;

It is important to note that, the estimation of *k* should be done after the occurrence of loss, to avoid illegal values of *k* that may cause minus values of *k*, however it does not matter if *k* is not an integer.

5.2.3 Representation of Integrated Congestion Control Algorithm

The integration among the proposed slow-start and proposed congestion avoidance, and the estimation of the optimum number of virtual flows represent the congestion control mechanism, which the generated TCP will support. This TCP includes a quick start-up to the window and can provide multiple flows as a virtual connection between the sender and the receiver, over the same connection, in order to achieve high throughput. The proposed TCP includes many features in order to rapidly transmit packets over network links. Therefore, the proposed TCP would be a significant candidate to be termed as TCP Rapid. In fact, the proposed TCP and the congestion control algorithm are implemented over TCP Reno, to gain the benefits of the fast recovery and fast retransmission algorithms from the Reno congestion control. The TCP Reno needs to get instant ACK, when a segment is received by the destination. This is because, when the source receives Duplicated ACK (DUPACK), and this DUPACK might have been possibly received, if the succeeding segment has been delayed in the network pipe and the segments might have reached out of sequence or otherwise the loss in packet is happened.

When the TCP Reno source receives number of DUPACKs, it takes required time and even though the segment had taken an extended route, it must reach the destination. As discussed earlier, in wireless and in high-speed networks, there is a great probability of packet losses to happen. Thus, TCP Reno proposes the fast retransmission algorithm, and when the sender receives three DUPACK's the transmitted segment that was lost must be transmitted without waiting until the timeout. Another reason to choose TCP Reno is that, after a segment is lost, the Reno does not decrease the *cwnd* to one segment unlike the Tahoe, because the decreasing makes the network pipeline empty.

The state transition diagram of proposed congestion control mechanism shown in Figure 5.1 consists of many phases, but all these phases represent an integrated congestion control. Every time, when three DUPACK's are received, it means that the segment has been already lost and the algorithm will retransmit that segment again and then enter the fast recovery mode. Also, it sets *ssthresh* to the half the current size of the *cwnd* and sets *cwnd* to become *cwnd-cwnd(2/(3k+1))*, where *k* is already estimated before. On other hand, for every DUPACK received, the *cwnd* is increased by *k/cwnd* and when the increasing of *cwnd* exceeds the amount of segments in the network pipeline then the transmitting of new segment will be delayed.

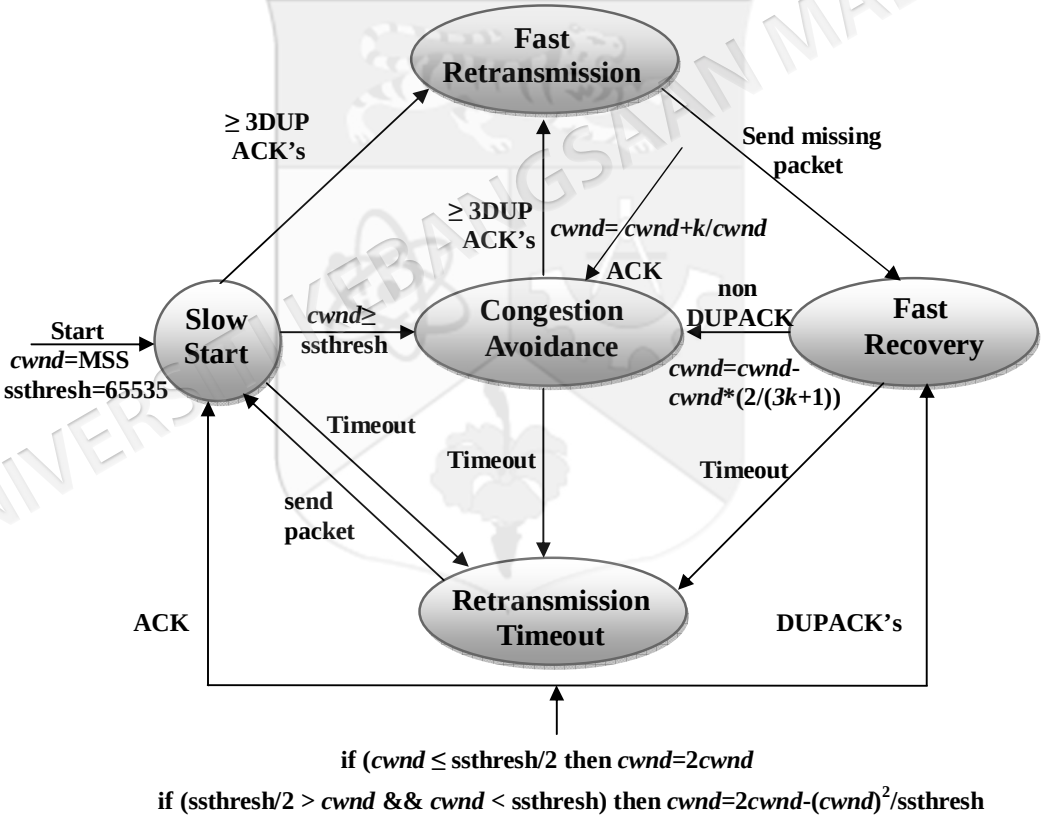


Figure 5.1 State transition diagram of proposed congestion control mechanism

5.3 MODELLING OF TCP RAPID IN NS-2

The proposed congestion control mechanism can be implemented with many source TCP variants such as, Tahoe, Reno, Newreno, and Fack, but it requires many modifications to get the final protocol. On other hand, the modified TCP should use the same name of the agent. For these reasons, the generation of new TCP over NS-2 platform represents a suitable solution to monitor the performance of the proposed congestion control with separated TCP agent. In fact, the generation of new TCP module in NS-2 remains a major challenge to NS-2 users and developers due to the complicated procedures used to achieve that. These procedures include adding and developing some source code files, generating header files, identifying functions and defining new variables.

5.3.1 Architecture of Network Simulator NS-2

The network simulator NS-2 (1989) is a freely accessible object-oriented and discrete-event network simulator, which provides a structure for constructing a network prototypical. The NS-2 identifies data as input parameters, analysing data output and giving outcomes and results. The two main reasons for the wide impact of NS-2 are; it free and open source and fits many researchers in laboratories and universities and huge range of network modules and objects can implement NS-2 (Olsén, 2003). NS-2 is written using C++ programming language, with an Object-Oriented Tool Command Language (OTcl). The OTcl is used to provide user interfacing, which permits the specified input of the model (Tcl scripts) to be executed. Mostly, the elements of any network topology in NS-2 are established as classes in object-oriented style.

For TCP modelling, NS-2 offers significant support for TCP simulating, modelling, queuing algorithms, routing algorithms, and multicast protocols. Modelling of TCP in NS-2 was initially based on source code of the Berkeley Software Distribution (BSD) kernel in 1990's (Wei & Cao 2006). Later, TCP modules in NS-2 have extremely assisted the research teams and groups to evaluate and investigate the behaviour of TCP.

NS-2 has two categories of TCP; the first category is a one way TCP, where it uses objects with several classes on sender and receiver sides. In the TCP sender side, some available classes are provided for TCP Tahoe, Newreno, Reno, Vegas, Sack, and Fack. While in the receiver side, the classes that are available for TCP are without delayed acknowledgements and with selective acknowledgements. Furthermore, other subclasses can be derived from these classes to apply the required modifications, to the standard congestion control mechanisms. The second category includes two-way TCP, where the TCP uses objects with same class on sender and receiver sides. As a matter of fact, the TCP's with one-way are more commonly used than the two-way TCP's. Figure 5.2 demonstrates an example of object hierarchy in C++ and OTcl. NS-2 design uses a model named shared object design, which means that the NS-2 architecture is based on programming in two languages.

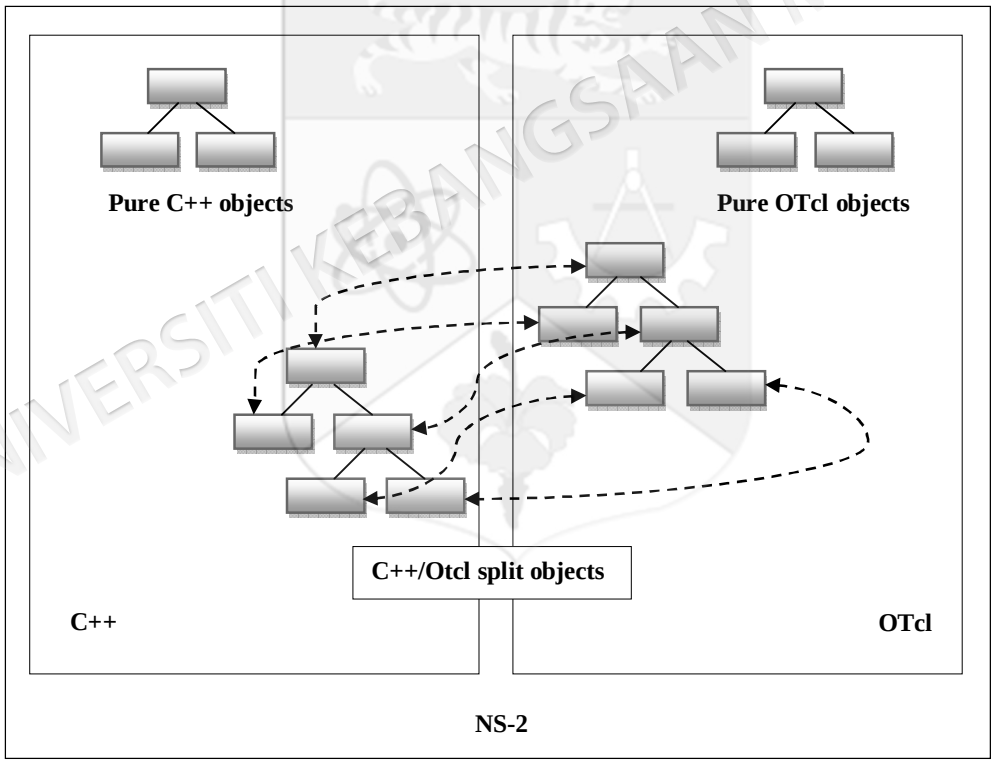


Figure 5.2 Corresponding object hierarchy between C++ and OTcl

Furthermore, with these two languages, there is a corresponding hierarchy of the network objects and the object of each one is open to the other. In addition, there are objects, which are only accessible to one portion of the system just for efficiency purpose.

NS-2 uses C++ to write and compile the network components in the data path, to reduce the packet and the required time for processing. The objects compiled by the NS-2 system are made existing to the OTcl interpreter over an OTcl linkage. This linkage generates an equivalent OTcl objects for each C++ objects and creates the configurable variables that have been identified by C++ objects, to effect as an associated variables and functions to the corresponding OTcl objects. In this manner, the controlling of C++ objects is agreed with OTcl and that makes it is possible to change the linked variables of C++ from a script of Tcl. Besides, it is possible to add variables and the functions as a C++ is linked with OTcl object. Certainly, there is no need to control some of the C++ objects during the simulation, or internally used by other objects, which are not required to be connected with OTcl. Figure 5.3 illustrates the common construction of NS-2 (Wang 2004).

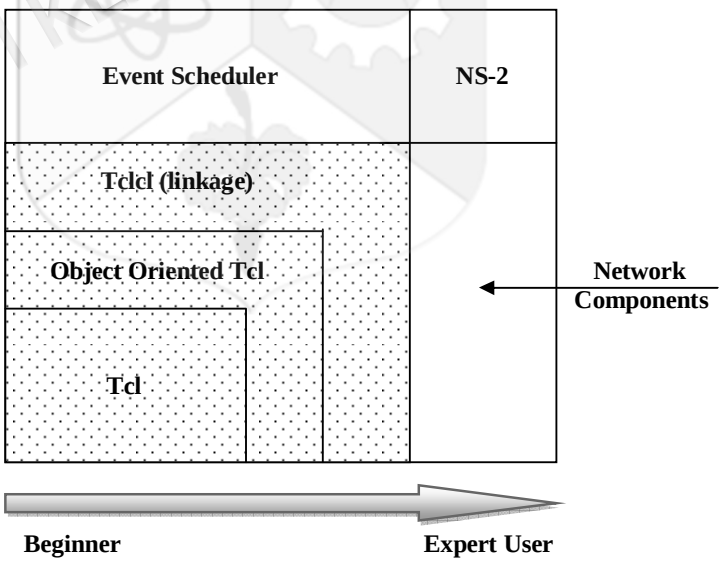


Figure 5.3 Architectural structure of NS-2

In Figure 5.3, the user (not the developer of NS-2) can be supposed standing in left bottom corner while designing and execution the simulations in Tcl by using the simulator objects in the library of OTcl. Then move from that point to the right top corner, which will give more understanding and knowledge of NS-2. The event scheduler and many network components are executed by using C++ language and existing to OTcl over an OTcl linkage, which is applied using Tcl with classes (tclcl) as a Tcl/C++ interface.

5.3.2 Implementation of TCP Rapid Agent

Many TCP protocols implementations are already available in NS-2 such as *Agent/TCP/Reno, Agent/TCP/Newreno, Agent/TCP/Sack1, Agent/TCP/Vegas, Agent/TCP/Fack, and Agent/TCP* (TCP Tahoe). The main goal here is to generate new TCP agent for TCP Rapid, to make it to be identified by NS-2 components. Furthermore, it has to merge the proposed congestion control mechanism with this new agent.

In this research, NS-2.32 is installed over Microsoft Windows XP Professional Service Pack 3 using Cygwin, where Cygwin provides a Linux-like environment under Microsoft Windows. As shown in Figure 5.4, the modelling of new module in NS-2 should be constructed in C++ class and OTcl class (Wang 2004).

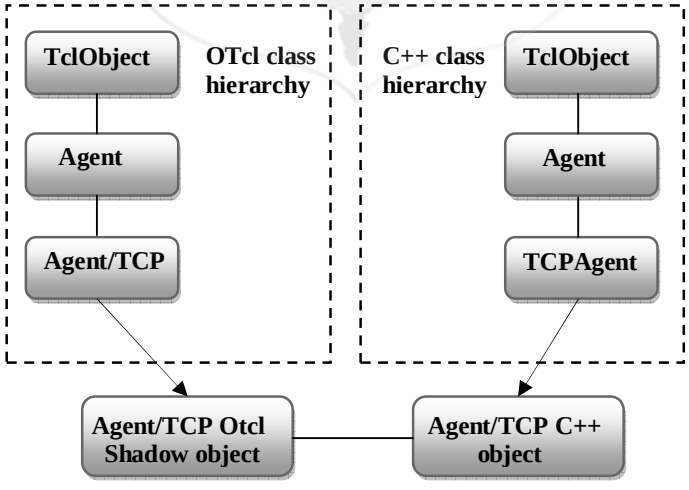


Figure 5.4 TclObject: Hierarchy and shadowing

The class TclObject represents the base class of the most of the other classes in the compiled and interpreted hierarchies. The users from the interpreter generate all the objects in the class TclObject. On other side, the equivalent shadow object is generated in the compiled side of hierarchy and these two objects are mutually accompanied with each other. The other important class is called as TclClass , which represents a pure virtual class in NS-2 system.

Figure 5.5 shows the structure of TclClass, which consists of two categories. The first category generates the interpreted class hierarchy, to reflect the compiled class hierarchy, while the second category delivers methods to generate new TclObjects. Thus, every derived class should be associated with a specific compiled class in the compiled class.

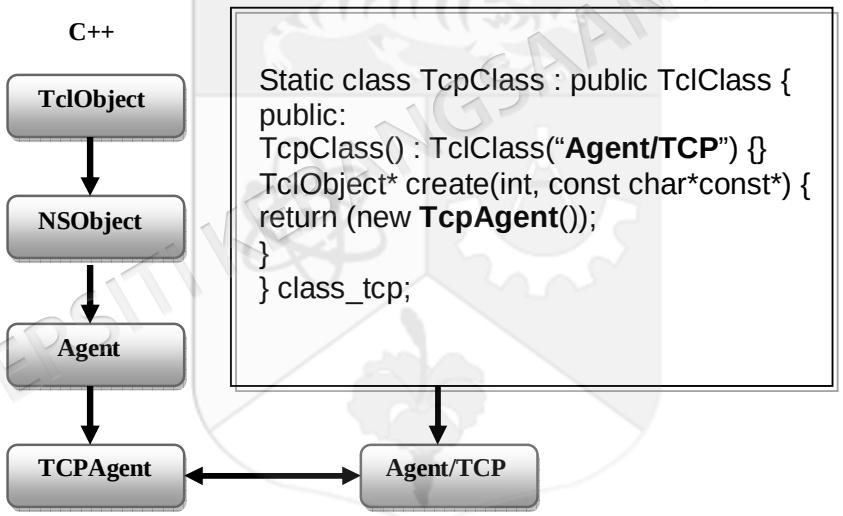


Figure 5.5 An example of TclCalss

TCP Rapid is built with the same concept of TCP Reno, with some modifications in the congestion control. Therefore, the modelling and implementation of TCP Rapid will use the source files of Reno, modify each file separately, and use the modified files to generate TCP Rapid as independent TCP agent. In TCP Rapid, the class becomes RapidTcpClass, where it is derived from TclClass and is associated with the class RapidTcpAgent.

The compiled class hierarchy of RapidTcpAgent is derived from TcpAgent and that in turn derived from TclObject. RapidTcpClass is defined as:

```
static class RapidTcpClass: public TclClass {
public:
RapidTcpClass() : TclClass("Agent/TCP/Rapid") {}
TclObject* create(int argc, const char*const* argv) {
return (new RapidTcpAgent());
}
  } class_rapid;
```

NS-2 will execute the constructor of RapidTcpClass for the static variable class_rapid, the constructor states the interpreted class clearly as Agent/TCP/Rapid, and this identifies the interpreted class hierarchy. The convention in NS-2 is to use the slash character "/" as a separator and for any assumed class A/B/C, it will represent a subclass of A/B, where that is a subclass of A, and A itself is a subclass of TclObject. In the case of RapidTcpClass, the constructer of TclClass builds three classes such as, *Agent/TCP/Rapid* subclass of *Agent/TCP* subclass of *Agent* subclass of TclObject. The generated classes are associated with the class RapidTcpAgent and it generates new objects in the associated class.

5.3.3 Required Modifications in NS-2 Source Files

To ensure the compatibility of TCP Rapid as a new protocol over NS-2 platform, many source files are required to generate or modify the TCP Rapid to make it recognizable and ready to merge with the proposed congestion control algorithm. Unfortunately, the procedures to add TCP Rapid in NS-2 files are very tedious and complex, because there is no complete documentation for these routines. The other risk is that NS-2 does not help in compiling operation (all required modifications are done using C++). Thus, when the developers encounter errors, all the performed steps have to be revised and debugged.

As the modified files involved here are based on releases 2.3x of NS, the proposed TCP have to be tested over NS-2.3x series. The modification steps can be illustrated as following (Appendix A includes brief info about the modified files):

1. The first modification applied on **ns-compact.tcl** file in the location */ns-allinone-2.3x/ns-2.3x/tcl/lib/* by adding the single line as shown below:

```
$self map_ns_defaults ns_rapidtcp
```

Then, adding the statements below in the correct place in same file:

```
# Agent/TCP/Rapid
TclObject set varMap_(rampdown) rampdown_
TclObject set varMap_(ss-div4) ss-div4_
```

In addition, it is necessary to add the couple of statements shown below:

```
set classMap_(tcp-rapid) Agent/TCP/Rapid
set classMap_(rapidtcp) Agent/TCP/Rapid
```

2. Then, the file **Makefile.in** that located in: */ns-allinone-2.3x/ns-2.3x/Makefile.in* needs to add a single statement in the same groups of other TCP variants as expressed below:

```
tcp/tcp-rapid.o
```

3. Other single line should be added to the file **ns_tcl.cc** located in: */ns-allinone-2.3x/ns-2.3x/gen/ns_tcl.cc* as following:

```
$self map_ns_defaults ns_rapidtcp\n\
```

4. Two short statements should be added to the file **FILES** in the main NS folder *ns-2.3x* as stated below:

```
tcp/tcp-rapid.cc
tcp/tcp-rapid.h
```

Additionally, in the same file, the next four lines must be added as shown below:

```
tcl/test/test-output-tcpVariants/fourdrops_rapid.gz
tcl/test/test-output-tcpVariants/onedrop_rapid.gz
tcl/test/test-output-tcpVariants/threedrops_rapid.gz
tcl/test/test-output-tcpVariants/twodrops_rapid.gz
```

5. In the location: */ns-allinone-2.3x/ns-2.3x/tcp* the file **tcp-fs.h** requires to involve the identification routine of TCP Rapid as expressed below:

```
/* TCP-FS with Rapid */

class RapidTcpFsAgent : public RapidTcpAgent, public
TcpFsAgent {
public:
RapidTcpFsAgent() : RapidTcpAgent(), TcpFsAgent() {}

      /* helper functions */

virtual void output_helper(Packet* pkt)
{TcpFsAgent::output_helper(pkt);}
virtual void recv_helper(Packet* pkt)
{TcpFsAgent::recv_helper(pkt);}
virtual void send_helper(int maxburst)
{TcpFsAgent::send_helper(maxburst);}
virtual void send_idle_helper()
{TcpFsAgent::send_idle_helper();}
virtual void recv_newack_helper(Packet* pkt)
{TcpFsAgent::recv_newack_helper(pkt);}

virtual void set_rtx_timer()
{TcpFsAgent::set_rtx_timer();}
virtual void cancel_rtx_timer()
{TcpFsAgent::cancel_rtx_timer();}
virtual void
cancel_timers(){TcpFsAgent::cancel_timers();}
virtual void timeout_nonrtx(int tno)
{TcpFsAgent::timeout_nonrtx(tno);}
virtual void timeout_nonrtx_helper(int tno);
};
```

6. The last step is to get a copy of the files **tcp-Reno.cc** and **tcp-Reno.h** and rename these two files, to become **tcp-Rapid.cc** and **tcp-Rapid.h** respectively. The file **tcp-Rapid.h** characterizes the header file that defines the routing agent and all necessary timers, which performs the functionality of TCP Rapid protocol. On other hand, the file **tcp-Rapid.cc** will execute all timers, Tcl hooks, and routing agent. After the configuration and the validation, the TCP Rapid will be recognized by NS-2. In fact, the new TCP represents an identical TCP from Reno and it carries the same characteristics and similar congestion control mechanism.

Nevertheless, this TCP is ready to modify the standard congestion control based on AIMD (1, ½). In addition, the Rapid interacts positively with the proposed slow-start algorithm installed in **tcp.cc** file, instead of the standard slow-start algorithm. Several commands are used to setup and manipulate the TCP Rapid flows during simulations. These parameters effect on the behaviour of TCP Rapid over different connections as shown below:

```
set tcp [new Agent/TCP/Rapid] ;          # create tcp agent
$ns_ attach-agent $node_(s1) $tcp ;      # bind src to node
$tcp set fid_ 0 ;                        # set flow ID field
set ftp [new Application/FTP] ;          # create ftp traffic
$ftp attach-agent $tcp ;                 # bind ftp traffic
                                         # to tcp agent
set sink [new Agent/TCPSink] ;           # create tcpsink agent

$ns_ attach-agent $node_(k1) $sink ;     # bind sink to node
$sink set fid_ 0 ;                       # set flow ID field
$ns_ connect $ftp $sink ;                # active connection
                                         # src to sink
$ns_ at $start-time "$ftp start" ;       # start ftp flow
```

As a result, currently the TCP Rapid agent is almost similar to the agent of TCP Tahoe, except having fast recovery mode. The number of DUPACKs inflates the current congestion window that the TCP source has received, before receiving the next ACK. A next ACK denotes to any ACK with a rate greater than the highest seen so far. Furthermore, the TCP Rapid agent does not return to slow-start mode during a fast retransmit. Rather, it sets *cwnd* to half of the current size and sets *ssthresh* to match this value.

The next section will explain how to improve the TCP Rapid congestion control by merging the improved AIMD mechanism and by replacing the standard slow-start algorithm with the enhanced slow-start.

5.4 INTEGRATING ENHANCED CONGESTION CONTROL IN TCP RAPID

In this stage, the TCP Rapid carries the same specifications of Reno, but it works independently as a new TCP agent. The enhancement of congestion control performance of Rapid is based on the improvement of the slow-start and congestion avoidance mechanisms and keeping the fast recovery and fast retransmit phases, which it has already involved in Rapid, as it comes from Reno.

After including the new slow-start and new congestion avoidance strategies, the TCP Rapid works with new congestion control and the behaviour of *cwnd* is controlled by a novel technique, to control the packets flow in network connection. The merging of new congestion control mechanism is separated into two parts: slow-start algorithm and congestion avoidance algorithm. These two algorithms should be merged in different files and requires different modifications in spite of being executed together. In fact, the merging of slow-start algorithm is so easy as compared with the integration of congestion avoidance, because the slow-start algorithm runs under only one condition, particularly when the *cwnd* is less than *ssthresh* of the network path, whereas, merging of congestion avoidance algorithm is subjected to many factors and conditions.

5.4.1 Integrating Enhanced Slow-Start Algorithm in TCP Rapid

As explained above, in the proposed slow-start strategy, the size of *cwnd* is increased faster by duplicating the current size by two and when the size touches or exceeds *ssthresh*, the algorithm stops duplicating and starts using interpolation to approximate *cwnd* towards *ssthresh*. The exchange operation between standard and proposed algorithms happens in one file called **tcp.cc** that contains the main routines that control the behaviour of the congestion window in slow-start phase. This file is located in */ns-allinone-2.3x/ns-2.3x/tcp* folder and uses C++ format.

Actually, the changes in slow-start algorithm that have been included in **tcp.cc** will not only impact on TCP Rapid, but also the other TCP versions, this is because, the slow-start algorithm represents a common and shared technique used by many TCP agents in NS-2.

The slow-start routine starts by opening the congestion window and then enters the slow-start mode when *cwnd* is less than *ssthresh* then increases the *cwnd* by one segment. Otherwise, when *cwnd* is greater than *ssthresh*, the congestion control goes to congestion avoidance mode. However, in the proposed slow-start scheme, the slow-start condition depends on either *cwnd* to be less than or equal *ssthresh/2* to duplicating *cwnd* size. When congestion control cannot duplicate the *cwnd* anymore (this case happens when the current *cwnd* lays in the region between *ssthresh/2* and *ssthresh*) then a new approach is used to increment *cwnd* based on quadratic interpolating polynomial. The pseudo code of standard and improved slow-start algorithms involved in **tcp.cc** and expressed as follows:

```
/* open up the congestion window */

void TcpAgent::opencwnd()
{

double increment;

/* Standard slow-start

if (cwnd_ < ssthresh_) {

cwnd_ += 1;
     }
*/

/* New slow-start   */

 if (cwnd_ <= ssthresh_/2)

/* duplicating increment of cwnd */

     { cwnd_ = 2*cwnd_ ;
     }

if (cwnd_ > (ssthresh_/2) && cwnd_ < ssthresh_)

/* Quadratic interpolation increment */

     {cwnd_ = 2*cwnd_ - cwnd_*cwnd_/ssthresh_ ;
     }

/* Go to Congestion Avoidance */
else {
```

In this routine, the standard slow-start algorithm is kept in the routine but in passive mode just for explanation and the choice, the command “else” denotes to go congestion avoidance mode, when the size of *cwnd* reaches *ssthresh*.

5.4.2 Integrating Enhanced Congestion Avoidance Algorithm in TCP Rapid

The implementation of congestion avoidance mechanism for TCP Rapid requires modifying the file **tcp.cc**. This file is responsible to control the window size in congestion avoidance phase and to limit the window size after timeout, and when the ACK is received. There is another file named **tcp-rapid.cc,** which controls the fast recovery and fast retransmit of *cwnd*. The file **tcp-rapid.cc** will be kept unchanged, as there is no variation in fast recovery and fast retransmit algorithm, because the TCP Rapid behaves like the Reno during these two phases. In the increasing mode, the congestion control needs to increase the window by *k/cwnd* for each ACK received. Initially, the value of *k* is set to one, which means that the number of virtual flow of TCP Rapid will start with single flow and then the value of *k* will be estimated according to the network congestion condition. To identify the initial value of *k* is it important to set the initial value with the initial value of *cwnd*. Hence, the expected place to identify *k* value is expressed in the following routine in **tcp.cc** as shown below:

```
// in 1-way TCP, syn_ indicates we are modeling
// a SYN exchange at the beginning.  If this is true
// and we are delaying growth, then use an nial
// window of one.  If not, we do whatever initial_window()
// says to do.

void
TcpAgent::set_initial_window()
{
      if (syn_ && delay_growth_)
          {
      cwnd_ = 1.0 ;  /* initial value of cwnd is 1 MSS */
      k_ = 1.0 ;     /* initial value of flows is 1 */

          }

      else

          cwnd_ = initial_window();
}
```

In **tcp.cc** file comprises a separated routine related with the execution of Reno, when the DUPACK is received, where the *cwnd* can be resized according to AIMD (*k*,2/(3*k*+1)). The decreasing approach can be applied here, because there is an indication that some of the segments have already been lost. This routine should set *ssthresh* to the half of the current size of *cwnd* (*sstresh=cwnd/2*) and set the *cwnd* to become *cwnd-cwnd(2/(3k+1))*. The value of *k* also can be estimated here, by estimating the current and previous *cwnd* size or before and after the occurrence of loss. The modified subroutine to determine the congestion window size after the occurrence of loss in addition to estimation of the number of TCP Rapid flows is represented as follows:

```
case 1:
/* TCP Rapid DUPACKs, or after a recent congestion indication */


    /* decrease cwnd by 2/(3k+1) instead of ½ in Reno    */
                cwnd_ = cwnd_ *(1-(2/(3*k_+1))) ;


            /* update ssthresh */
                ssthresh_ = cwnd_/2 ;
            /* Estimation of number of TCP Rapid flows k */

                k_ = 0.5*(f_ /(f_ -cwnd_))  ;

     /* f is temporary variable to store the previous cwnd */
                f_ = cwnd_ ;

break;
```

The implementation of the other part of AIMD (*k*,2/(3*k*+1)) denotes to the increasing phase after congestion occurrence with each RTT. Actually, in **tcp.cc** there are nine algorithms to avoid congestion, and each one uses different techniques to increase the window such as, the algorithm proposed by Floyd & Jacobson (1991) or by using the algorithm of High-Speed TCP.

The increasing formula for TCP Rapid is set in case five and the number of this case is joined with the agent of TCP Rapid. This case can be written in the following simple form:

```
case 5:
        cwnd_ =cwnd_ + (k_/cwnd_);
 break;
```

The three stages in implementation of AIMD (k,2/(3k+1)) as a congestion control algorithm for TCP Rapid is involved in **tcp.cc,** in addition to the other two phases such as, fast recovery and fast retransmit, where these two algorithms are kept in the original **tcp-rapid.cc** and the modifications are carried out in the main file **tcp.cc**.

The final step to consider Rapid in NS-2 is based on configuration and validation, to ensure that, the Rapid becomes compatible and recognizable from the main NS-2 components and elements. This process executed using three cascaded commands: /configure, make, and ./validate-full (in NS-2.34 it should be used ./validate instead of validate-full). Figure 5.6 shows the screenshot of the recently completed validation process.

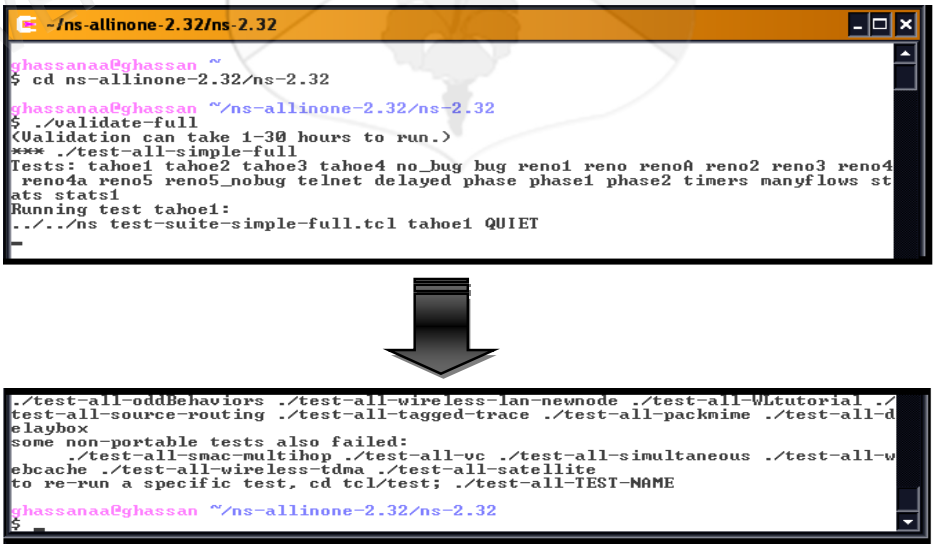


Figure 5.6 Screenshot for NS-2 validation after adding TCP Rapid

Furthermore, a simple script is used to test the functionality of Rapid with simple topology, to prove that Rapid is ready to be experimented over the LTE-Advanced model proposed in next chapter. Figure 5.7 shows the standard congestion window of TCP Rapid over simple network topology.

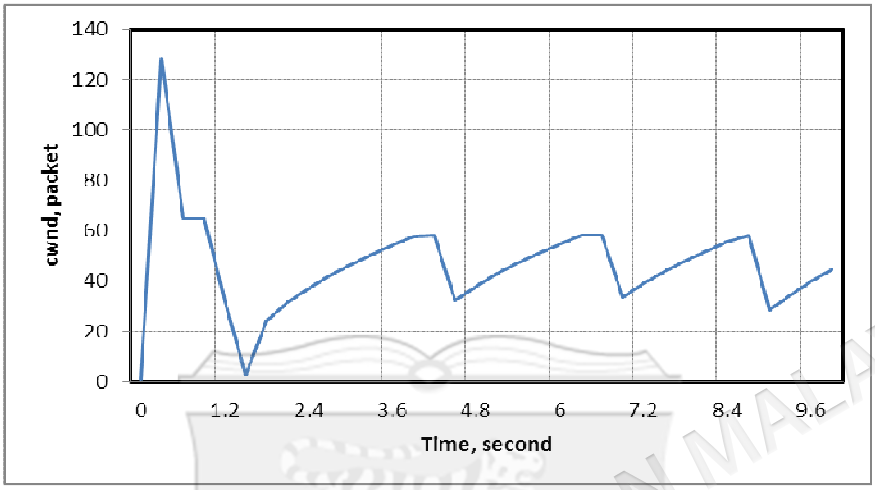


Figure 5.7 The standard congestion window of TCP Rapid

5.5 SUMMARY

In this chapter, a new TCP called Rapid was generated with an enhanced congestion control algorithm. The TCP Rapid carry the general architecture of TCP Reno but it is based on new AIMD mechanism with a novel relationship between the pair of parameters *a* and *b*. Additionally, the relationship between this pairs can give extra number of virtual flows to TCP Rapid instead of single flow in Reno. The other significant facility in Rapid is that, the new congestion control provided an algorithm to dynamically estimate the number of optimal flows and avoid the burden of setting it by user. This chapter had also discussed how to generate the agent of Rapid in NS-2 modeller and how to identify and configure the TCP Rapid over NS-2 platform. The TCP Rapid also included an innovative scheme for the slow-start to assist TCP Rapid to start-up faster over high-speed network and to delay the link congestion point as much as possible. The pseudo code of all files, routines, and subroutines were illustrated in this chapter, in addition to the illustration of the general transition state diagram of the full congestion control mechanism.

CHAPTER VI

TCP RAPID OVER LTE-ADVANCED NETWORK

6.1 INTRODUCTION

The wireless communication networks get advanced periodically, hence, it is crucial to fulfil the associated requirements and expectations of those networks. Therefore, it is essential to have an experimental model, to validate the behaviour of protocols or any other components of these networks, to ensure their performance under different conditions. The modelling and simulation is an effective method to explore the difficulties and resolve the problems. Furthermore, experimental scenarios and settings can be easily and inexpensively simulated using the modelling technique. This section introduces the process of figuring a precise and efficient traffic model of LTE-Advanced network.

6.2 MODELLING AND SIMULATION OF LTE-ADVANCED NETWORK

Many traffic types are held from the eNodeB and the other entities in LTE-Advanced. Each one of these traffics might have different transmissions, security requirements, connectivity, and give specific direction for different network elements. The following are the different types of traffics (Akyildiz et al. 2010):

1. S1-u traffic intended to SGW.
2. S1-c traffic intended to MME.
3. X2-c and X2-u traffics intended to other eNodeBs.

4. Operations Support System (OSS) traffic intended to the core applications that deliver fault, performance management, and configurations.
5. Traffic of the network synchronization.

6.2.1 Model Configuration

The resources of data in the network are accessible while using NS-2 simulator. Therefore, the performance of the network session can be simply investigated. In addition, as the NS-2 source code is opened, therefore, it is appropriate to generate the simulation in any level of the network. All this facilities are employed and installed to simulate LTE-Advanced. In LTE-Advanced modelling, many configuration parameters can be simply changed using TCL such as, change the number of UE and the bandwidth among network components. On other hand, some of other parameters cannot be easily changed within the simulation scenario, due to the limitations in the model implementation, such as the number of eNodeB and aGW (Qiu et al. 2009).

In LTE-Advanced, E-UTRAN consists of eNodeBs, and provides the EUTRAN control plane (RRC) and user plane (PDCP/RLC/MAC/PHY) protocol terminations on the way to the UE. The eNodeBs are connected to each other via X2 interface, and these eNodeBs are interconnected to the EPC via S1 interface. The S1 connection interface supports the relationship among MMEs/S-GWs and the eNodeBs. The EPC is the core of SAE architecture, as it correspondents with the GPRS networks using the S-GW, MME, and P-GW subcomponents (Qiu et al. 2010). The proposed LTE-Advanced simulation model is based on LTE/SAE network model and consists of the following main elements:

1. Server to offer FTP and HTTP signalling services.
2. Two routers point represents an aGW to offer flow control to the data stream.
3. Three eNodeBs as base stations to provide flow control information to UEs.
4. One Relay node to provide extra coverage for five UEs.
5. Thirty UEs connected to eNodeBs and distributed by 10 UEs for each eNodeB.

Figure 6.1 shows the proposed topology of LTE-Advanced traffic model with main elements and links where this topology will be used instead of the topology used in Figure 3.9.

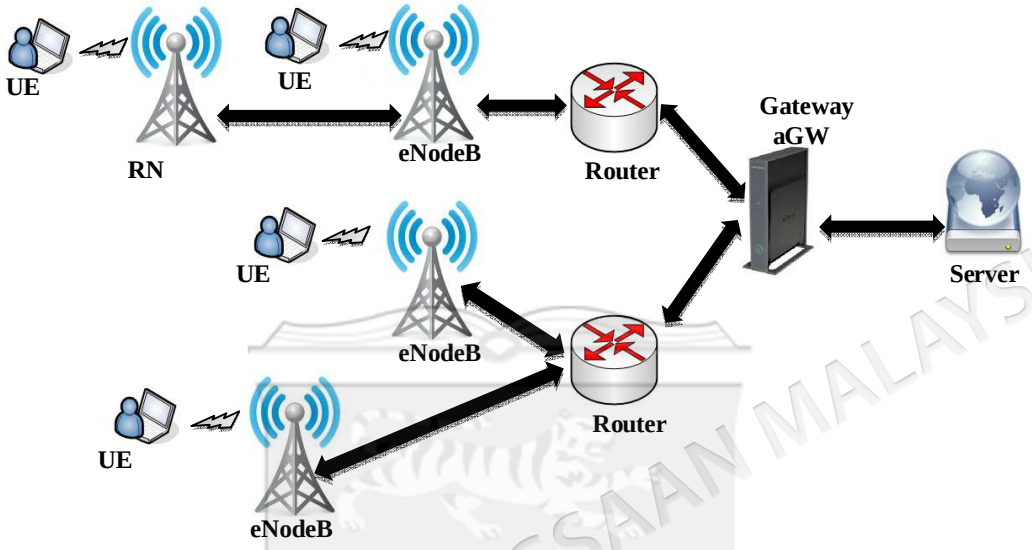


Figure 6.1 Proposed topology of LTE-Advanced

6.2.2 Simulation Parameters

The proposed topology consists of three eNodeBs, which are connected to the access routers, Router 1 and Router 2 with 20 Mbps bandwidth link. The Gateway (aGW) represents the network core, and it is connected to the access routers with large bandwidth link of 1 Gbps and with the server by 100 Mbps. An important scenario is added to this topology by adding relaying node, to provide coverage to five extra UEs nodes. These nodes can only access eNodeB1 via Relay Node, with a 20 Mbps bandwidth link.

Each eNodeB is interconnected with 10 UEs by 2 Mbps bandwidth. Therefore, the total UEs used in this model becomes 35. Furthermore, as the proposed UE have proposed to be wired nodes and this assumption is used to avoid the handover scenario that may happen if one or more UE move from one eNodeB to another.

This assumption is considered to achieve high data streaming between UEs. Due to the use of wireless links, the UEs will not provide high congestion links, which is the major objective of the experiment conducted towards the developed congestion control mechanism. However, full mobility feature of UE will be considered in future study by developing new congestion control mechanism, taking into account the probability of handover over LTE-Advanced.

In the proposed model, each link needs to assign the bandwidth and the propagation delay (latency). In LTE/LTE-Advanced, the networks are expected to have low latency connections, because it is one of the main requirements of LTE-Advanced. In fact, the link latency is a significant performance indicator in wireless communication systems. Practically, in first release of LTE, it provided a radio-link delay less than 5 ms (Astély et al. 2009). The immediate processing of the requirements of RRC and NAS at the eNodeB enhances the latency in LTE-Advanced. Furthermore, it can be also enhanced by decreasing the delay in message processing at different nodes apart from decreasing the duration of Random Access Channel (RACH) and Physical Uplink Control Channel (PUCCH).

The LTE-Advanced, constitutes two types of latencies. The first is the control plane latency, which is related with the connection setup. The second is the user plane latency, which is related with the transferring delay in RAN (Abeta 2010). For best circumstance scenario, the latency of all links in the proposed LTE-Advanced model is set to 3 ms and this value represents the link latency over all the links in the simulated model, to test the expected degradation that may occur by the proposed congestion control mechanism.

In NS-2, every link requires to indicate the queue scheme. The schemes of queue management can be generally distributed into two sets. The first scheme uses the immediate queue size such as, Drop-Tail. The second scheme advocates elements of averaging the queue size such as, Random Error Detection (RED) queue (Patil et al. 2011). The Drop-Tail queuing is the simplest queue management strategy, where it totally drop the inward packets, when the buffer is filled.

For the above mentioned reason, it is suitable to choose Drop-Tail queue to support the access links in this model. The proposed LTE-Advanced model can use several TCP source variants such, as Tahoe, Reno, Newreno, Vegas, Sack, Fack, and TCP Rapid, but only one variant can be used in each scenario. The maximum advertised window size is set to 48 Kbytes and the maximum TCP/IP packet size is set to 1500 bytes (Bajzik et al. 2007), while the maximum TCP's window size is set to different values, to monitor the behaviour in slow-start phase. The Table 6.1 shows the links parameters of the proposed model, including the bandwidth and propagation delay for each link, while Table 6.2 illustrates the other simulation parameters.

Table 6.1 Topology links parameters

Link	**Bandwidth**	**Delay**
Server-aGw	100 Mbps	100 ms
aGW-Router 1	1 Gbps	3 ms
aGW-Router 2	1 Gbps	3 ms
Router 1-eNodeB 1	20 Mbps	3 ms
Router 2-eNodeB 2	20 Mbps	3 ms
Router 2-eNodeB 3	20 Mbps	3 ms
eNodeB1-Relay Node	20 Mbps	3 ms
eNodeB-UE	2 Mbps	3 ms

Table 6.2 Simulation parameters

Parameter	**Value**
Number of UEs with Relay Node	5
Number of UEs with eNodeB	10
Packet size	1500 bytes
Advertised window size	48 Kbytes
Queue scheme	Drop-Tail
Maximum window size	Variable
Simulation time	10 s
TCP protocol	Tahoe, Reno, Newreno, Vegas, Sack, Fack, and Rapid

6.2.3 Validation of Simulation Model

Figure 6.2 shows the screenshot of the real animation of the proposed model using NS-2, where, the green node represents the server that is directly connected to aGW.

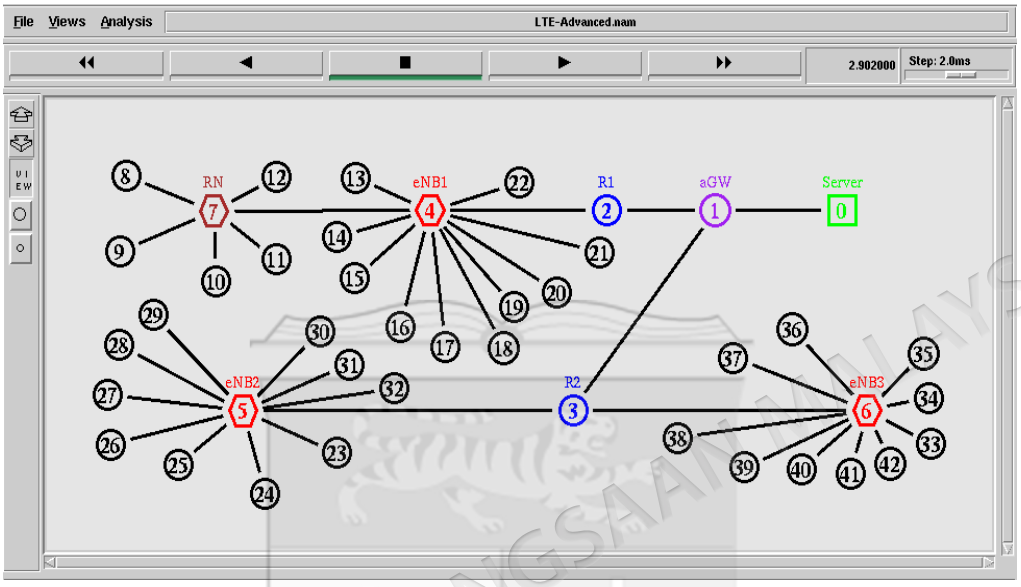


Figure 6.2 Screenshot of LTE-Advanced model animation in NS-2

The two routers R1 and R2 are connected to aGW with 1 Gbps and 3 ms link and these two links represent the bottleneck of the model. The first router is linked with the base station eNB1, while the second router is linked with two base stations such as, eNB2 and eNB3. One of the most important scenarios considered in this model is using the Relaying Node RN, to achieve additional coverage from five separated UEs. Each base station is connected with 10 UEs to obtain 35 UEs, which are connected to these three eNBs. All the UE nodes have similar link parameter of 2 Mbps as bandwidth, and 3 ms as latency. The activity of these UE nodes not always occurs during the simulation, but several nodes have specific roles, but all are used during simulation. The simulation scenario includes many activities and includes variation in connection path used among UEs. Furthermore, the status of the two bottlenecks R1-aGW and R2-aGw are monitored, to indicate the queue size and the packet loss that may happen during the simulation period.

6.2.4 Performance Metrics

The evaluation of the proposed TCP Rapid is based on many measurements and metrics, to indicate, whether the proposed congestion control mechanism fits and be suitable to run efficiently over LTE-Advanced network. Usually, these metrics are used to evaluate and monitor the behaviour of the networks links and protocols over different situations and conditions such as packet loss, handovers, and congestions. The metrics included in the proposed simulation model consists of monitoring congestion window behaviour, throughput, packet loss, queue size management, and goodput. All these metrics are estimated over LTE-Advanced model, except goodput, because goodput estimation uses private network topology, which includes one bottleneck and four TCP senders linked with four TCP receivers. The goodput measurement will be explained in a separate section later.

The most important metric to estimate the network performance is the throughput, where it indicates the real output of the packet transferred over the whole simulation. For any communication network, the throughput represents the average rate of the successful segments or packets rate transferred over a network links. This data rate may be distributed over a physical or logical link, or pass through a certain network node. The throughput is typically estimated in bits per second, but sometimes it is estimated in data packets per second or data packets per time slot. The throughput estimation had been already illustrated in Section 3.3 while Section 4.2, had showed how to calculate the throughput over large bandwidth with the probability of packet losing. In this research, the system throughput or aggregate throughput represents the sum of the segments rates that are transported to all terminals in the network topology.

The queue size at a network bottleneck would affect the performance of TCP protocols, particularly when executed in a single TCP flow in networks. One of the important metrics to validate the performance is to know the queue sizes of bottleneck, because the dynamic behaviour of TCP protocol implementations varies according to bottleneck queue size. This section evaluates and investigates the performance of Rapid and other TCP variants in accordance to the queue length by taking all the packet traces at the sender.

The estimation of queue size over TCP subjects to several factors and limitations, one of these limitations is the expectation of TCP to transmit packets. In addition, there is no control to the flow rate in the slow-start period or to rapidly discover the capacity of bandwidth. Therefore, TCP transmits packets as a burst per RTT intervals at a rate up to the available bandwidth capacity. The estimation of queued packets for each RTT in the slow-start phase is obtained by the following formula (Hirabaru 2006):

$$Queued\_packet = \frac{cwnd * RTT * (R_i - R_0)}{C} \tag{6.1}$$

where *cwnd* is the TCP's window size in packet, C is the bandwidth capacity, R_o is the bottleneck rate, and R_i is a burst rate at which the TCP sender generates the traffic. As the standard TCP's increase the window size almost for each RTT period, the final growth of window size can avoid overflow of the bottleneck queue until it indicates packet loss.

The estimation of packet loss is a challenging process, because it typically occurs in short duration and relatively rare incidences during packet transmission. The behaviour of packet loss has an important impact on the performance of TCP. In addition, the packet loss probability is inferred by the TCP source, and if the expected reaction from the destination host is not received in the interior of specified period. It is well known that, packet loss can have a substantial effect on the performance of a wide range of TCP protocols and the understanding of the characteristics of loss. In addition, the features of loss are also essential elements of TCP throughput modelling. Essentially, the measurement of packet loss should be estimated according to different conditions and variety of parameters. That means the estimation of packets loss must include the factors that may occur in real network, such as changing the queue size and the packet size.

6.3 PERFORMANCE EVALUATION OF TCP RAPID

The congestion control mechanism proposed in previous chapters includes, two phases, slow-start and congestion avoidance. These mechanisms were improved over the base scheme of standard congestion control used by many TCP source variants. Therefore, the performance evaluation of TCP Rapid should be compared with other TCP's that carry the same features. As explained in Chapter II, there are six TCP versions, representing the source variants of the TCP protocol and many of the proposed TCP's are based on these variants.

A lot of enhanced TCPs are available, which work over different applications and wide range of networks, but none of the TCPs can have common protocol to be professionally executed over all. Hence, the evaluation of TCP Rapid is proposed to experiment and compare only with these six variants such as, TCP's are Tahoe, Reno, Newreno, Vegas, Sack, and Fack, to monitor the physical effect of the proposed congestion control over LTE-Advanced network. Furthermore, the metrics used to evaluate the performance of Rapid with these variants are also based on many significant metrics.

6.3.1 Behaviour of TCP Rapid against other TCP Variants in Slow-Start Phase

As shown in Chapter III, the formulation of the proposed slow-start algorithm was built on the concept of fast increment of the congestion window and minimizing the starting up period. The fast growth in *cwnd* can assist to delay the congestion point of the network and can make the window clocking more frequent during shorter period, which permits to send larger number of segments. Figure 6.3 shows the behaviour of TCP Rapid during initialization start-up in slow-start phase compared with six TCP's. As the initial slow-start period is very short, the experiment time did not exceeded 0.5 s after the connection was established. As illustrated in Figure 6.3, the TCP Rapid requires only 0.21 s to reach *ssthresh*, while the other TCPs required longer time to reach *ssthresh* in the same network situation.

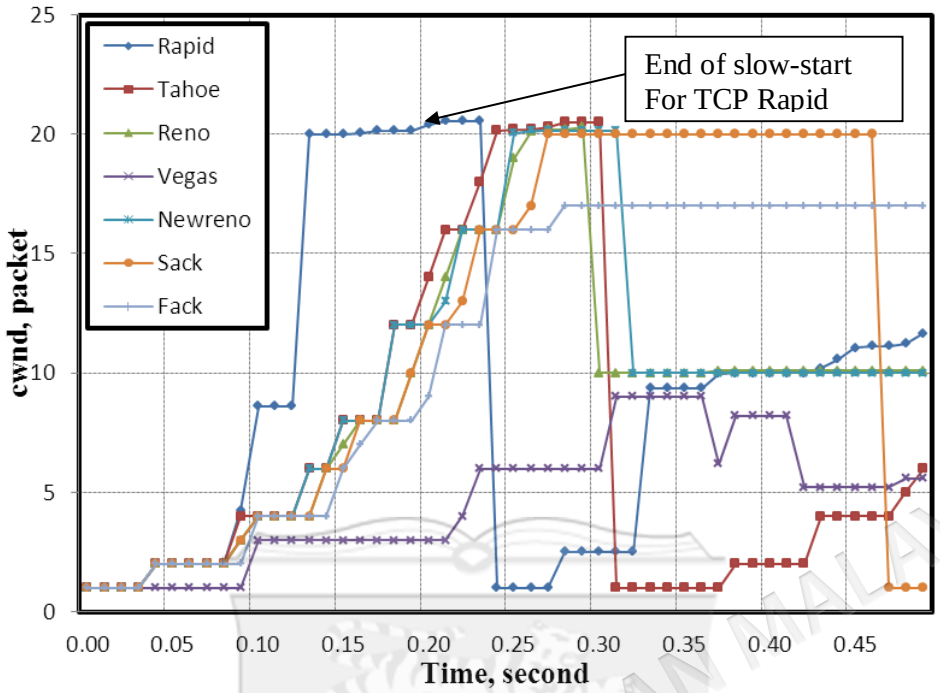


Figure 6.3 Slow-start behaviour comparisons of TCP Rapid against TCP variants

Reno, Tahoe, and Newreno have used the linear and exponential increase in standard slow-start phase; however, due to the effects of other congestion control parameters there are differences in the slow-start time, as shown in Figure 6.3. For instance using the recovery operation after the packet loss by these variants significantly creates the time variation. Therefore, Tahoe and Reno only needed 0.29 s to reach *ssthresh* while Newreno required 0.31 s to reach same window level. On other hand, Sack TCP has arrived *ssthresh* within reasonable time (0.27 s) but it need 0.48 s to detect the congestion, whereas, Fack TCP was unable to reach the maximum window size.

Thus, the Rapid has offered quick start-up to the congestion window and reached the maximum window threshold within a shorter time. The Rapid has reduced the start-up period about 27% as compared with Tahoe and Reno. It is noteworthy that, Vegas, uses other congestion control approach with new approach in slow-start and that is why its experiment has showed low performance in initialization.

The fast start-up in congestion window provided by the Rapid will certainly minimize the handover period in LTE-Advanced network. This because, if the handover occurs between UE and eNodeB, the required time to rebuild the congestion window will be deduced and then the handover period will be confidently minimized.

6.3.2 Congestion Window Behaviour of TCP Rapid against other TCP Variants

In previous section, only the effect of the slow-start mechanism of TCP Rapid was considered, without the enhanced congestion avoidance mechanism. In Chapter IV, the proposed congestion avoidance mechanism had been discussed in details. The main goal is to realize a new congestion window behaviour using AIMD and based on new relationship between the parameters pair *a* and *b*. This new relationship permits the congestion control, to provide different adjustments to *cwnd* in case of increased packet loss. The behaviour of congestion window for Rapid and other TCP's are illustrated in Figure 6.4.

In Figure 6.4, the initial slow-start of Rapid is preceded by the slow-start for other TCP variants and the distinguished difference in enlarged figure shows the fast growth of *cwnd* for Rapid. This difference will continually increase for each new congestion window depending on the simulation time. As expected in the analytical study and in the formulation stage, the window gives fast clocking and regular frequency, in spite of the congested links and the inclusion of re-routing events in the simulation scenario. The regular behaviour of Rapid is not because of the proposed slow-start technique, but because of the novel techniques used in congestion avoidance via the enhanced AIMD or by the estimation of virtual flows number. That allowed estimating a suitable number of available flows to properly send the packets and avoid overflow in network connections.

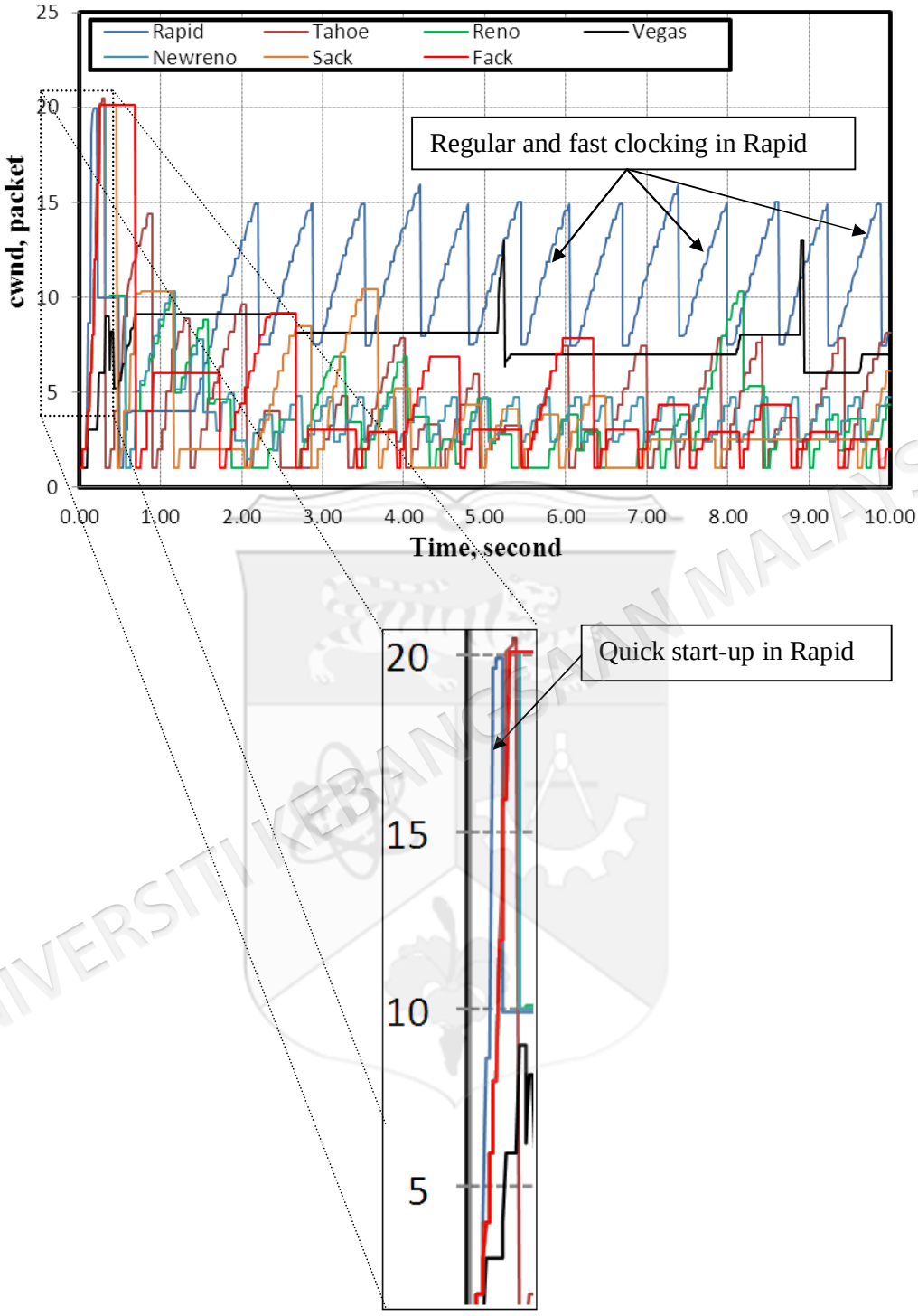


Figure 6.4 Congestion window comparisons of TCP Rapid against other TCP variants with zoom on initial slow-start phase

Therefore, when the congestion occurs due to the large traffics or packet overflow, the congestion control, managed the packet flows by increasing/decreasing the *cwnd* by enhanced AIMD, to avoid the heavy loss in transmitted packets. Essentially, the regular *cwnd* shown in Figure 6.4 can suffer from malformation, when the network topology includes more UEs nodes or when the recent nodes make more complex events such as handovers. Anyway, if comparing *cwnd* behaviour of Rapid with other TCP windows, it is evident that there is a wide difference among other TCP windows and Rapid. Even though the Vegas provide stable window but actually, the Vegas window cannot give high throughput, which is explained in the next section.

6.3.3 Throughput Comparison of TCP Rapid against other TCP Variants

The estimation of the normalized throughput in this experiment is independently measured for each TCP in separate simulation, to avoid the effect of the interference of using many TCP's in the same simulation. Therefore, the same topology model is used to experiment and determine the throughput of the six protocols in addition to the TCP Rapid. The Figure 6.5 shows the comparative normalized throughput of TCP Rapid and other TCP variants.

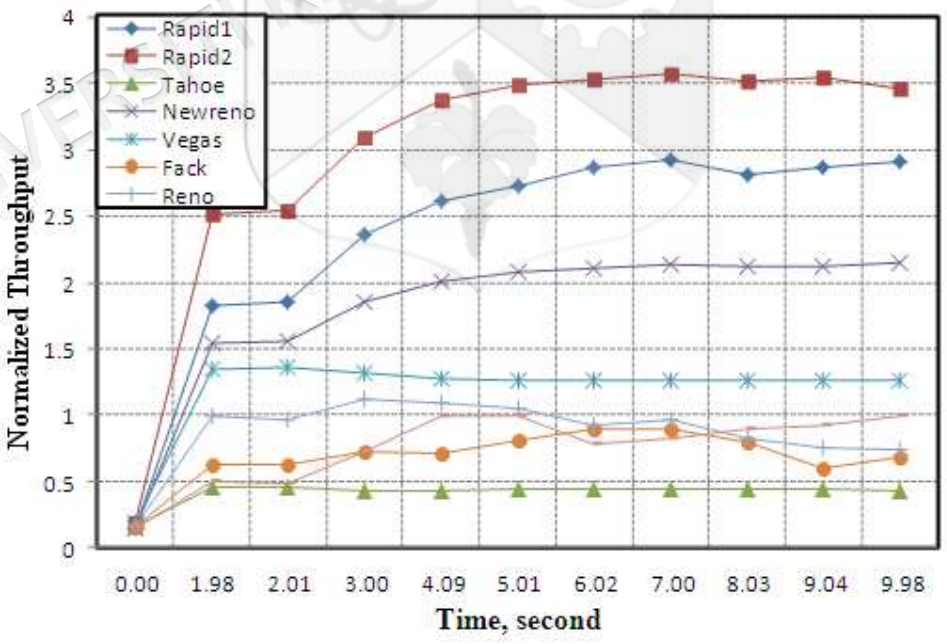


Figure 6.5 Throughput comparisons of TCP Rapid against other TCP variants

In Figure 6.5, two TCP Rapids are used such as, Rapid1 and Rapid2. In fact, Rapid1 only includes the enhanced slow-start algorithm and keeps the standard congestion avoidance used by Reno. However, the Rapid2 includes the two enhanced algorithms: slow-start and congestion avoidance. This methodology is used to distinguish and monitor the effective throughput of the enhanced slow-start mechanism over LTE-Advanced model.

As shown, when the Rapid includes an enhanced slow-start algorithm, the rational throughput has increased from 2.1 in Reno to 2.9 in Rapid. This means that, the average throughput gain has increased by 65% as compared with Reno (from 1 to 2.9), 86% with Tahoe (from 0.4 to 2.9), and 55% with Vegas (from 1.3 to 2.9). On other hand, As Rapid2 has used the enhanced slow-start and congestion avoidance mechanisms with multiple TCP flow, it yields high throughput due to the integration between these two mechanisms, where the improved throughput has reached to 3 folds of the throughput given by Tahoe (0.5 for Tahoe and 3.5 for Rapid).

It has been proved that the effective performance of the enhanced congestion avoidance mechanism has been witnessed in Rapid2 and has ensured the operative techniques used in mechanism enhancements. The three cascaded enhancements such as, the new relationship between the parameters pair *a* and *b*, the multiple flows, and the dynamic estimation to the number of flows, help the Rapid2 to provide higher throughput as compared with other TCP variants.

In spite of short simulation period (10 seconds), the Rapid was able to performance well, due to the usages of multiple virtual flows, to send segments over same connection. As explained in Chapter IV, the multiple flows of Rapid have expected that, the throughput should be *k*-times of standard Reno, where *k* is the number of flows. As a matter of fact, the value of *k* has ranged between 1 and 3 flows only and this range was estimated from monitoring the number of flows during simulation. On average, the Rapid has performed twice better than the standard Reno. This is not only due to the extra virtual flows, but also due to the quick start of the congestion window in slow-start phase.

Even though Reno and Tahoe use similar slow-start algorithm, there is a vast difference in the throughput of, because Tahoe is not able to recover all the packet losses, like Reno. Moreover, Tahoe requires to begin slow-start from initial level *cwnd* (assumed 1 segment in this experiment) while Reno adopts Fast recovery and Fast retransmit. In addition, the congestion avoidance of Reno is considered more intelligent than Tahoe as explained in Chapter II. The Newreno can give better performance, but one of the major weaknesses of Newreno is the degradation of throughput, when the loss rate becomes large. In other words, Newreno cannot perform well during high loss rate, has been considered in this experiment.

Usually, Vegas give stable and steady output but deliver low performance. Since Vegas control the window size according to the current and expected windows level, while all other TCP's (including Rapid) wait the loss event to control *cwnd*. This will degrade the total throughput due to the packet loss used to detect the congestion point of the network connection.

6.3.4 Queue Size Management of TCP Rapid against other TCP Variants

The default queue length proposed in LTE-Advanced model has been set to 12 packets and the queue monitoring has been focused on the link R2-aGw as shown in Figure 6.2. The purpose of using one bottleneck in queue monitor instead of many bottlenecks is because the main goal here is not to test the model, but to evaluate the improved TCP version. Moreover, the queue size is not expected to exceed 12 packets but simultaneously it should be in large enough to avoid congestion in bottleneck as much as possible. The queue management by Rapid and other standard TCP variants is shown in Figure 6.6.

From the Figure 6.6, it is evident that Rapid and Vegas have provided better queue management. As shown in throughput analyses the Vegas provides stable performance with smaller loss, due to the technique used in congestion control, which does not consider the loss event to adjust the window size. This assists the Vegas to reduce packet loss and provide acceptable bottleneck management; however, the throughput will be degraded.

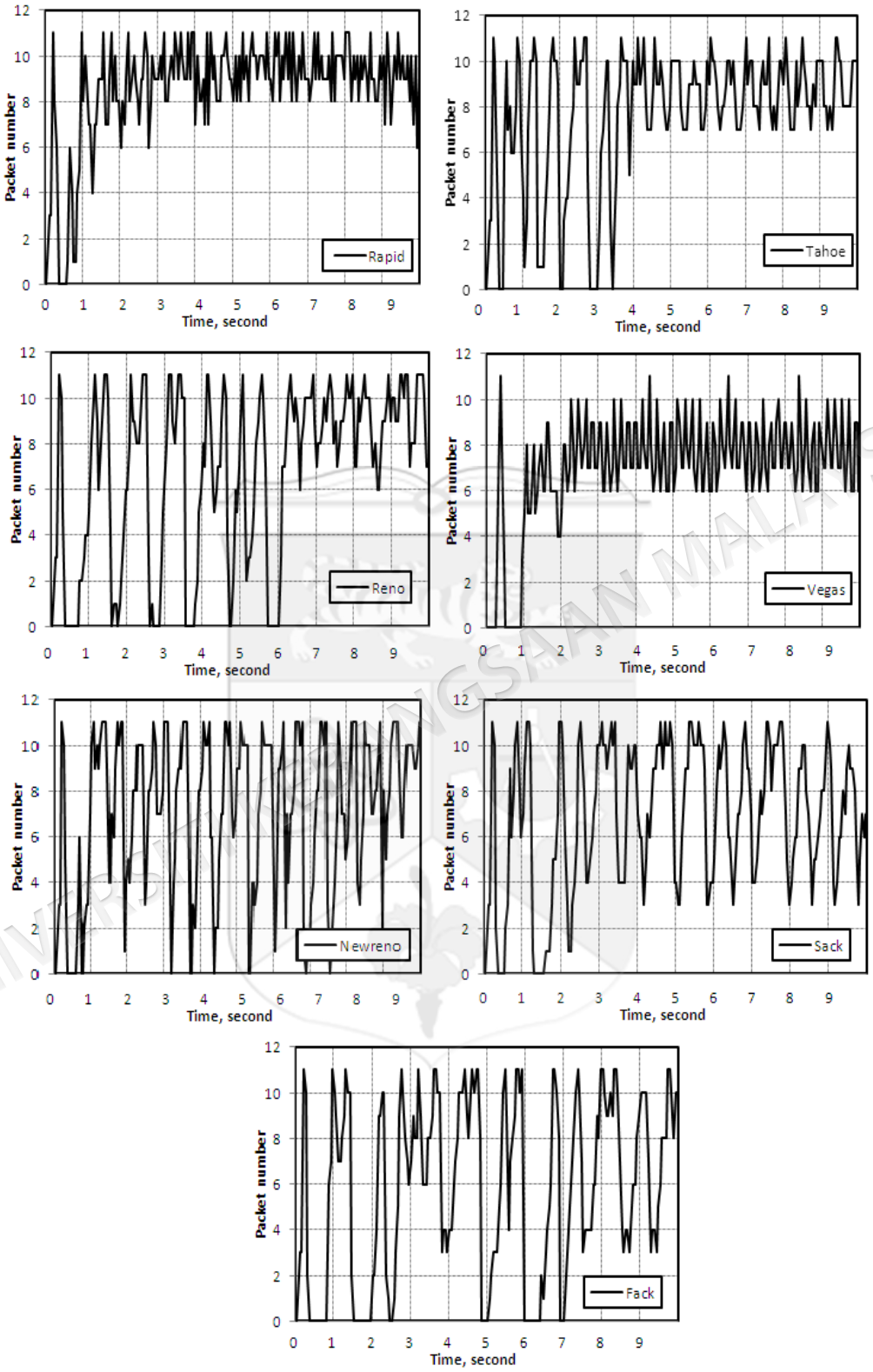


Figure 6.6 Queue size management comparisons of TCP Rapid against other TCP variants

With TCP Rapid, the queue size illustrated in Figure 6.6 moderately manages the data rate in queue with semi-flat behaviour, especially after three seconds in the simulation session. The Rapid provides high queue management and high throughput over large-bandwidth low-latency link; because it enhances the congestion avoidance by estimating the available number of flows, before sending the packets and before increasing the window size.

As explained in Chapter IV, the number of flows can be dynamically estimated according to *cwnd* in current situations and during the occurrences of losses. Therefore, the enhanced congestion control assigns the suitable number of virtual flows, to avoid sending large number of packets that may cause hard congestion in bottleneck connection. Despite transferring large amount of packets in the network, the fair queue management provided by the Rapid congestion control decreases the number of packet loss in bottleneck, by reducing the number of packets injected into network connections, before the connection goes into overflow state.

6.3.5 Packet Loss Comparisons of TCP Rapid against other TCP Variants

In this research as the enhanced TCP is proposed to run over LTE-Advanced network, the packet loss estimation for TCP Rapid keeps all the model parameters without modifying any of the Rapid limitations. In Figure 6.7, the packet loss for TCP variants plus TCP Rapid are illustrated. In packet loss simulation scenario, all the TCP protocols run with normal network condition, whereas, the re-routing and high congestion scenarios are set to be passive. This setting will assist to avoid the large amount of packet loss and to discover the difference in packet loss, when the network is under normal situation. In fact, if the experiment is conducted under high-congested network, the graphs will become quite complex and distorted to recognize, which will make the comparative analysis difficult.

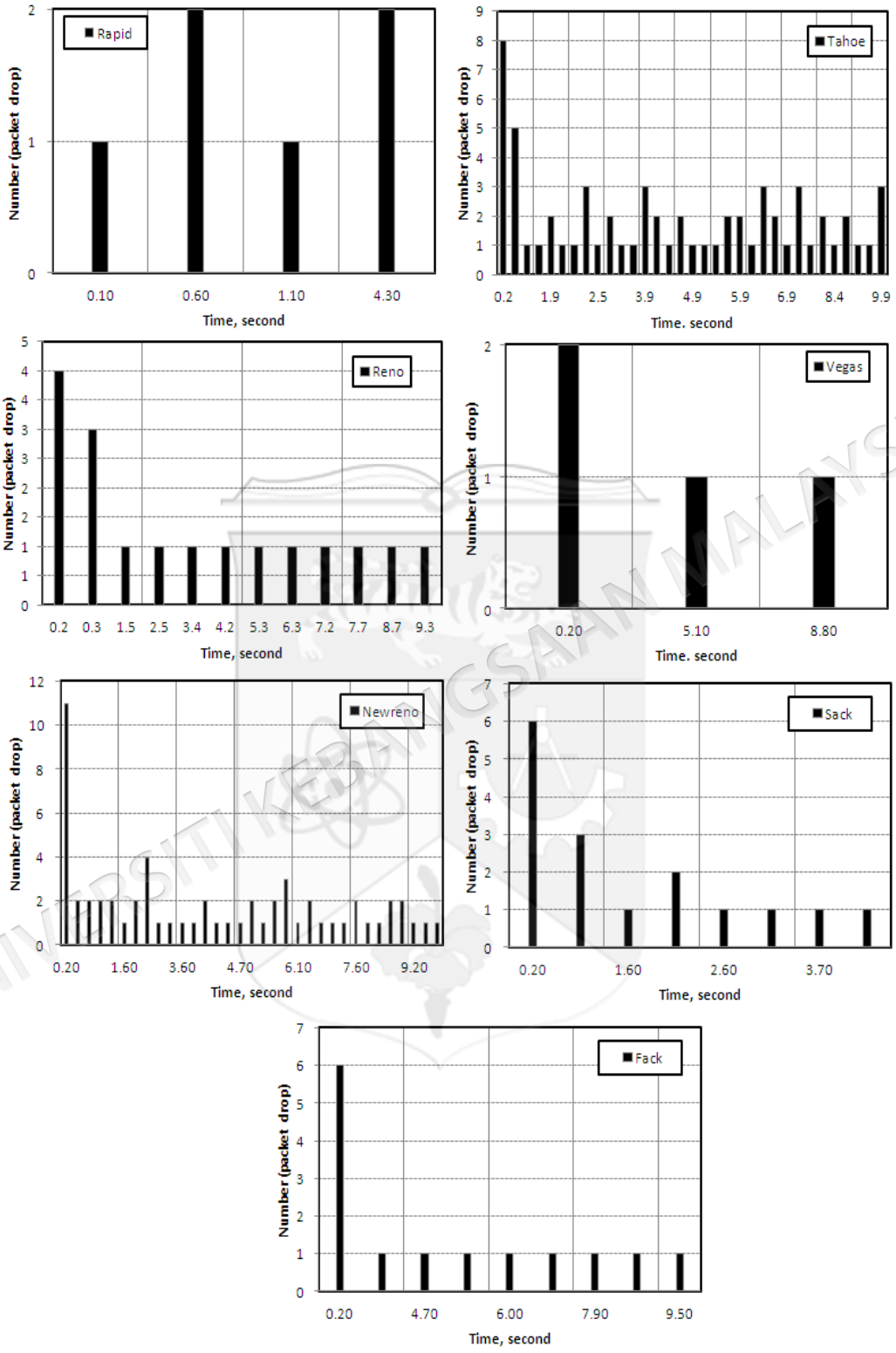


Figure 6.7 Packet loss comparisons of TCP Rapid against other TCP variants over low congested network

Figure 6.7 show that Tahoe and Newreno have offered high packet loss rate despite the network scenario has not included wide congestion state. This is because the standard congestion control mechanism is used in Tahoe and due to the degradation of Newreno when used over large bandwidth connection. On other hand, Rapid and Vegas have offered lower packet loss and kept the loss in reasonable level. Of which the Vegas has provided lower loss level than the Rapid, but the throughput and the queuing management of Rapid is better than Vegas. Certainly, Rapid did not encounter any packet loss after 4.3 s, while in Vegas the packet loss continued until the end of simulation or exactly for 8.8 s.

The congestion control of Rapid is capable of providing high performance in both, throughput and packet loss. This is due to the precise design of the proposed congestion avoidance mechanism and employing multiple packets flow over the available bandwidth.

6.4 TCP RAPID GOODPUT

Generally, TCP goodput refers to the number of useful data bits transferred by the network elements, to certain destinations, per unit of time. Specifically, the goodput is defined as the sum of single packets delivered to host in a given time period. The rationale behind measuring the goodput is that, the end operators of TCP are concerned with transferring single packets and do not care about the number of retransmissions cycles executed by the TCP source (Sharma et al. 2010). For example, if a file is transmitted from a TCP sender, the goodput estimated will correspond to the size of file in bits, divided by the file transferring time duration.

Goodput is slightly lesser than the throughput, because throughput is the gross bits rate, which is physically transmitted. Mostly, the goodput estimation requires the limited network access connections rapidity (the link bandwidth or the link capacity) than throughput. On other hand, the TCP flow control, congestion avoidance, and slow-start may possibly cause degradation in goodput, which does not widely affect the throughput (Celandroni & Potortì 2003).

6.4.1 Test Topology and Simulation Parameters

The simulation of the goodput for TCP Rapid uses new simple topology, instead of the LTE-Advanced model that has been previously proposed in Figure 6.1. This is because the goodput is measures the data transfer over single bottleneck within specified simulation time. Basically, the proposed topology includes two routing nodes, R1 and R2, with large bandwidth of 100 Mbps and low propagation delay of 3 ms (these parameters refers to the main LTE-Advanced links) as shown in Figure 6.8. The TCP source has four nodes (S1, S2, S3, and S4) that are connected through the bottleneck link R1-R2 to the corresponding four TCP destinations (D1, D2, D3, and D4) connected to R1 and R2 with 10 Mbps bandwidth and 3 s latency.

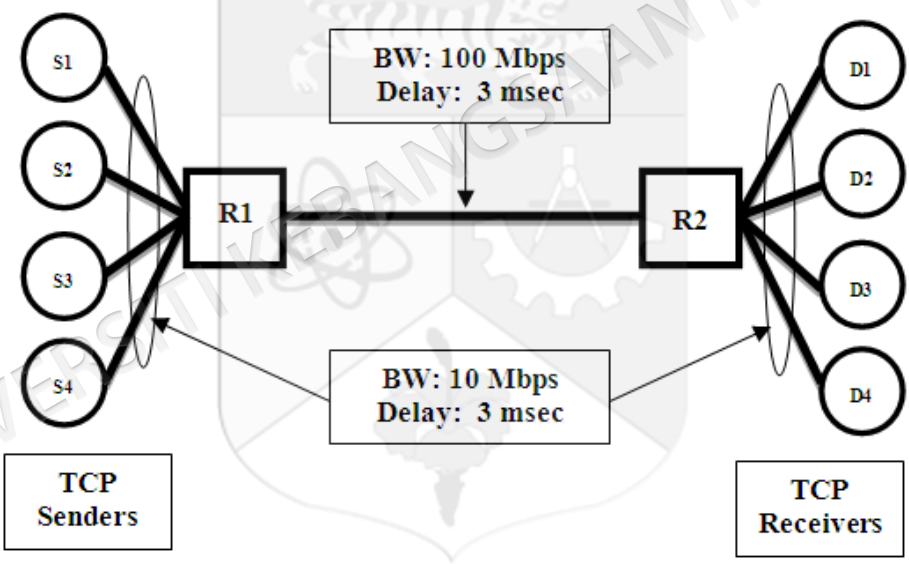


Figure 6.8 Proposed network topology used for goodput measurements

The queue size for the bottleneck link is set to 64 packets, and the Drop-Tail is chosen as queuing type. On other hand, the maximum bound of TCP window size is set to 128 packets and tcpTick to 0.5 s, where tcpTick is the timer clock granularity for measuring RTT intervals (the default value 0.1 s).

Generally, TCP Goodput is a measure of total TCP payload bytes per second that the system under test can transmit and receive between its ports. In addition, the maximum segment size (MSS) impact the goodput, hence the simulation has used two segment sizes such as, 536 bytes and 1460 bytes, to obtain accurate results to the estimated goodput. Finally, the simulation duration is set to 11 s for each experiment (when MSS= 536 bytes and when MSS=1460 bytes). Table 6.3 summarizes the simulation parameters to estimate the goodput for TCP Rapid against other TCP variants.

Table 6.3 Goodput simulation parameters

Parameter	**Value**
Number of Source nodes	4
Number of Source nodes	4
Number of Routers	2
Bandwidth of Bottleneck (R1-R2)	100 Mbps
Propagation delay for all links	3 ms
Bandwidth of Source Nodes (S - R1)	10 Mbps
Bandwidth of Destination Nodes (D - R1)	10 Mbps
Packet size	536,1460 bytes
Max. window size	128 packets
Queue scheme	Drop-Tail
tcpTick	0.5 s
Queue size	64 packets
Simulation time	11 s
TCP protocol	Tahoe, Reno, Newreno, Vegas, Sack, Fack, and Rapid

6.4.2 Goodput Performance and Analysis

As mentioned before, the goodput evaluation is based on two trials; the first when the MSS size is equal 536 bytes and the second when the size is equal 1460 bytes.

In each simulation, only one goodput test of the TCP versions is evaluated, in addition to TCP Rapid. Therefore, the total sessions became seven, because the test script separately measures the goodput for each MSS value. The full results of the goodput measurements over the proposed topology for TCP Rapid with the other TCP source variants are illustrated in Table 6.4 and in the bar chart shown in Figure 6.9.

Table 6.4 Goodput results of Rapid and other TCP's

TCP	**Simulation 1 MSS=536 bytes Goodput (Mbps)**	**Simulation 2 MSS=1460 bytes Goodput (Mbps)**
Tahoe	18.220	18.837
Reno	21.936	25.179
Newreno	25.044	27.860
Vegas	8.640	6.723
Sack	22.110	23.752
Fack	1.369	3.107
Rapid	34.521	36.624

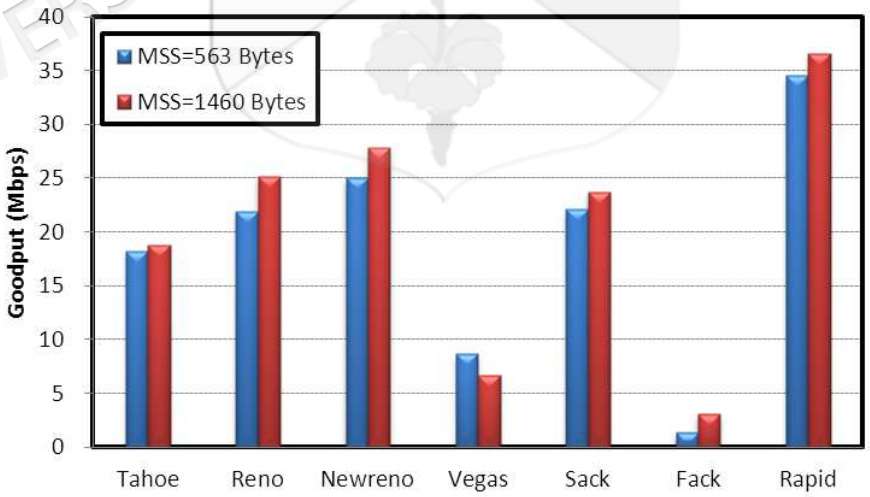


Figure 6.9 Demonstration of goodput measurements of TCP Rapid against other TCP variants with two different MMS values, 536 bytes and 1460 bytes

From the results as shown in Figure 6.9, the Fack TCP provides lower goodput, while TCP Rapid delivers higher goodput level in both segment sizes. In addition, the Reno, Sack, and Newreno have performed well and have given reasonable goodput as against Tahoe and Fack. This is because the goodput takes into account the number of retransmissions, which happens when packet dropped. Hence, when the Reno, Newreno, and Sack (Sack is improved Reno version) uses fast recovery and fast retransmit, to recover the lost packets, the goodput achieves appropriate levels.

Rapid is built over the Reno as previously explained, thus the facilities of fast recovery and fast retransmit are available. Furthermore, other factors that influence the goodput are the fast start in slow-start phase and the multiple virtual flows available in congestion avoidance phase. The slow-start will assist Rapid to send the packet amounts in shorter period and the duration represents the major factor in goodput measurements. Additionally, the multiple flows forces the Rapid to send larger amount of data over the large bottleneck bandwidth. Numerically, Rapid increases the goodput by 57% as compared with Reno, where the data bit rate in this test is increased from 21.936 Mbps in Reno to 34.521 Mbps in Rapid, when the MSS was equal to 536 bytes and about 45% when MSS was equal to 1460 bytes as illustrated in Table 6.4.

6.5 SUMMARY

This chapter had proposed the traffic model of LTE-Advanced networks and traffic types; furthermore it had thoroughly explained all links and interfaces between LTE-Advanced elements. In addition, the simulation parameters and the model configuration were illustrated in this chapter, to obtain the final transport model with the demonstration of the topology animation using NS-2 simulator. The aim of the modelling of LTE-Advanced was to evaluate the performance of TCP Rapid over LTE-Advanced model, to validate the high-expected throughput of Rapid. The performance evaluation of Rapid was performed and the results were compared with other TCP variants in terms of congestion window, throughput, packet loss, and queue size.

The slow-start analysis had proved that, the Rapid slow-start is faster than Reno (and other TCP variants) and it reduces the start-up period. On other hand, the congestion window of Rapid had verified respectable level with regular behaviour and frequent clocking. In all the evaluation stages, the Rapid had shown decent performance with higher throughput, lower packet loss, and acceptable queue size management, wherever the throughput and the goodput increased as compared with Reno. Numerically, the performance evaluation or TCP Rapid over LTE-Advanced improved the slow-start initialization period by 32% compared to Newreno and the normalized throughput improved by 74% and 40% compared to Tahoe and Newreno respectively while the goodput enhanced by 57% compared to Reno.

CHAPTER VII

CONCLUSIONS AND RECOMMENDATIONS

7.1 CONCULSIONS

This chapter presents a summary of the study undertaken on developing a novel congestion control mechanism to improve the performance of the TCP protocol over LTE-Advanced networks. For this purpose, three integrated mechanisms were used to provide a better behaviour to the congestion window in large-bandwidth low-latency networks, including two slow-start algorithms and one congestion avoidance. One slow-start algorithm, in combination with the congestion avoidance algorithm, was used to achieve high-speed and an intelligent scheme of congestion control that can support the TCP protocol to efficiently run over the LTE-Advanced network. Nonetheless, the major challenge in the proposed mechanism was to make it able to deal with the high-speed links, with the capability to transfer a huge number of packets among the network elements in parallel with avoiding the congestion occurrence to minimize the packet dropping as much as possible. The next section of this chapter presents a conclusion of the current work by resuming the achieved results and suggesting a number of directions for future work, which includes the required steps to be done before the improved TCP, can be deployed.

This research work has proposed and thoroughly investigated a novel mechanism for improving the TCP performance over the environments of LTE-Advanced networks. In specific, the objectives of this research were achieved in order to obtain the enhanced TCP.

These objectives based on new slow-start mechanism combined with congestion avoidance with efficient performance to run over high-speed networks with small RTT.

The first contribution in the development of an efficient slow-start algorithm was to obtain a faster increment to the congestion window instead of the slow increment used by the traditional TCP variants. The source variants use linear-exponential growth to the congestion window in initialization and after the congestion avoidance phases. However, these will degrade the network throughput due to the long period, which requires it to reach the window threshold, in addition to the large amount of packet loss. Thus, in order to reduce the starting up time in the slow-start phase, the window size for each ACK received will be duplicated but not exceed *ssthresh*, and then, the second stage of the slow-start algorithm will begin linear interpolation to obtain small approximated window steps until the *ssthresh* is reached. The integration between these two stages constructs the new slow-start mechanism with a faster increment. Meanwhile, the modelling and experimentation of the proposed slow-start scheme over high-congested network topology, using NS-2 simulator, have shown higher throughput and lower packet loss compared to the standard slow-start.

In order to obtain an efficient slow-start strategy, the interpolation approach used in the second stage in the previous slow-start mechanism was replaced by the Lagrange interpolating method to minimize the number of approximation steps where this represents the second contribution of this research. The second stage in the first slow-start method had been replaced by a new formula based on the quadratic interpolation retrieved from the Lagrange polynomial. The new algorithm provided faster start-up by reduces the iteration steps in order to minimize the initialization period. The behaviour of the congestion window of the proposed slow-start mechanism shows that it has reduced the start-up period, achieved better throughput, and kept the loss rate within the low levels when it was tested over a congested network topology.

The third contribution was built on developing a reliable and adaptive congestion avoidance mechanism. In fact, the improved congestion avoidance mechanism was based on developing the standardization of the classic AIMD in three cascaded stages. The developed mechanism was based on the use of the standard AIMD algorithm, where the AIMD associates linear increases *cwnd* with the exponential decreases if any congestion occurs. Some of the traditional TCP variants use the standard AIMD with standard parameter pair value, AIMD (1,½). This algorithm and the parameters used to control the window increment and decrement in the congestion avoidance phase. The improved AIMD centred on using the new relationship between the parameters pair and exchange (a,b) from (1,½) to (k, $2/(3k+1)$), where k represents a variable parameter that can permit to send k number of virtual TCP flows over the same connection. The improved relationship can provide a wide range of the desired values for a and b according to the virtual paths number assumed by the users.

The use of the multiple virtual flows by the proposed mechanism will help to send multiple packets in parallel over a single TCP connection. This technique is supported by a novel method to estimate the optimum values of k and to obtain the optimum number of virtual flows over the connection when every loss occurs dynamically without any setting by the users. The setting of the connection number by the user practically causes a reasonable amount of loss and an additional recovering period. For this reason, the proposed congestion avoidance algorithm adopted a new approach to estimate the optimum virtual flows available by the network conditions. This particular technique gave the improved AIMD algorithm more reliability to run in a stable mode over different congestion conditions. Moreover, it is able to reduce the TCP flows when it senses a losing event by compare the current congestion window size with the previous size to estimate the candidate flow number. The throughput comparison was to evaluate the improved AIMD and it was shown that when k=4, the throughput became better than it was when k=2. However, in the case when these two values were set manually, and when using the dynamic mechanism to estimate k, the throughput became much better and it was more stable, apart from the reduction in the packet dropping. The multiple virtual flows and the dynamic estimation to the flows number represent the fourth contribution of this study.

With the purposes of testing and evaluating the proposed slow-start and congestion avoidance mechanisms, a new TCP agent was created using the NS-2 simulator and this characterizes the fifth contribution. The combination of the new slow-start and new congestion avoidance mechanisms was constructed to enhance the congestion control mechanism. This congestion control mechanism can be merged with any TCP agent like Reno or Newreno, but for more confidentiality, a new TCP agent called Rapid, which includes the developed congestion control, was formed. The formation, identification, and configuration of the new TCP agent are illustrated in details in addition to the explanation of all the possible modifications in the source files of NS-2. Finally, TCP Rapid included the enhanced congestion control apart from the slow-start, congestion avoidance, and multiple flow facility. Therefore, Rapid has individual characterizations and features, which are not available in any other TCP, but it still open for future developments and modifications.

After the new slow-start mechanism, new congestion avoidance, and new TCP agent, the performance of proposed TCP (Rapid) was evaluated over LTE-Advanced model; this serves as the sixth contribution of the current work. The main objective of this research was to develop TCP congestion control that fulfils the requirements of the LTE-Advanced network and achieves high throughput, efficient, and a reliable connection among the network elements. Consequently, it is important to evaluate the proposed TCP over the LTE-Advanced traffic model because this particular model represents the main traffics of the LTE-Advanced with an extra scenario to get hold of the real behaviour and to indicate the expected performance when this TCP is practically employed. Using NS-2, the traffic model of the LTE-Advanced was simulated by three base stations, 35 UE's nodes, and one relay node. The simulation scenario included some congestion events such as bottlenecks, re-routing, and congested links.

The monitoring of the TCP Rapid over LTE-Advanced was separated into two classes; the first involved the behaviour of congestion window, while the second dealt with several evaluation metrics. The comparisons were performed according to six standards of TCP's, namely, Tahoe, Reno, Newreno, Sack, Fack, and Vegas.

In behaviour observing, the enhanced slow-start mechanism provided by Rapid was faster in the start-up of the than other TCP's, where the initial start-up of congestion window in the Rapid required a shorter time compared to Reno and Tahoe to complete the same phase. On other hand, the congestion window of Rapid gave fast clocking, higher window level, and regular frequency as compared to the other TCP's in spite of the congestion and re-routing events inserted in the simulation scenario.

The evaluation process consists of four metrics, throughput, packet loss, queue size, and goodput. When the TCP Rapid is supported by the new slow-start mechanism only, the average throughput enlarges as compared to Reno, Tahoe, and Vegas. However, when Rapid uses the completely improved congestion control, the throughput was also increased a number of times compared to the throughput provided by Tahoe and Newreno. In queue size monitoring, the TCP Rapid appeared to be a moderate management to the data stream rate in queue with semi-flat behaviour during the simulation time especially after the connection establishment by several seconds.

The estimation and comparison of the packet loss proved that Rapid and Vegas contributed to lesser packet loss and they kept the loss at a reasonable level. Even Vegas has provided a lower packet loss than Rapid, but the throughput and the queuing management of Rapid were better than Vegas. In addition, the loss in Rapid came with the early simulation session, which was then stopped, while in Vegas, packet losing was continued until the last stage of the simulation scenario, indicating that Rapid begins to strongly control the queuing and losing during the simulation.

The last metric used to evaluating the performance of TCP Rapid over LTE-Advanced topology was the goodput. The need to test the goodput besides throughput goes back to the fact that the TCP flow control, congestion avoidance, and slow-start may possibly cause a poorer goodput in spite of the high-level throughput. However, Rapid improved the goodput and achieved higher goodput compared to other TCP variants even when the segment size changed from 536 bytes to 1460 bytes.

Despite the fact that the evaluation processes showed an acceptable performance, as provided by the proposed congestion control, this study still has some limitations that represent challenges before the practical employment. One of these challenges is that the TCP Rapid still requires some extra modifications to fulfil all the requirements of the LTE-Advanced networks such as multi-homing and multi-streaming. As for the LTE-Advanced, the proposed model also needs to add wireless nodes as user equipment with full mobility features to observe the behaviour of the proposed congestion control with some handover scenarios.

7.2 RECOMMENDATIONS FOR FUTURE WORK

In this section, some recommendations are put forward based on the findings of the present study. This is hoped to further extend and improve the results presented in this thesis. These recommendations are summarized as follows:

1. Using different interpolating methods to improve the congestion window starting-up after the duplicating stage in the slow-start phase.

2. Modifying the proposed AIMD algorithm in order to find reliable and efficient relationship between the parameters pair *a* and *b*.

3. Improving the dynamic method to estimate the virtual flows number according to other network conditions to be dependent on other network conditions.

4. Employing TCP Rapid to run within patch format that can be executed directly on NS-2 over different versions.

5. Increasing the number of UE used in the LTE-Advanced model to obtain more traffics among the model elements.

6. Although the terminal nodes (UEs) are assumed as wired nodes, it may be suitable to add some wireless nodes as well.

7. A handover scenario is not considered in this research. Thus, it is appropriate to test some handover scenarios and to monitor the behaviour of the proposed congestion control, especially in the slow-start phase.

8. In addition to the LTE-Advanced, it is possible to experiment the proposed congestion control over different network topologies such as satellite or ad-hoc networks.

REFERENCES

3GPP. 2009a. 3GPP IMT-Advanced Evaluation Workshop - Other considerations. 3GPP IMT-Advanced Evaluation Workshop. Beijing, China, 17-18 December. Available from: http://www.3gpp.org/ftp/workshop/2009-12-17_ITU-R_IMT-Adv_eval/docs/pdf/REV-090022%20other%20considerations.pdf [1 August 2012].

3GPP. 2009b. LTE-Advanced Radio Layer 2 and RRC aspects. *3GPP LTE-Advanced Evaluation Workshop,* TSG-RAN WG2. Available from: http://www.3gpp.org/ftp/workshop/2009-12-17_ITU-R_IMT-Adv_eval/docs/pdf/REV-090004%20Radio%20layer%202%20and%20RRC%20aspects.pdf [1 August 2012].

3GPP. 2012. Overview of 3GPP release 8. *Tech. Rep* v.0.2.6. Available from: http://www.3gpp.org/ftp/Information/WORK_PLAN/Description_Releases/ [1 August 2012].

Abeta, S. 2010. Toward LTE Commercial Launch and Future Plan for LTE Enhancements (LTE-Advanced). 146-150.

Abrahamsson, H., Hagsand, O. & Marsh, I. 2002. TCP over High Speed Variable Capacity Links: *A Simulation Study for Bandwidth Allocation*. 117-129.

Afif, M., Martins, P., Tabbane, S. & Godlewski, P. 2005. A SCTP–Layer 2 Cross Layer Mechanism for Data Handover in Wireless Networks (application to EGPRS).

Agilent. 2010. Introducing LTE-Advanced, *Agilent Technologies*. Available from: http://cp.literature.agilent.com/litweb/pdf/5990-6706EN.pdf [1 August 2012].

Akyildiz, I. F., Gutierrez-Estevez, D. M. & Reyes, E. C. 2010. The Evolution to 4G Cellular Systems: LTE-Advanced. *Physical Communication* 3(4): 217-244.

Alex, S. 2006. Computer Network Layers. *CIS748 Class Notes*. Available from: http://www.theparticle.com/cs/bc/net/layers.pdf [1 August 2012].

Ali, R. 2010. Performance of Network Redundancy in SCTP. Master Program in Computer Science, Doctoral Dissertation, University West.

Allcock, W., Hegde, S. & Kettimuthu, R. 2005. Restricted Slow-Start for TCP. *IEEE International Conference in Cluster Computing.* 1-2.

Allman, M. 2003. TCP Congestion Control with Appropriate Byte Counting (ABC). Available from: http://tools.ietf.org/html/rfc3465 [1 August 2012].

Alouini, M. S., Tang, X. & Goldsmith, A. J. 1999. An Adaptive Modulation Scheme for Simultaneous Voice and Data Transmission over Fading Channels. *Selected Areas in Communications, IEEE Journal on* 17(5): 837-850.

Antila, J. 2005. TCP Performance Simulations Using Ns2. Available from: http://web.mst.edu/~bckd2/CpE401/project/NS2%20simulations/special_study%20on%20TCP.pdf [1 August 2012].

Astély, D., Dahlman, E., Furuskar, A., Jading, Y., Lindstrom, M. & Parkvall, S. 2009. LTE: the evolution of mobile broadband. *Communications Magazine, IEEE* 47(4): 44-51.

Bajkó, G., Moldován, I., Pop, O. & Bíró, J. 2004. TCP Flow Control Algorithms for Routers. *High Speed Networks Laboratory, Department of Telecommunications and Telematics, Technical University of Budapest-Hungary*.0-4.

Bajzik, L., Horvath, P., Korossy, L. & Vulkan, C. 2007. Impact of Intra-LTE Handover with forwarding on the user connections. *16th Mobile and Wireless Communications Summit*. 1-5.

Bakre, A. & Badrinath, B. 1995. I-TCP: Indirect TCP for mobile hosts. *15th International Conference on Distributed Computing Systems*. 136-143.

Balakrishnan, H., Seshan, S., Amir, E. & Katz, R. H. 1995. Improving TCP/IP Performance over Wireless Networks. *ACM MOBICOM* . 2-11.

Bannister, J., Mather, P. & Coope, S. 2004. *Convergence Technologies For 3G Networks: IP, UMTS, EGPRS and ATM*. New York: John Wiley & Sons Inc.

Bansal, D. 2005. Third-Party TCP Rate Control. Available from: http://www.novell.com/connectionmagazine/2001/05/sequence51.pdf [1 August 2012].

Bao, G. 1996. Performance Evaluation of TCP/RLP Protocol Stack over CDMA Wireless Link. *Wireless Networks* 2(3): 229-237.

Bates, R. J. 2002. *Broadband Telecommunications Handbook*. New York: McGraw-Hill, Inc.

Biaz, S. & Vaidya, N. H. 2005. De-randomizing Congestion Losses to Improve TCP Performance over Wired-Wireless Networks. *IEEE/ACM Transactions on Networking (TON)* 13(3): 596-608.

Brakmo, L. S., O'malley, S. W. & Peterson, L. L. 1994. TCP Vegas: New Techniques for Congestion Detection and Avoidance. *Department of Computer Science, University of Arizona*. 24-35.

Brakmo, L. S. & Peterson, L. L. 1995. TCP Vegas: End to End congestion avoidance on a global Internet. *IEEE Journal on Selected Areas in Communications* 13(8): 1465-1480.

Bartók, I. & Cselényi, I. 2001. Implementation and Evaluation of the BLUE Active Queue Management Algorithm. Diploma thesis, Budapest University of Technology and Economics.

Brown, K. & Singh, S. 1997. M-TCP: TCP for mobile cellular networks. *ACM SIGCOMM Computer Communication Review* 27(5): 19-43.

Caini, C., Liberato, N. C., Firrincieli, R. & Giambene, G. 2006. TCP Hybla Performance in GEO Satellite Networks: Simulations and Testbed. *International Workshop on Satellite and Space Communications*.41-45.

Carney, V. 2008. Message Integrity, Reliable Delivery, Acknowledgements, Non-Repudiation and Encryption. Available from: http://www.veritycarney.net/articles/Message_Integrity.pdf [1 August 2012].

Cavendish, D., Kumazoe, K., Tsuru, M., Oie, Y. & Gerla, M. 2009. CapStart: An Adaptive TCP Slow Start for High Speed Networks. *International Conference on INTERNET*. 15-20.

Celandroni, N. & Potortì, F. 2003. Maximizing single connection TCP goodput by trading bandwidth for BER. *International Journal of Communication Systems* 16(1): 63-79.

Chan, A. C. F., Tsang, D. H. K. & Gupta, S. 1997. Impacts of Handoff on TCP Performance in Mobile Wireless Computing. *IEEE International Conference on Personal Wireless Communications* . 184-188.

Chan, Y. C., Chan, C. T. & Chen, Y. C. 2004. Performance improvement of TCP Vegas over heterogeneous networks. *24th International Conference on Distributed Computing Systems Workshops*. 36-41.

Chappell, L. 2001. TCP Sequencing and Sliding Windows. *Novel Certified Professional*. Available from: http://www.novell.com/connectionmagazine/2001/05/sequence51.pdf [1 August 2012].

Chen, X., Zhai, H., Wang, J. & Fang, Y. 2005. A survey on Improving TCP Performance over Wireless Networks. *Resource Management in Wireless Networking*: 657-695.

Cheng, R. S., Lin, H. T., Hwang, W. S. & Shieh, C. K. 2005. Improving The Ramping up Behaviour of TCP Slow Start. *19th International Conference on Advanced Information Networking and Applications*. 807-812.

Chiu, D. M. & Jain, R. 1989. Analysis of the Increase and Decrease Algorithms for Congestion Avoidance in Computer Networks. *Computer Networks and ISDN systems* 17(1): 1-14.

Chockalingam, A., Zorzi, M. & Rao, R. R. 1998. Performance of TCP on Wireless Fading Links with Memory. *IEEE International Conference on Communications,* ICC. 595-600.

Chouly, A., Brajal, A. & Jourdan, S. 1993. Orthogonal Multicarrier Techniques Applied to Direct Sequence Spread Spectrum CDMA Systems. *IEEE Global Telecommunications Conference*. 1723-1728.

Chuah, M. C. & Zhang, Q. 2006. *Design and Performance of 3G Wireless Networks and Wireless LANS*. USA: Springer.

Corral, G., Zaballos, A., Fulgueira, T. & Abella, J. 2000. Simulation-based study of TCP flow control mechanisms using OPNET Modeler. Available from: http://web.salleurl.edu/~tomasf/index_fitxers/article.pdf [1 August 2012].

Crowcroft, J. & Oechslin, P. 1998. Differentiated end-to-end Internet Services Using a Weighted Proportional Fair Sharing TCP. *ACM SIGCOMM Computer Communication Review* 28(3): 53-69.

Dahlman, E. 2008. *3G Evolution: HSPA and LTE for Mobile Broadband*. Oxford: Academic Press.

Dahlman, E., Parkvall, S. & Sköld, J. 2011. *4G: LTE/LTE-Advanced for Mobile Broadband*. Oxford: Academic Press.

de AE Lima, M. M., da Fonseca, N. L. S. & de Rezende, J. F. 2003. On the Performance of TCP Loss Recovery Mechanisms. *IEEE International Conference on Communications*. 1812-1816.

De Souza, E. & Agarwal, D. 2003. Report on A High-Speed TCP Study. *Characteristics and Deployment Issues, Lawrence Berkeley National Lab*. Berkeley, USA.

Demers, A., Keshav, S. & Shenker, S. 1989. Analysis and Simulation of a Fair Queueing Algorithm. *ACM SIGCOMM Computer Communication Review* 19(4): 1-12.

Eddy, W. & Swami, Y. 2005. Report on Adapting End Host Congestion Control for Mobility. *National Aeronautics and Space Administration*. USA.

Ekiz, N., Rahman, A. H. & Amer, P. D. 2011. Misbehaviours in TCP SACK Generation. *ACM SIGCOMM Computer Communication Review* 41(2): 16-23.

Elaarag, H. 2002. Improving TCP performance over mobile networks. *ACM Computing Surveys (CSUR)* 34(3): 357-374.

Fahmy, S. & Karwa, T. P. 2000. Report on TCP Congestion Control: Overview and Survey of Ongoing *Department of Computer Science Research, Purdue University*. France.

Fall, K. & Floyd, S. 1996. Simulation-based comparisons of Tahoe, Reno and SACK TCP. *ACM SIGCOMM Computer Communication Review* 26(3): 5-21.

Fisk, M. & Feng, W. 2001. Dynamic Right-Sizing in TCP. Available from: http://library.lanl.gov/cgi-bin/getfile?00796247.pdf [1 August 2012].

Floyd, S. 1996. Issues of TCP with SACK. Technical report, January 1996. Available from: ftp://ftp.ee.lbl.gov/papers/issues_sa.pdf [1 August 2012].

Floyd, S. 2003. HighSpeed TCP for Large Congestion Windows. Available from: http://rsync.tools.ietf.org/html/rfc3649 [1 August 2012].

Floyd, S. 2004. Limited Slow-Start for TCP with Large Congestion Windows. Available from: https://art.tools.ietf.org/html/rfc3742 [1 August 2012].

Floyd, S., Handley, M. & Padhye, J. 2000. A Comparison of Eequation-based and AIMD Congestion Control. Available from: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.37.7442&rep=rep1&type=pdf [1 August 2012].

Floyd, S., Handley, M. & Padhye, J. 2000a. A Comparison of Equation-Based and AIMD Congestion Control. *ACIRI*. Available from: http://web.cs.wpi.edu/~claypool/courses/525-S01/slides/FHP00.pdf [1 August 2012].

Floyd, S., Henderson, T. & Gurtov, A. 1999. The NewReno Modification to TCP's Fast Recovery Algorithm. RFC 2582. Available from: http://www.hjp.at/doc/rfc/rfc3782.html [1 August 2012].

Floyd, S. & Jacobson, V. 1991. Traffic phase effects in packet-switched gateways. *ACM SIGCOMM Computer Communication Review* 21(2): 26-42.

Fu, C. P. & Liew, S. C. 2003. TCP Veno: TCP enhancement for transmission over wireless access networks. *Selected Areas in Communications, IEEE Journal on* 21(2): 216-228.

Gasca, M. & Sauer, T. 2000. Polynomial interpolation in several variables. *Advances in Computational Mathematics* 12(4): 377-410.

Gevros, P., Risso, F. & Kirstein, P. 1999. Analysis of a Method for Differential TCP Service. *Global Telecommunication Conference*. 1699-1708.

Ghazaleh, H. & Muhanna, M. 2008. Enhancement of Throughput Time using MS-TCP Transport Layer Protocol for 4G Mobiles. *5th International Multi-Conference on Systems, Signals and Devices*. 1-5.

Ghosh, A., Zhang, J., Muhamed, R. & Andrews, J. G. 2010. *Fundamentals of LTE*. New Jersey: Prentice Hall.

Giordano, S., Pagano, M., Procissi, G. & Secchi, R. 2008. A Simple Markovian Model of TCP Startup Behaviour. Available from: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.109.4152&rep=rep1&type=pdf [1 August 2012].

Goff, T., Moronski, J., Phatak, D. S. & Gupta, V. 2000. Freeze-TCP: A True end-to-end TCP Enhancement Mechanism for Mobile Environments. *19th Annual Joint Conference of the IEEE Computer and Communications Societies.* 1537-1545.

Grieco, L. A. & Mascolo, S. 2004. Performance evaluation and comparison of Westwood+, New Reno, and Vegas TCP congestion control. *ACM SIGCOMM Computer Communication Review* 34(2): 25-38.

Gurtov, A. 2000. TCP performance in the Presence of Congestion and Corruption Losses. *Master's Thesis, University of Helsinki, Department of Computer Science*, Finland.

Ha, S., Rhee, I. & Xu, L. 2008. CUBIC: A new TCP-friendly high-speed TCP variant. *ACM SIGOPS Operating Systems Review* 42(5): 64-74.

Haeri, M. & Rad, A. H. M. 2004. TCP Retransmission Timer Adjustment Mechanism Using Model-based RTT Predictor. *5th Asian Conference on Control.* 686-693.

Hassan, I. A. 2005. Predicting TCP Congestion through Active and Passive Measurements. Available from: http://eternity.iu.hio.no/theses/pdf/master2005/Ismail.pdf [1 August 2012]

Hengartner, U., Bolliger, J. & Gross, T. 2000. TCP Vegas Revisited. *19th Annual Joint Conference of the IEEE Computer and Communications Societies*. 1546-1555.

Henna, S. 2009. A Throughput Analysis of TCP Variants in Mobile Wireless Networks. *Third International Conference on Next Generation Mobile Applications, Services and Technologies.* 279-284.

Hildebrand, F. B. 1987. *Introduction to Numerical Analysis*. New York: Dover Publications.

Hirabaru, M. 2006. Impact of Bottleneck Queue Size on TCP Protocols and Its Measurement. IEICE *Transactions on Information and Systems Series D* 89(1): 132-140.

Ho, C. Y., Chan, Y. C. & Chen, Y. C. 2005. An Enhanced Slow-Start Mechanism for TCP Vegas. *Journal of Communications and Networks*. 405-411 Vol. 401.

Holma, H. & Toskala, A. 2000. *WCDMA for UMTS*. West Sussex: Wiley Online Library.

Hoque, K., Haque, R. R., Hossain, M. A., Farazi, M. S. F. & Hossain, G. 2007. Modeling and Performance of TCP in a MC-CDMA System for 4G Communications. *10th International Conference on Computer and Information Technology*. 1-5.

Hughes. 2006. TCP Revisited. Available from: http://www.hsc.com/Portals/0/Uploads/Articles/HSC-TCP Revisited633851531918260439.pdf [1 August 2012].

Iguchi, T., Hasegawa, G. & Murata, M. 2005. A New Congestion Control Mechanism of TCP with Inline Network Measurement. *Information Networking. Convergence in Broadband and Mobile Networking*: 109-121.

Islam, M. S., Kashem, M., Sadid, W., Rahman, M., Islam, M. & Anam, S. 2009. TCP variants and network parameters: a comprehensive performance analysis. *The International Multi Conference of Engineers and Computer Scientists.* 1-6.

Iwamura, M., Takahashi, H. & Nagata, S. 2009. Relay Technology in LTE-Advanced. *NTT DoCoMo Technical Journal* 12(2): 29-36.

Iyengar, J., Ford, B., Amin, S. O., Nowlan, M. F. & Tiwari, N. 2011. Wire-Compatible Unordered Delivery in TCP and TLS. *Arxiv preprint arXiv:1103.0463*.

Jacobson, V. 1988. Congestion Avoidance and Control. University of California at Berkeley. 314-329. Available from: http://ee.lbl.gov/papers/congavoid.pdf [1 August 2012].

Jain, R. & Ramakrishnan, K. 1988. Congestion Avoidance in Computer Networks with a Connectionless Network Layer: Concepts, goals and methodology. 134-143.

Jamal, H. & Sultan, K. 2008. Performance analysis of TCP Congestion Control Algorithms. *International Journal of Computers and Communications* 2(1): 18-24.

Jin, C., Wei, D., Low, S. H., Bunn, J., Choe, H. D., Doylle, J., Newman, H., Ravot, S., Singh, S. & Paganini, F. 2005. FAST TCP: From Theory to Experiments. *Network, IEEE* 19(1): 4-11.

Kaaranen, H., Ahtiainen, A., Laitinen, L., Naghian, S. & Niemi, V. 2001. *UMTS Networks*. West Sussex: Wiley Online Library.

Karakaya, B., Arslan, H. & C rpan, H. A. 2009. An Adaptive Channel Interpolator Based on Kalman Filter for LTE Uplink in High Doppler Spread Environments. *EURASIP Journal on Wireless Communications and Networking* : 7-16.

Karn, P. & Partridge, C. 1987. Improving round-trip time estimates in reliable transport protocols. *ACM SIGCOMM Computer Communication Review* 17(5): 2-7.

Kelly, T. 2003. Scalable TCP: Improving Performance in Highspeed Wide Area Networks. *ACM SIGCOMM Computer Communication Review* 33(2): 83-91.

Kettimuthu, R. & Allcock, W. 2004. Improved Selective Acknowledgment Scheme for TCP. *International Conference on Internet Computing.* 913-919.

Khan, F. 2009. *LTE for 4G Mobile Broadband: Air Interface Technologies and Performance.* Cambridge University Press.

Kiiski, M. 2010. LTE-Advanced: The Mainstream in Mobile Broadband Evolution. *European Wireless Conference*. 983-988.

Kliazovich, D., Granelli, F., Redana, S. & Riato, N. 2007. Cross-layer Error Control Optimization in 3G LTE. *Global Telecommunications Confere*nce. 2525-2529.

Kodama, M., Hasegawa, G. & Murata, M. 2008. Implementation experiments of TCP Symbiosis: bio-inspired mechanisms for Internet congestion control. *Proceedings of CQR.*

Koga, H. 2009. Dynamic TCP acknowledgment with sliding window. *Theoretical Computer Science* 410(8-10): 914-925.

Kozierok, C. 2005. *The TCP/IP Guide*. San Francisco: No Starch Press.

Kuo, F. C. & Fu, X. 2008. Probe-Aided MulTCP: An Aggregate Congestion Control Mechanism. *ACM SIGCOMM Computer Communication Review* 38(1): 17-28.

La, R. J., Mo, J., Walrand, J. & Anantharam, V. 1998. A Case for TCP Vegas and Gateways using Game Theoretic Approach. Available from: http://walrandpc.eecs.berkeley.edu/Papers/game.pdf [1 August 2012].

La, R. J., Walrand, J. & Anantharam, V. 1999. *Issues in TCP Vegas.* Electronics Research Laboratory, College of Engineering, University of California.

Lai, Y. C. & Yao, C. L. 2001. TCP Congestion Control Algorithms and a Performance Comparison. *Tenth International Conference on Computer Communications and Networks.* 523-526.

Lakshman, T. & Madhow, U. 1997. The performance of TCP/IP for networks with high bandwidth-delay products and random loss. *Networking, IEEE/ACM Transactions on* 5(3): 336-350.

Law, K. L. E. & Hung, W. C. 2001. Problems and Solutions for the TCP Slow-Start Process. Available from: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.67.8163&rep=rep1&type=pdf [1 August 2012].

Leith, D. & Shorten, R. 2004. H-TCP: TCP for High-Speed and Long-Distance Networks. Conference of *PFLDnet'04*.

Lescuyer, P. & Lucidarme, T. 2008. Evolved packet System (EPS). *The LTE and SAE Evolution of 3G UMTS*. West Sussex:Wiley.

Liao, W., Kao, C. J. & Chien, C. H. 2005. Improving TCP performance in mobile networks. *Communications, IEEE Transactions on* 53(4): 569-571.

Lin, D. & Kung, H. 1998. TCP fast recovery strategies: Analysis and Improvements. *Seventeenth Annual Joint Conference of the IEEE Computer and Communications Societies*. 263-271.

Liu, Y., Huang, H., Xu, K. & Yang, C. 2008. Developing Slow Start over Wireless Networks. *International Multisymposiums on Computer and Computational Sciences*. 106-109.

Low, S. H., Peterson, L. L. & Wang, L. 2002. Understanding TCP Vegas: a Duality Model. *Journal of the ACM (JACM)* 49(2): 207-235.

Ludwig, R. & Katz, R. H. 2000. The Eifel Algorithm: Making TCP Robust Against Spurious Retransmissions. *ACM SIGCOMM Computer Communication Review* 30(1): 30-36.

Luglio, M., Roseti, C. & Gerla, M. 2004. The Impact of Efficient Flow Control and OS Features on TCP Performance over Satellite Links. *ASSI Satellite Communication Letter (Sat-Comm Letter), 9th edition, special issue on Multimedia Satellite Communication* 3(1): 1-9.

Mahmoodi, T. 2009. Transport Layer Performance Enhancements over Wireless Networks. Thesis, University of London.

Mascolo, S., Casetti, C., Gerla, M., Sanadidi, M. Y. & Wang, R. 2001. TCP Westwood: Bandwidth Estimation for Enhanced Transport over Wireless Links. *7th Annual International Conference on Mobile Computing and Networking*. 287-297.

Mathis, M., Semke, J., Mahdavi, J. & Ott, T. 1997. The Macroscopic Behaviour of the TCP Congestion Avoidance Algorithm. *ACM SIGCOMM Computer Communication Review* 27(3): 67-82.

Mbarek, R., Othman, M. T. B. & Nasri, S. 2008. Performance Evaluation of Competing High-Speed TCP Protocols. *IJCSNS* 8(6): 99-105.

Meijering, E. 2002. A Chronology of Interpolation: from Ancient Astronomy to Modern Signal and Image Processing. *Proceedings of the IEEE* 90(3): 319-342.

MIT6.02. 2010. Network Layering. *MIT 6.02 DRAFT Lecture Notes* LECTURE 21. Available from: http://web.mit.edu/6.02/www/s2010/handouts/lectures/L21-notes.pdf [1 August 2012].

Modi, N. 2008. Performance Evaluation of TCP Variants over UMTS Networks. Available from: http://uu.diva-portal.org/smash/get/diva2:174009/FULLTEXT01 [1 August 2012].

Möller, N. 2005. *Automatic Control in TCP over Wireless*. School of Electrical Engineering, Royal Institute of Technology.

Moraru, B., Copaciu, F., Lazar, G. & Dobrota, V. 2003. Practical Analysis of TCP Implementations: Tahoe, Reno, Newreno. *RoEduNet International Conference*. 125–138.

Mukherjee, A., Bandyopadhyay, S. & Saha, D. 2003. *Location Management and Routing in Mobile Wireless Networks*. London: Artech House Publishers.

Mulroy, P., Appleby, S., Nilsson, M. & Crabtree, B. 2009. The Use of MulTCP for The Delivery of Equitable Quality Video. *17th International Packet Video Workshop*. 1-9.

Nabeshima, M. 2005. Performance evaluation of multcp in high-speed wide area networks. *IEICE Transactions on Communications* 88: 392-396.

Nagamalai, D. & Lee, J. K. 2004. Performance of SCTP over High Speed Wide Area Networks. *Conference on Cybernetics and Intelligent Systems*. 890-895.

Nagamalai, D., Lee, S. H., Lee, W. G. & Lee, J. K. 2005. SCTP over High Speed Wide Area Networks. *Networking-ICN 2005*: 628-634.

Nakamura, T. 2009. Proposal for Candidate Radio Interface Technologies for IMT-Advanced Based on LTE Release 10 and Beyond (LTE-Advanced). *3rd Workshop on IMT-Advanced.*

Nam, Y. H., Liu, L., Wang, Y., Zhang, C., Cho, J. & Han, J. K. 2010. Cooperative Communication Technologies for LTE-Advanced. *IEEE International Conference Acoustics Speech and Signal Processing,* 5610-5613.

NS-2, 1989. Network Simulator 2. Available from: http://www.isi.edu/nsnam/ns/ [1 August 2012].

Olsén, J. 2003. *Stochastic Modeling and Simulation of the TCP Protocol*. Department of Mathematics, Uppsala University.

Olsson, M., Sultana, S., Rommer, S., Frid, L. & Mulligan, C. 2009. *SAE and The Evolved Packet Core: Driving the Mobile Broadband Revolution*. Burlington: Academic Pr.

Ong, B. L. & Badlishah, R. 2011. Performance Evaluation of TCP Vegas versus Different TCP Variants in Homogeneous and Heterogeneous Wired networks. *WASET Journal* (74): 179-185.

Pacifico, D., Pacifico, M., Fischione, C., Hjalrmasson, H. & Johansson, K. 2009. Improving TCP Performance During the Intra LTE Handover. *Global Telecommunications Conference.* 1-8.

Parkvall, S., Furuskar, A. & Dahlman, E. 2011. Evolution of LTE toward IMT-Advanced. *Communications Magazine, IEEE* 49(2): 84-91.

Parvez, N., Mahanti, A. & Williamson, C. 2006. TCP NewReno: Slow-but-Steady or Impatient? *IEEE International Conference on Communications.* 716-722.

Parziale, L., Organization, I. B. M. C. I. T. S. & Online, S. B. 2006. *TCP/IP Tutorial and Technical Overview*. IBM International Technical Support Organization.

Patil, G., McClean, S. & Raina, G. 2011. Drop tail and RED queue management with small buffers: stability and Hopf bifurcation. *ICTACT Journal on Communication Technology* 2(2): 339-344.

Podlesny, M. & Williamson, C. 2010. Providing Fairness between TCP NewReno and TCP Vegas with RD Network Sservices. *18th International Workshop on Quality of Service*. 1-9.

Puzmanova, R. 2001. *Routing & Switching: Time of Convergence*. Boston: Addison-Wesley.

Qiu, Q., Chen, J., Zhang, Q. & Pan, X. 2009. LTE/SAE Model and its Implementation in NS-2. *5th International Conference on Mobile Ad-hoc and Sensor Networks*. 299-303.

Qiu, Q., Jian, C., Lingdi, P. & Xue-zeng, P. 2010. Hierarchy Virtual Queue Based Flow Control in LTE/SAE. *Second International Conference on Future Network.* 78-82.

Qureshi, B., Othman, M. & Hamid, N. 2009. Progress in Various TCP Variants. *2nd International Conference on Computer, Control and Communication.* 1-6.

Rahman, M., Kabir, A., Lutfullah, K., Hassan, Z. & Amin, M. 2008. Fair Comparisons of Different TCP Variants for Future Deployment of Networks. *International Conference on Electronics, Computer and Communication*. 260-263.

Ramakrishnan, K., Floyd, S. & Black, D. 2001. The addition of explicit congestion notification (ECN) to IP, RFC 3168. Available from: http://www.hjp.at/doc/rfc/rfc3168.html [1 August 2012].

Reale, R., Roverso, R., El-Ansary, S. & Haridi, S. 2012. DTL: Dynamic Transport Library for Peer-To-Peer Applications. *Distributed Computing and Networking*: 428-442.

Ron, A., Li, Y., Cowlishaw, M. & Fillmore, N. 2010. Lecture 7: Newton Interpolation. *Introduction to Numerical Analysis*, CS412.

Rong, L., Elayoubi, S. E. & Haddada, O. B. 2010. Impact of Relays on LTE-Advanced Performance. *IEEE International Conference on Communications*. 1-6.

Sanadhya, S. & Sivakumar, R. 2011. Adaptive Flow Control for TCP on Mobile Phones. *INFOCOM, 2011 Proceedings IEEE.* 2912-2920.

Saniee, K. 2007. A Simple Expression for Multivariate Lagrange Interpolation. *New Providence High School, New Providence* NJ 07974.

Sarolahti, P. 2007. TCP Performance in Heterogeneous Wireless Networks. Available from: https://www.doria.fi/handle/10024/5892 [1 August 2012].

Sarolahti, P. & Kuznetsov, A. 2002. Congestion Control in Linux TCP. *USENIX Annual Technical Conference.* 49-62.

Sharma, P. 2006. Performance Analysis of High-Speed Transport Control Protocols. Thesis Clemson University.

Sharma, S., Gillies, D. & Feng, W. 2010. On the Goodput of TCP NewReno in Mobile Networks. *International Conference on Computer Communications and Networks.* 1-8.

Shirazi, H. 2009. Smart Congestion Control in TCP/IP Networks. *Journal of Information and Communication Technology* 2(2): 73-78.

Sing, J. & Soh, B. 2005. TCP New Vegas: Improving The Performance of TCP Vegas over High Latency Links. *Fourth IEEE International Symposium on Network Computing and Applications*. 73-82.

Singh, M., Pradhan, P. & Francis, P. 2004. MPAT: Aggregate TCP Congestion Management as a Building Block for Internet QoS. *12th IEEE International Conference on Network Protocols.* 129-138.

Siyan, K. S. & Parker, T. 2002. *TCP/IP*. Milano: Apogeo Editore.

Stencel, V., Muller, A. & Frank, P. 2010. LTE Advanced - A Further Evolutionary Step for Next Generation Mobile Networks. *20th International Conference Radioelektronika*. 1-5.

Stewart, R., Ramalho, M., Xie, Q., Tuexen, M. & Conrad, P. 2004. Stream Control Transmission Protocol (SCTP) Partial Reliability Extension. RFC 3758 (Proposed Standard).

Stewart, R., Tüxen, M. & Lei, P. 2008. SCTP: What is it, and how to use it? Available from: http://www.bsdcan.org/2008/schedule/attachments/44_bsdcan_sctp.pdf [1 August 2012].

Subedi, L., Najiminaini, M. & Trajkovi , L. 2008. Performance Evaluation of TCP Tahoe, Reno, Reno with SACK, and NewReno Using OPNET Modeler. *Communication Networks Laboratory - OPNET technologies.*

Swales, A. 1999. Open Modbus/TCP Specification. Available from: http://194.206.139.68/~rellier/archives/2011_2012/tp/openmbus_specif.pdf [1 August 2012].

Tan, Y. 2009. Active Queue Management for LTE uplink in eNodeB. Thesis Helsinki University of Technology.

Tayade, M., Sharma, V. & Sahu, S. 2011. Review of different TCP Variants in Adhoc Networks. *International Journal of Engineering Science* 3.

Tian, Y., Xu, K. & Ansari, N. 2005. TCP in Wireless Environments: Problems and Solutions. *Communications Magazine, IEEE* 43(3): S27-S32.

Vallamsundar, B., Zhu, J., Ponnambalam, K., Huang, C., Srinivasan, A. & Cheng, B. 2007. Congestion Control for Adaptive Satellite Communication Systems Using Intelligent Systems. *International Symposium on Signals, Systems and Electronics.* 415-418.

Vangala, S. & Labrador, M. A. 2002. Performance of TCP over Wireless Networks with the Snoop Protocol. *27th Annual IEEE Conference on Local Computer Networks.* 600-601.

Walrand, J. & Parekh, S. 2010. Communication Networks: A Concise Introduction. *Synthesis Lectures on Communication Networks* 3(1): 1-192.

Wang, H., Xin, H., Reeves, D. S. & Shin, K. G. 2000. A Simple Refinement of Slow-Start of TCP Congestion Control. *Fifth IEEE Symposium on Computers and Communications*. 98-105.

Wang, J. 2004. NS-2 Tutorial. *Multimedia Networking Group, the Department of Computer Science, UVA.*

Wang, J., Wen, J., Zhang, J. & Han, Y. 2011. TCP-FIT: An Improved TCP Congestion Control Algorithm and Its Performance. *IEEE INFOCOM*. 2894-2902.

Wang, Q. & Yuan, D. 2008. An Improved TCP Congestion Control Mechanism with Adaptive Congestion Window. *International Symposium on Performance Evaluation of Computer and Telecommunication Systems.* 231-235.

Wannstrom, J. 2012. LTE-Advanced. *3GPP*. Available from: http://www.3gpp.org/IMG/pdf/lte_advanced_v2.pdf [1 August 2012].

Wei, D. X. & Cao, P. 2006. NS-2 TCP-Linux: An NS-2 TCP Implementation with Congestion Control Algorithms from Linux. *Workshop on NS-2*.

Welzl, M. 2010. Parallel TCP Data Transfers: A Practical Model and its Application. Thesis University of Innsbruck.

Welzl, M., Goutelle, M., He, E., Kettimut, R., Hegde, S., Gu, Y., Allcock, W., Kettimuthu, R., Leigh, J. & Primet, P. 2005. A Survey of Transport Protocols other than "Standard" TCP. DOI: 10.1.1.60.6440 [1August 2012].

Wisely, D. 2009. *IP for 4G*. Great Britain: Wiley.

Wojek, C., Dorkó, G., Schulz, A. & Schiele, B. 2008. Sliding-windows for Rapid Object Class Localization: A Parallel Technique. *Pattern Recognition*: 71-81.

Wu, H., Feng, Z., Guo, C. & Zhang, Y. 2010. ICTCP: Incast Congestion Control for TCP in data center networks. *6th International Conference, ACM.* 13-19.

Xu, L., Harfoush, K. & Rhee, I. 2004. Binary Increase Congestion Control (BIC) for Fast Long-Distance Networks. *Twenty-third Annual Joint Conference of the IEEE Computer and Communications Societies.* 2514-2524.

Xylomenos, G. & Polyzos, G. C. 1999. Internet Protocol Performance over Networks with Wireless Links. *Network, IEEE* 13(4): 55-63.

Yang, Y., Hu, H., Xu, J. & Mao, G. 2009. Relay technologies for WiMAX and LTE-Aadvanced mobile systems. *Communications Magazine* 47(10): 100-105.

Zhang, L., Liu, F., Huang, L. & Wang, W. 2010. Traffic Load Balance Methods in the LTE-Advanced System with Carrier Aggregation. *International Conference on Communications, Circuits and System.* 63-67.

Zorzi, M. & Rao, R. R. 1997. The effect of correlated errors on the performance of TCP. *Communications Letters, IEEE* 1(5): 127-129.

APPENDICES

APPENDIX A

BRIEF DESCRIPTION OF NS-2 SOURCE FILES

File Name	Location in NS-2.3x	Function
tcp.cc	*/ns-allinone-2.3x/ns-2.3x/tcp/*	Including the main routines for slow-start, congestion avoidance and other recovering routines and setting the initial congestion control values.
ns-compact.tcl	*/ns-allinone-2.3x/ns-2.3x/tcl/lib/*	Including source objects which are not derived from TclObject and specify the queue types and settings.
makefile.in	*/ns-allinone-2.3x/ns-2.3x/*	Including specific make for VC++ and explicitly define compilation rules.
ns_tcl.cc	*/ns-allinone-2.3x/ns-2.3x/gen/*	Checking the executable tclsh and configure the wrong version of tclsh may break NS-2 test suites.
FILES	*/ns-allinone-2.3x/ns-2.3x/*	Archiving the roots of all headers, tools, and source files.
tcp-fs.h	*/ns-allinone-2.3x/ns-2.3x/tcp/*	Identify the classes of all TCP source variants.

APPENDIX B

COEFFICIENTS ESTIMATION OF QUADRATIC LAGRANGIAN INTERPOLATION

If there are three points on *xy*-coordinate system can be defined as (x_0, y_0), (x_1, y_1), (x_2, y_2) as shown in the Figure B.1 below:

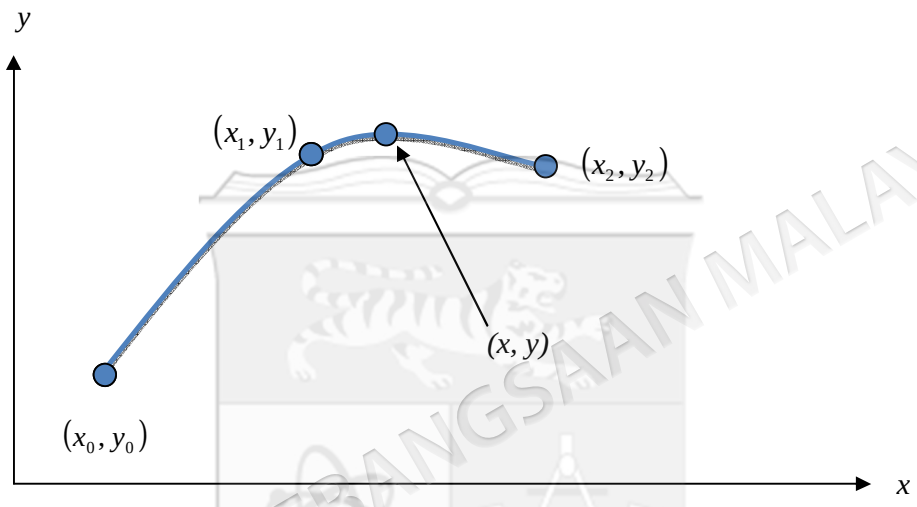


Figure B.1 Quadratic interpolation

The coefficient estimation proposed to use with first, basic, and only condition for the initial value:

$$x_0 \neq x_1 \neq x_2 \tag{1}$$

The condition can be expressed as below, too. Nevertheless, the condition below is more specific:

$$x_0 < x_1 < x_2 \tag{2}$$

To find out a *y* value corresponding to an arbitrary point *x*, we need a function such that:

$$y = f(x) = Ax^2 + Bx + C \tag{3}$$

To find the coefficients *A*, *B*, and *C*, Let write the equation (3) according to the initial values:

$$y_0 = f(x_0) = Ax_0^2 + Bx_0 + C \tag{4}$$

$$y_1 = f(x_1) = Ax_1^2 + Bx_1 + C \tag{5}$$

$$y_2 = f(x_2) = Ax_2^2 + Bx_2 + C \tag{6}$$

To solve the previous equations, it can be written in other arrangement as shown below:

$$y_0 - y_1 = f(x_0) - f(x_1) = A(x_0^2 - x_1^2) + B(x_0 - x_1) \tag{7}$$

$$y_1 - y_2 = f(x_1) - f(x_2) = A(x_1^2 - x_2^2) + B(x_1 - x_2) \tag{8}$$

$$y_0 - y_2 = f(x_0) - f(x_2) = A(x_0^2 - x_2^2) + B(x_0 - x_2) \tag{9}$$

Then, rearranging the Equations (7), (8), and (9), it can obtain:

$$\frac{f(x_0) - f(x_1)}{x_0 - x_1} = A(x_0 - x_1) + B \tag{10}$$

$$\frac{f(x_1) - f(x_2)}{x_1 - x_2} = A(x_1 - x_2) + B \tag{11}$$

$$\frac{f(x_0) - f(x_2)}{x_0 - x_2} = A(x_0 - x_2) + B \tag{12}$$

Subtracting Equation (11) from Equation (10), it can get the following:

$$\frac{f(x_0) - f(x_1)}{x_0 - x_1} - \frac{f(x_1) - f(x_2)}{x_1 - x_{21}} = A(x_0 + x_1 - x_1 - x_2) \tag{13}$$

$$A = \frac{f(x_0)}{(x_0 - x_1)(x_0 - x_2)} \underbrace{- \frac{f(x_1)}{(x_0 - x_1)(x_0 - x_2)} - \frac{f(x_1)}{(x_1 - x_2)(x_0 - x_2)} + \frac{f(x_2)}{(x_1 - x_2)(x_0 - x_2)}}_{\text{R}}$$

By calculate the second and third terms (R) of the equation above:

$$\begin{aligned} R &= \frac{1}{(x_0 - x_2)} \left\{ \frac{f(x_1)}{(x_0 - x_1)} + \frac{f(x_1)}{(x_1 - x_2)} \right\} \\ &= \frac{1}{(x_0 - x_2)} \left\{ \frac{f(x_1)(x_1 - x_2) + f(x_1)(x_0 - x_1)}{(x_0 - x_1)(x_1 - x_2)} \right\} \\ &= \frac{1}{(x_0 - x_2)} \left\{ \frac{f(x_1)(x_1 - x_2 + x_0 - x_1)}{(x_0 - x_1)(x_1 - x_2)} \right\} \\ &= \frac{1}{(x_0 - x_2)} \left\{ \frac{f(x_1)(x_0 - x_2)}{(x_0 - x_1)(x_1 - x_2)} \right\} \\ &= \frac{1}{(x_0 - x_2)} \left\{ \frac{f(x_1)}{(x_0 - x_1)(x_1 - x_2)} \right\} \end{aligned}$$

Then:

$$A = \frac{f(x_0)}{(x_0 - x_1)(x_0 - x_2)} + \frac{f(x_1)}{(x_1 - x_0)(x_1 - x_2)} + \frac{f(x_2)}{(x_2 - x_0)(x_2 - x_1)} \tag{14}$$

The rest of the process is straightforward. If A already estimated, it can find out B by using the Equations (10), (11), or (12). Let use Equation (10), then:

$$B = \frac{f(x_0) - f(x_1)}{(x_0 - x_1)} - \frac{f(x_0)(x_0 + x_1)}{(x_0 - x_1)(x_0 - x_2)} - \frac{f(x_1)(x_0 + x_1)}{(x_1 - x_0)(x_1 - x_2)} - \frac{f(x_2)(x_0 + x_1)}{(x_2 - x_0)(x_2 - x_1)}$$

Then:

$$B = f(x_0)\left\{\frac{1}{(x_0 - x_1)} - \frac{(x_0 + x_1)}{(x_0 - x_1)(x_0 - x_2)}\right\}$$
$$- f(x_1)\left\{\frac{1}{(x_0 - x_1)} + \frac{(x_0 + x_1)}{(x_1 - x_0)(x_1 - x_2)}\right\} - \frac{f(x_2)(x_0 + x_1)}{(x_2 - x_0)(x_2 - x_1)}$$

$$B = \frac{f(x_0)}{(x_0 - x_1)}\left\{\frac{(x_0 - x_2 - x_0 - x_1)}{(x_0 - x_2)}\right\}$$
$$- \frac{f(x_1)}{(x_1 - x_0)}\left\{\frac{(-x_1 + x_2 + x_0 + x_1)}{(x_1 - x_2)}\right\} - \frac{f(x_2)(x_0 + x_1)}{(x_2 - x_0)(x_2 - x_1)}$$

Then:

$$B = -\frac{f(x_0)}{(x_0 - x_1)(x_0 - x_2)}(x_1 + x_2) - \frac{f(x_1)}{(x_1 - x_0)(x_1 - x_2)}(x_0 + x_2)$$
$$- \frac{f(x_2)}{(x_2 - x_0)(x_2 - x_1)}(x_0 + x_1)$$

Finally:

$$B = -\left[\begin{array}{l}\dfrac{f(x_0)(x_1 + x_2)}{(x_0 - x_1)(x_0 - x_2)} \\ + \dfrac{f(x_1)(x_0 + x_2)}{(x_1 - x_0)(x_1 - x_2)} + \dfrac{f(x_2)(x_0 + x_1)}{(x_2 - x_0)(x_2 - x_1)}\end{array}\right] \quad (15)$$

The last coefficient *C* is neglected due to it not takes into account as explained. To obtain the values of *A* and *B* in term of *cwnd* and *ssthresh*, it will suppose the estimation happen for each RTT, and then we can get the following:

$$x_0 = 1,\ f(x_0) = 0$$
$$x_1 = 2,\ f(x_1) = cwnd\ (t)$$
$$x_2 = 3,\ f(x_2) = ssthresh$$

By substituting the assumed parameters in Equation (14) and Equation (15), it can find the final estimation of *A* and *B* as following:

$$A = \frac{-1}{ssthresh}$$

$$B = 2$$

Then:

$$y = f(x) = \left(\frac{-1}{ssthresh}\right)x^2 + (2)x$$

Then, it can write the final formula, which based on quadratic Lagrangian interpolation as shown in below:

$$cwnd\ (t+\tau) = \frac{-1}{ssthresh}(cwnd\ (t))^2 + 2cwnd\ (t)$$

This formula represents the required interpolated increment of TCP slow-start to approximate the congestion window towards *ssthresh* when the congestion control unable to duplicate the window size any more.

APPENDIX C

DIFFERENCE BETWEEN LINEAR AND QUADRATIC INTERPOLATION

Linear Interpolation	Quadratic Lagrangian Interpolation
4.5000	4.8000
4.7500	4.9920
4.8750	4.9999
4.9375	
4.9687	
4.9843	
4.9921	
4.9966	
4.9986	
4.9990	
4.9995	
4.9997	
4.9998	
4.9999	

The table shows the difference in number of required iterations to reach *ssthresh* when using duplicating window approach in slow-start phase. In this example, *ssthresh* set to 5 packets only and the interpolation process start after *cwnd* duplicated from 1, 2, to 4 packets. Then, the interpolation will cover the zone between 4-5 packets. The estimated values showed that linear interpolation requires to 14 iterations to reach *ssthresh* while the quadratic requires only 3 iterations. Figure C.1 shows that slow-start with quadratic interpolation reaches *ssthresh* (128 packets) in lesser number of iterations than slow-start with linear interpolation.

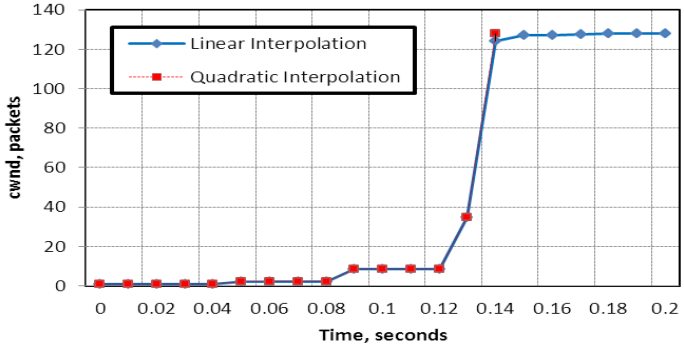


Figure C.1 Linear and quadratic interpolation comparison

APPENDIX D

LIST OF PUBLICATIONS

JOURNALS

1. Abed, G. A., Ismail, M. & Jumari, K. 2010. Modeling and Performance Evaluation of LTE Networks with Different TCP Variants. *International Journal of Information and Communication Engineering* 6(4): 227-232.

2. Abed, G. A., Ismail, M. & Jumari, K. 2011. Behaviour of *cwnd* for TCP source variants over parameters of LTE networks. *Information Technology Journal* 10(3): 663-668.

3. Abed, G. A., Ismail, M. & Jumari, K. 2011. Architecture and Functional Structure of Transmission Control Protocol over Various Networks Applications. *Journal of Theoretical and Applied Information Technology* 34(1):71-79.

4. Abed, G. A., Ismail, M. & Jumari, K. 2011. Characterization and observation of (transmission control protocol) TCP-Vegas performance with different parameters over (Long term evolution) LTE networks. *Scientific Research and Essays* 6(9): 2003-2010.

5. Abed, G. A., Ismail, M. & Jumari, K. 2011. Distinguishing Employment of Stream Control Transmission Protocol over LTE-Advanced Networks. *Research Journal of Information Technology* 3(4): 207-214.

6. Abed, G. A., Ismail, M. & Jumari, K. 2011. A Survey on Performance of Congestion Control Mechanisms for Standard TCP Versions. *Australian Journal of Basic and Applied Sciences* 5(12): 1345-1352.

7. Alrashdan, M., Ismail, M., Jumari, K., Abed, G. A. & Hassan, S. 2011. Performance Comparison Analysis of SSHMIPv6 and HMIPv6 Lost Packets using Network Simulator NS-2. *Information Technology Journal* 10(10): 1989-1993.

8. Abed, G. A., Ismail, M. & Jumari, K. 2012. The Evolution to 4G Cellular Systems: Architecture and Key Features of LTE-Advanced Networks. *IRACST –International Journal of Computer Networks and Wireless Communications (IJCNWC)* 2(1): 21-26.

9. Abed, G. A., Ismail, M. & Jumari, K. 2012.Experimented Goodput Measurement of Standard TCP Versions over Large-Bandwidth Low-Latency Bottleneck. *Journal of Computing* 4(5):212-216.

10. Abed, G. A., Ismail, M. & Jumari, K. 2012. Exploration and Evaluation of Traditional TCP Congestion Control Techniques. *Journal of King Saud University - Computer and Information Sciences-Elsevier* (24): 145-155.

11. Abed, G. A., Ismail, M. & Jumari, K. 2012. Improvement of TCP Congestion Window over LTE-Advanced Networks. *International Journal of Advanced Research in Computer and Communication Engineering* 1(4): 185-192.

12. Abed, G. A., Ismail, M. & Jumari, K. Comparative Performance Investigation of TCP and SCTP Protocols over LTE/LTE-Advanced Systems. *International Journal of Advanced Research in Computer and Communication Engineering* 1(6): 466-471.

13. Abed, G. A., Ismail, M. & Jumari, K. 2012. A Realistic Model and Simulation Parameters of LTE-Advanced Networks. *International Journal of Advanced Research in Computer and Communication Engineering* 1(6): 461-465.

PROCEEDINGS

14. Abed, G. A., Ismail, M. & Jumari, K. 2011. Traffic Modeling of LTE Mobile Broadband Network Based on NS-2 Simulator. *Third International Conference on Computational Intelligence, Communication Systems and Networks (CICSyN)*, Bali-Indonesia, 120-125.

15. Abed, G. A., Ismail, M. & Jumari, K. 2011. Appraisal of Long Term Evolution System with Diversified TCP's. *Fifth Asia Modelling Symposium (AMS)*, Manila-Philippine, 236-239.

16. Abed, G. A., Ismail, M. & Jumari, K. 2011. Implementation of New TCP Congestion Control Mechanism over Long Term Evolution Advanced Networks. *Regional Engineering Postgraduate Conference (EPC)*. Kuala Lumpur- Malaysia.

17. Abed, G. A., Ismail, M. & Jumari, K. 2011. Influence of Parameters Variation of TCP-Vegas in Performance of Congestion Window over Large Bandwidth-Delay Networks. *17th Asia-Pacific Conference on Communications (APCC)*, Sabah-Malaysia, 434-438.

18. Abed, G. A., Ismail, M. & Jumari, K. 2011. A Comparison and Analysis of Congestion Window for HS-TCP, Full-TCP, and TCP-Linux in Long Term Evolution System Model. *IEEE Conference on Open Systems (ICOS)*, Langkawi- Malaysia, 358-362.

19. Abed, G. A., Ismail, M. & Jumari, K. 2012. Links Interfacing Demonstration of LTE-Advanced Networks Using NS-2 Modeller. *Third International Conference on Intelligent Systems, Modelling and Simulation (ISMS)*, Sabah-Malaysia, 669-673.

20. Abed, G. A., Ismail, M. & Jumari, K. 2012. Integrated Approaches to Enhance TCP Performance over 4G Wireless Networks. *IEEE Symposium on Computers & Informatics. Penang-Malaysia (ISCI)*, Penang-Malaysia, 154-158.