TCP Versus UDP Performance In Term Of Bandwidth Usage

A thesis submitted to the Faculty of Information Technology in partial fulfilment of the requirement for the degree
Master of Science (Information Technology)
Universiti Utara Malaysia

By
Mostfa M. Kaytan

PERMISSION TO USE

In presenting this thesis in partial fulfilment of the requirements for a postgraduate degree from University Utara Malaysia, I agree that the University Library may make it freely available for inspection. I further agree that permission for copying of this thesis in any manner, in whole or in part, for scholarly purposes may be granted by my supervisor, in his absence, by the Dean of the Faculty of Information Technology. It is understood that any copying or publication or use of this thesis or parts thereof for financial gain should not be allowed without my written permission. It is also understood that due recognition shall be given to me and to University Utara Malaysia for any scholarly use which may be made of any material from my thesis.

Request for permission to copy or to make use of material in this thesis, in whole or in part should be addressed to:

Dean of Research and Postgraduate Studies

College of Arts and Sciences

Universiti Utara Malaysis

06010 UUM Sintok

Kedah Darul Aman

Malaysia

ABSTRACT

This project is mainly about how to establish User Datagram Protocol (UDP) and Transmission Control Protocol (TCP) connection in the same network simulation. For that, we will be using four types of TCP which are TCP Tahoe, TCP Reno, TCP NewReno and TCP Vegas. From there, we are going to differentiate them in term bandwidth usage and define how it works and describes several effect that occurred when its work together. In order to create the topology and run the protocols, we use Network Simulator 2 (NS2) to create and run the coding. To run the codes, we use command which use a few code in running the coding. Then we will get a topology, which is the flow of the packet within the source and destination, base on the coding. A graph also appears after the command.

ACKNOWLEDGMENTS

In the name of Allah, Allah says:

((Work; so Allah will see your work and (so will) His Messenger and the believers ;))

(Al-Quran: Tawba-105)

Conducting this project marks the end of an interesting and eventful journey. It could not have been achieved without the academic professional and personal support of the following people.

Firstly, I would like to extend special thanks to my supervisor, Assoc. Prof. Hatim Mohamad Tahir, of the Faculty of Information Technology, University Utara Malaysia (UUM) for tirelessly offering his encouragement, wisdom and experience, who provided me with constant guidance and constructive criticism throughout all stages of my research. I would like to thank my evaluator Mr. Rosmadi B Bakar for his suggestion and his encouragement. I would like to thank my friend Dr. Mohammed M. Kadhum for his consultation and his suggestion regard the research results. As well, thanks to the Ministry of Higher Education Iraq for the financial support awarded to me.

Secondly, I am grateful to all FTM lecturers for their guidance and unconditional support, also for all UUM staff that provided me with a warm hospitality and assistance during my time in Sintok.

Thirdly, Much appreciation to my friends, who have helped me to get accustomed to the culture and traditions, and have showed me a magnificent meaning of friendship at every crossroad. Their warmth and empathy will ever never be forgotten.

Finally, a very big thank must go to all my family members for their immeasurable support. I wish to acknowledge my parents for their unwavering support and confidence in me. There are not enough words for me to express my feelings of deep appreciation to my parents. I would like to dedicate this thesis to my wife and daughters who lovely encouraged and supported me through all my study. Many thanks as well as to my brothers Bsam and Ali for their assistances and do all my business in Iraq during the time I m doing my Master.

For those all, I would like to say

"شكرا لثقتكم بي ودعائكم لي ودعمكم اللا محدود"

Contents

LIST OF FIGURE

ABBREVIATIONS

TCP	Transport Control Protocol
UDP	User Datagram Protocol
FTP	File Transfer Protocol
CBR	Constant Bit Rate
NS	Network Simulation
NAM	Network Animator
TCL	Tool Command Language
OTCL	Object extension of TCL
HTTP	Hypertext Transfer Protocol
POP	Post Office Protocol
SMTP	Simple Mail Transfer Protocol
ATM	Asynchronous Transfer Mode
DSSS	Direct-Sequence Spread Spectrum.

CHAPTER ONE

INTRODUCTION

1.1 Introduction

Transmission Control Protocol/Internet Protocol (TCP/IP), the most common of all network protocol suites, used for communication on the Internet. TCP/IP is a hierarchical protocol made up of interactive layers (as shown in Figure I) each layer has a specific functionality. (Ross, 2008)

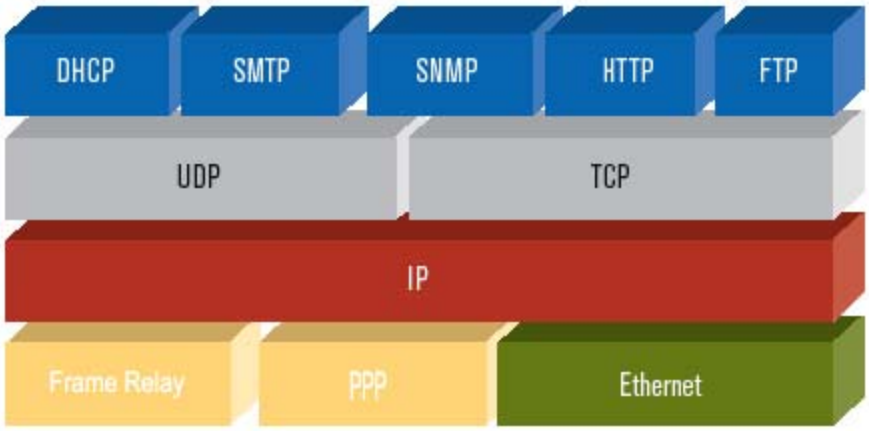


Figure 1.1 TCPIIP Protocol Suite

According to (Ross,2008) application layer are placed at the top of TCP / IP stack, it defines protocols such as (FTP, HTTP, Telnet and so on) for application communication. These protocols are acting as interface for the actual application program. The transport layer follows the application layer. TCP/IP makes available two distinct transport layer protocols to the application layer: Transmission Control Protocol (TCP) and User Datagram Protocol (UDP). The transport layer follows the application

layer. TCP/IP makes available two distinct transport layer protocols to the application layer: Transmission Control Protocol (TCP) and User Datagram Protocol (UDP). The most popular transport layer protocols which have been used in the internet consist of two, TCP (Transmission Control Protocol) and UDP (User Datagram Protocol). TCP is probably the most commonly used protocol, simply because it is used for so many applications such as HTTP, POP, SMTP, etc. TCP is a protocol which guarantees that the receiver will receive exactly what the sender sent - there will be no errors, it will be in the correct order, everything will work just fine (Kurose and Ross, 2005).

Despite the recent explosion in availability of broadband Internet access, the majority of home users have relatively small-bandwidth links in comparison with the sites hosting desired content (Zakhor, 2003).

Applications using TCP, such as web-browsers, ftp, and various P2P programs, dominate most of the Internet traffic today. In many cases the last-hop access links are bottlenecks due to their limited bandwidth capability with users running many simultaneous network applications (Zakhor, 2003).

UDP is similar to TCP in that it is a protocol for sending and receiving packets across a network, but with two major differences. First, it is connection less. This means that one program can send off a load of packets to another, but that's the end of their relationship. The second might send some back to the first and the first might send some more, but there's never a solid connection. If one just stops sending packets that's fine

In this project we will use network simulation 2(NS2) to create topology and run codes. We believe that once the basics of NS2 are grasped, the readers can go through

other documentations, and readily understand the details of other NS2 components (Hossain, 2009).

According to (Postel, 1981) the transport protocol used by the majority of Internet traffic is the Transmission Control Protocol (TCP) which lacks the ability to distinguish between packet losses due to errors and those due to network resource contention (congestion).

The protocol's goal is only to recover from packet losses by retransmission, so distinguishing the causes of losses isn't required. By design, standard TCP implementations assume congestion as the cause of all packet losses and slow their sending rates in response (AlIman, Paxson,and Stevens, 1999).

According to (Jacobson, 1988) the motivation for slowing down for perceived congestion was that, at the time, there were several congestion collapse events during which the Internet was unusable, and so fixing TCP to send more conservatively in the presence of losses was important. This was not an indefensible decision on the TCP designers' part given that at the time most Internet links were wired, so it was a safe assumption that congestion caused the vast majority of packet losses. This assumption, however, leads to suboptimal performance when TCP is used over networks where packets are dropped due to corruption and in which corruption losses are independent of the congestion level. Since most applications in common use today (web, email, file transfer) run over TCP, we see them perform more poorly than necessary over wireless networks (Eddy, 2004).

1.2 Work Background

1.2.1 Transmission control protocol (TCP)

TCP is the dominant protocol used in the Internet today, since it is the transport protocol that is responsible for the transmission of around 90% of the Internet traffic (Allman and Falk, 1999). The IETF is the main standardization organization that is concerned with TCP, and unlike other standardization Organization (ATM), all their standards are free and available on-line According to (Forouzan, 2000).

TCP has several responsibilities. Firstly, it is responsible for creating a process-to-process (program-to-program) communication and in order to accomplish this, TCP uses port numbers, Three-Way Handshaking for connection establishment, and Four-Way Handshaking for connection termination. Secondly, the responsibility of TCP is to create a flow and error control mechanism at the transport layer. TCP uses a sliding window protocol to achieve flow control. It uses the acknowledgment packet, time-out, and retransmission to achieve error control. TCP is also called a connection-oriented, reliable transport protocol and thus adds connection-oriented and reliability features to the services of IP.

According to (Huston and Telstra, 2009) Although TCP attempts to discover the delay bandwidth product of the connection, and attempts to automatically optimize its flow rates within the estimated parameters of the network path, some estimates will not be accurate, and the corresponding efforts by TCP to optimize behavior may not be completely successful.

The main responsibility of TCP is to avoid congestion in the network and to adapt the transmission rate of packets to the available bandwidth. Modern implementations of TCP

contain four congestion control algorithms: slow start, congestion avoidance, fast retransmit, and fast recovery (Stevens, 1997). This end-to-end congestion control mechanisms of TCP have been a critical factor in the robustness of the Internet (Floyd and Fall, 1999).

TCP is a reliable data transfer protocol used widely over the Internet for numerous applications, from FTP to HTTP. The current implementation of TCP Reno/NewReno mainly includes two phases: Slow-start and Congestion avoidance (Wang, et al, 2002).

1.2.2 TCP Reno

New versions of TCP have been proposed to improve data transmission performance. The first version of TCP defined the basic structure of TCP, namely, the window-based flow control scheme and a coarse grain timeout timer. The second version, TCP Tahoe, added the congestion avoidance scheme. Two years later saw the introduction of a third version, TCP Reno, which retained all the enhancements in TCP Tahoe (fast retransmit, slow start, and congestion avoidance), but also incorporated a new algorithm, the fast recovery algorithm (Hassan and Jain, 2004).TCP Reno has become the most popular version of TCP today. However this version predicts available bandwidth by detecting the packet loss, which causes network congestion and unfair bandwidth usage. The main reason TCP Reno remains in use is because it has an aggressive control scheme that expands and covers more bandwidth until the transmitted packets are lost (Lai and Yao, 2000).

According to (Kurose and Ross, 2005) Fast Recovery algorithm in TCP Reno cancels the slow start phase after a triple duplicate ACK and remains in the congestion

avoidance phase while dropping the congestion window by half. The philosophy behind canceling slow start in this case is that even though a packet has been lost, the arrival of three duplicate ACKs indicates that some segments have been received at the sender. Thus, unlike the case of a timeout, the network is showing itself to be capable of delivering at unlike the case of a timeout; the network is showing itself to be capable of delivering at least some segments, even if other segments are being lost due to congestion.

1.2.3 User datagram protocol (UDP)

UDP is a simple, connectionless transport protocol (Stevens, 1994) .UDP does just about as little as a transport protocol can. Aside from the multiplexing / demultiplexing function and some light error checking, it adds nothing to IP. In fact, if the application developer chooses UDP instead of TCP, then the application is talking almost directly with IP. UDP takes messages from application process, attaches source and destination port number fields for the multiplexing/demultiplexing service, adds two other fields of minor importance, and passes the resulting "segment" to the network layer. The network layer encapsulates the segment into an IP datagram and then makes a best-effort attempt to deliver the segment to the receiving host. If the segment arrives at the receiving host, UDP uses the port numbers and the IP source and destination addresses to deliver the data in the segment to the correct application process. Note that with UDP there is no handshaking between sending and receiving transport layer entities before sending a segment. For this reason, UDP is said to be connectionless (Kurose and Ross, 2005).

1.2.4 Network Simulation-2(NS2)

The Network Simulation Experiments Manual. As networking systems have become more complex and expensive, hands-on experiments based on networking simulation have become essential for teaching the key computer networking topics to students and professionals. The simulation approach is highly useful because it provides a virtual environment for an assortment of desirable features such as modeling a network based on specified criteria and analyzing its performance under different scenarios (Peterson and Davide, 2003).

NS2 is an open-source event-driven simulator designed specifically for research in computer communication networks. Since its inception in 1989, NS2 has continuously gained tremendous interest from industry, academia, and government. Having been under constant investigation and enhancement for years, NS2 now contains modules for numerous network components such as routing, transport layer protocol, application, etc. To investigate network performance, researchers can simply use an easy-to-use scripting language to configure a network, and observe results generated by NS2. Undoubtedly, NS2 has become the most widely used open source network simulator, and one of the most widely used network simulators (Hossain, 2009).

Ns is an object-oriented simulator written in C++ with an object-oriented TCL (OTCL) interpreter as a front-end. C++ is used for detail protocol implementation and OTCL for simulation configuration. One drawback of combining two languages is that debugging becomes more complicated than with one language alone (Mattsson, 2004).

According to (Aggarwal, 2003) the figure 1.2 shows the overall functioning of NS-2 simulation tool. NS-2 is an object-oriented network simulator written in C++ and OTCL. Simulation

topologies are written in TCL (tool command language) language, which are linked with background C++ simulator modules using OTCL linkage. NS is primarily useful for simulating local and wide area networks. As it can be observed that the functionality changes have to be carried out on ns-classes which are written in C++.

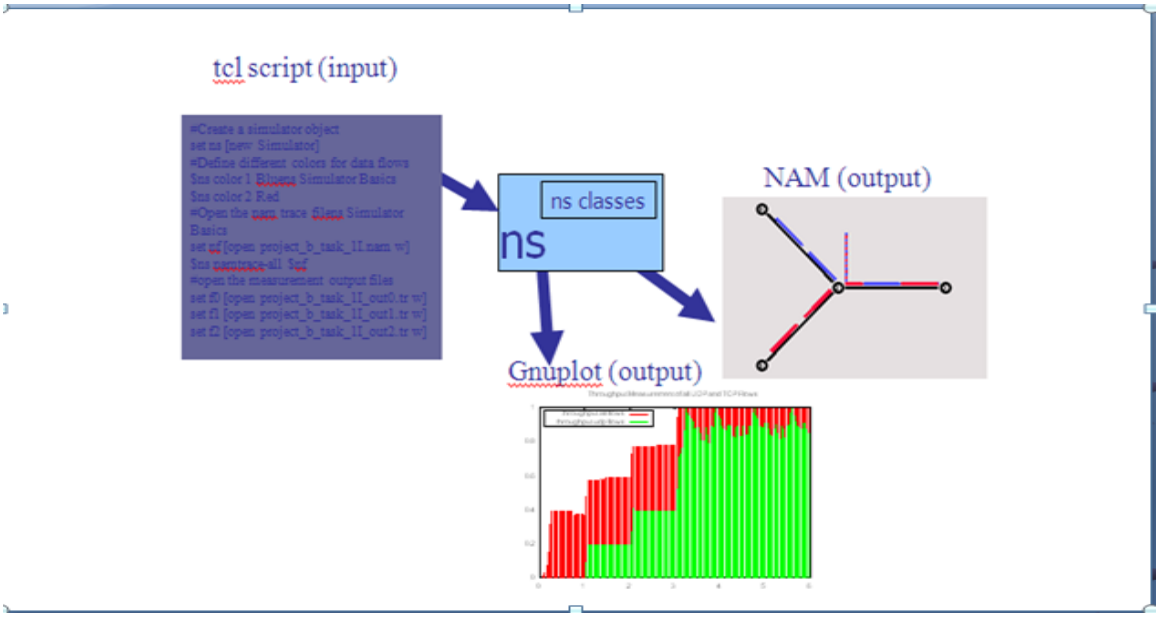


Figure 1.2:Discreet Event Scheduler

The basic simulator objects that ns include are Nodes, Links, Agents, and Applications. Nods and Links define the network topology. Agent represent end points where network layer packets are constructed or consumed and are commonly used to implement protocols at various levels. Applications are traffic sources that send and receive data. The objects are connected to each other in a layered fashion as seen in figure1.3 (Chohan, 2006).

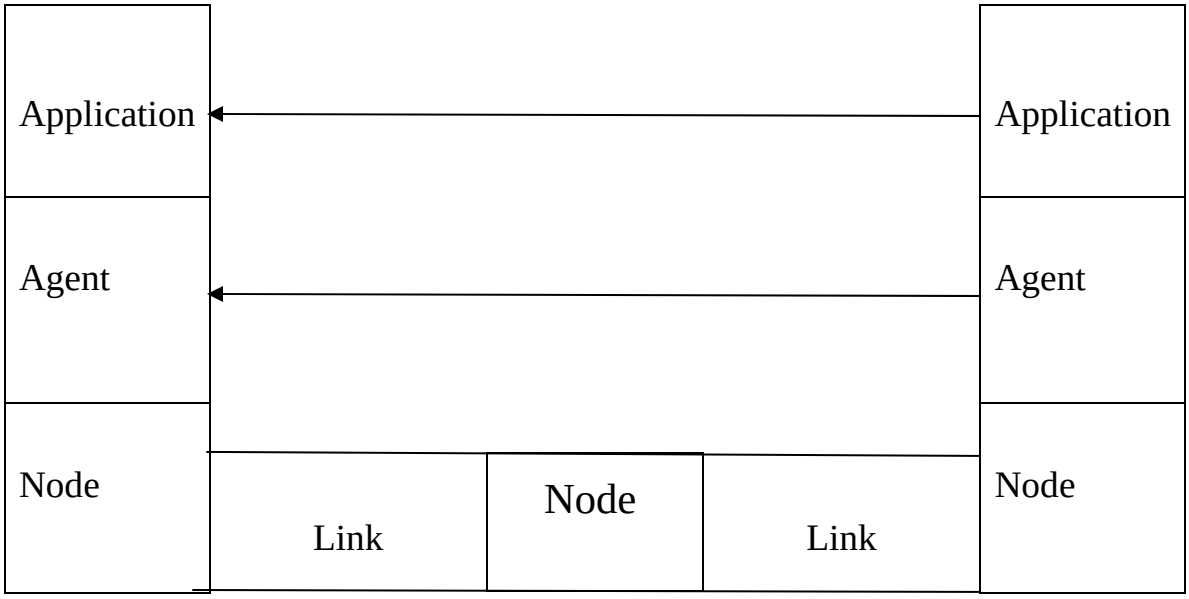


Figure 1.3- The basic simulation objects in ns and their interconnections

1.3 Problem Statements

Computer networks such as Internet are costly in terms of building and operating. Therefore it is preferred to study the network protocols performance in simulated environment. The performance of the network may not be as good as the one estimated before the installation and this may sometimes lead to changing of the network characteristics. Network simulators such as NS2, GLOMOSIM, and SWANS help in estimating the performance of network protocols before implementing theses protocols in real environment. In this project we use NS2 to study the TCP and UDP protocols performance in terms of bandwidth usage. This study helps in taking a proper decision about actual establishment of the network.

1.4 Research Questions

This project will attempt to answer the following questions:

1. How much bandwidth that usage by protocols in transport layer, the transmission Control Protocol or User Datagram Protocol when they share same link and which one consume the bandwidth more than other?
2. How we can measure the consuming bandwidth in each protocol in transport layer by using the network simulation?

1.5 Research Objectives

Networking systems have become more complex and expensive, hands-on experiments based on networking simulation have become essential for teaching the key computer networking topics to students and professionals. The simulation approach is highly useful because it provides a virtual environment for an assortment of desirable features such as modeling a network based on specified criteria.

The implement of this thesis will try to achieve the below objectives:

1. To identify which protocol is consuming the bandwidth more than the other in transmission layer.
2. To measure the consuming bandwidth.
3. To analyze the protocols theoretically and through simulation.

1.6 Scope and Limitations

This research will be limiting by the following boundaries:

1.The establish connection efficiency. This will be when we created the code for TCP and UDP agent and sink

2. Create the topology and run the protocols, this refer to simulation arrangement.

3. Create and run the coding, when we run our code up on NS2

4. We are using network simulator to define the result.

1.7 Significant of Study

This study tries to compare the evaluation of performance TCP and UDP in usage bandwidth. In our project we will use the simulation network. Performance evaluation is a critical component of systems research that allows the evaluation of new ideas, identification of problems and bottlenecks and optimization of existing systems. There are three general approaches to performance evaluation:

1. **Prototyping:** build it (or a scaled down version of it) and see how it works.
2. **Analytical modeling:** build a mathematical model of it and use it to analyze the system.
3. **Simulation:** build a software model of the system. Prototyping is often not feasible, or time consuming especially for large scale systems.

It also provides limited controllability and observability. Similarly, analytical modeling cannot capture highly complex systems. Thus, simulation has emerged as an attractive alternative that is heavily used in performance evaluation of computer systems.

Network simulation enables us to predict behavior of a large-scale and complex network system such as the Internet at low cost under different configurations of interest and over long periods. Many network simulators, such as NS-2, SSFNet, Opnet, Qualnet, etc., are widely available. We will use NS-2 for this project. NS-2 is a discrete event

simulator written in C++, with an OTCL interpreter shell as the user interface that allows the input model files (TCL scripts) to be executed. Most network elements in NS-2 are developed as classes, in object-oriented fashion. The simulator supports a class hierarchy in C++, and a very similar class hierarchy in OTCL. The root of this class hierarchy is the TCLObject in OTCL. Users create new simulator objects through the OTCL interpreter, and then these objects are mirrored by corresponding objects in the class hierarchy in C++. NS2 provides substantial support for simulation of TCP, routing algorithms, queuing algorithms, and multicast protocols over wired and wireless (local and satellite) networks, etc. It is freely distributed, and all source code is available. In the first project, you will not need to worry about the internals of NS-2.

1.8 Definition of Terms

For clarity of understanding the following terms will be defined. The definitions given will be operational definitions that are pertinent to this study and the software used to conduct the research.

- **TCP:** (Transmission Control Protocol) is the main transport protocol utilized In IP networks. The TCP protocol exists on the Transport Layer of the OSI Mode.
- **TCP Reno:** TCP Reno advanced the Fast Transmit, where three duplicate acknowledgments signaled a re-transmittance without a timeout, with Fast Recovery. Fast Recovery meant that once a certain threshold of acks were received the window size was decreased by half, rather than starting over with slow start. Only during timeout does it go back into slowstart(Chohan, 2006).

- **TCP New Reno:** TCP New Reno responded better compared to TCP Reno with the interpretation of Partial acks as indications of packet loss and does not remove it out from the Fast Recovery phase. Because the timeout timer is renewed when acks are received, New Reno is able to maintain high throughput (Chohan, 2006).

- **TCP Vegas:** TCP Vegas uses packet delay as an indication of congestion. During the situation when a duplicate ack is received the timestamp for the ack is compared to a timeout value. If the timestamp is greater than the timeout value then Vegas will retransmit rather than waiting for three duplicate acks(Chohan,2006).

- **UDP:** User Datagram Protocol is part of the Internet Protocol suite, using which, programs running on different computers on a network can send short messages known as Datagrams to one another. UDP can be used in networks where TCP is traditionally used.
- **Bandwidth:** In computer networks, bandwidth is often used as a synonym for data transfer rate - the amount of data that can be carried from one point to another in a given time period (usually a second).

- **Simulation:** is the discipline of designing a model of an actual or theoretical physical system, executing the model on a digital computer, and analyzing the execution output.
- **NS-2:** is a discrete event simulator targeted at networking research. Ns provides substantial support for simulation of TCP, routing, and multicast protocols over wired and wireless (local and satellite) networks.

1.9 Organization of the Thesis

Chapter two gives an overview of transmission control protocol and user datagram protocol, network simulation-2 components, and some related work. Chapter three describes a methodology that used to setup a test-bed environment and implements the experiments. Chapter four present the finding and analysis of the experiments results that done. Chapter five gives a conclusion and suggested for future works.

CHAPTER TWO

LITERATURE REVIEW

2.1 Introduction

A common form of bandwidth allocation is to allow weighted fair sharing of bandwidth among different applications. For instance, a user may decide to set aside one fourth of the available bandwidth for a peer-to-peer sharing application, another fourth of the bandwidth for an ftp download, and to allocate the remaining bandwidth for web browsing. However, there are cases in which an application requires a minimum guaranteed bandwidth allocation regardless of current link capacity. Multimedia streaming applications are a prime example of such applications, since they generally require constant play out at a particular rate, and are sensitive to fluctuations in the received rate. Many online games also have strict minimal bandwidth requirements for adequate usability. These applications can suffer from severe performance degradations if they fail to receive a minimum desired bit-rate. Hence it may be desirable to specify a minimum bit-rate for these applications regardless of the total link capacity, and to perform weighted sharing of any remaining bandwidth (Zakhor, 2003).

Pervasive computing or ubiquitous computing is a new concept in computer science. Several attempts have been made to define what pervasive computing is. A rough definition of pervasive computing could computers be everywhere at anytime. The term Desegregated Computing can be used as well to denote devices with small processors embedded and interconnected, such as monitors, projectors, printers, input devices, PDAs, and phones. Currently, resources such as mail boxes, printers, and disk

space are readily available at any network. Devices such as PDAs and cell phones are capable of accessing them. The growth in the popularity of this kind of portable devices has increased the need to develop technology for connecting to these resources. Further study in this area will lead us eventually to the definition of pervasive computing stated above computers everywhere at any time (Barbeau, 2000).

2.2 Interaction between TCP and UDP flows in Wireless

A link in a wireless network is an abstract notion defined by the connectivity between two nodes. Data sent by a node might be overheard by many other nodes or even be perceived as only interference. The presence of a link in a wireless network might be intermittent (e.g., due to mobility), links have varying quality (and hence bandwidth) and unlike wired networks, where congestion is in general caused by overfull queues, there may also be contention for the spatial reuse of the ether, causing congestion in the network. The TCP feedback loop is responsible for adapting the sender data rate in response to, e.g., congestion. However, this end-to-end congestion control mechanism has reduced efficiency in wireless networks because transmission is inherently broadcast. Furthermore, there are different ranges for uncast radio transmission, broadcast radio transmission and interference. Even if the offered data rate is adapted to the bottleneck in a network path, transmissions might still contend and interfere with other. TCP has been designed to be fair between competing flows, i.e., TCP backs off in a way that an equilibrium is reached where all competing flows get an equal chunk of the available bandwidth at a bottleneck. CBR UDP flows are not rate adaptable and lacks congestion control. However, moderate rate flows, e.g., MP3 or voice streams could be considered

marginal because they use relatively little bandwidth. In the presence of UDP flows, TCP should probe the data path by increasing its congestion window until loss occurs. In that case it should back off and not use more bandwidth than what is left after the UDP flow. The work in this paper examines the efficiency of the TCP rate adaptation in the presence of UDP flows (Rohner, et al, 1998).

2.3 Multi-hop UDP with TCP flow

In this scenario we increase the complexity by adding an extra node 3, which is mobile and sends a TCP flow to node 0. Node 3 starts at the far left position in Figure 2.1, outside the transmission range of node 1. Ten seconds into the scenario, node 3 starts moving toward node 0 and then when reaching that position (at time 38) it ends its movement.

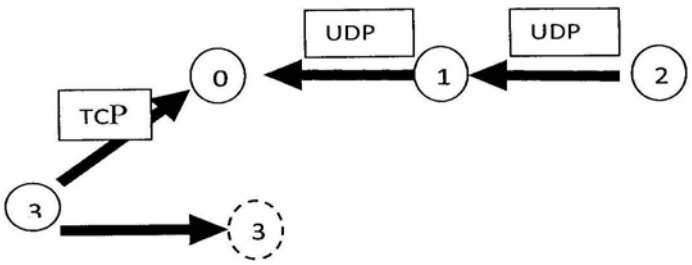


Figure 2.1: The TCP vs UDP scenario

The purpose of this scenario is to see how TCP adapts its send rate (if at all) when moving to the position close to node 0, where it is potentially more affected by channel contention from both node 1 and node 2. We also want to see how the UDP flow from node 2 to node 0 is affected by the increased contention. Over five test runs, the average UOP delivery ratio was 97.4 %, a slight decrease from the previous scenario. The average

TCP throughput was 1.34 Mbit/s. At first this number seemed unexpectedly low. However, considering the overhead of TCP, feedback loop (packets in both directions and extra header size) and the contention with the UOP flow from both links (between nodes 2 and 1 and 1 and 0, respectively), the number 1.34 Mbit/s might not seem so unreasonable. Understanding the exact reasons for this particular throughput requires further investigation.

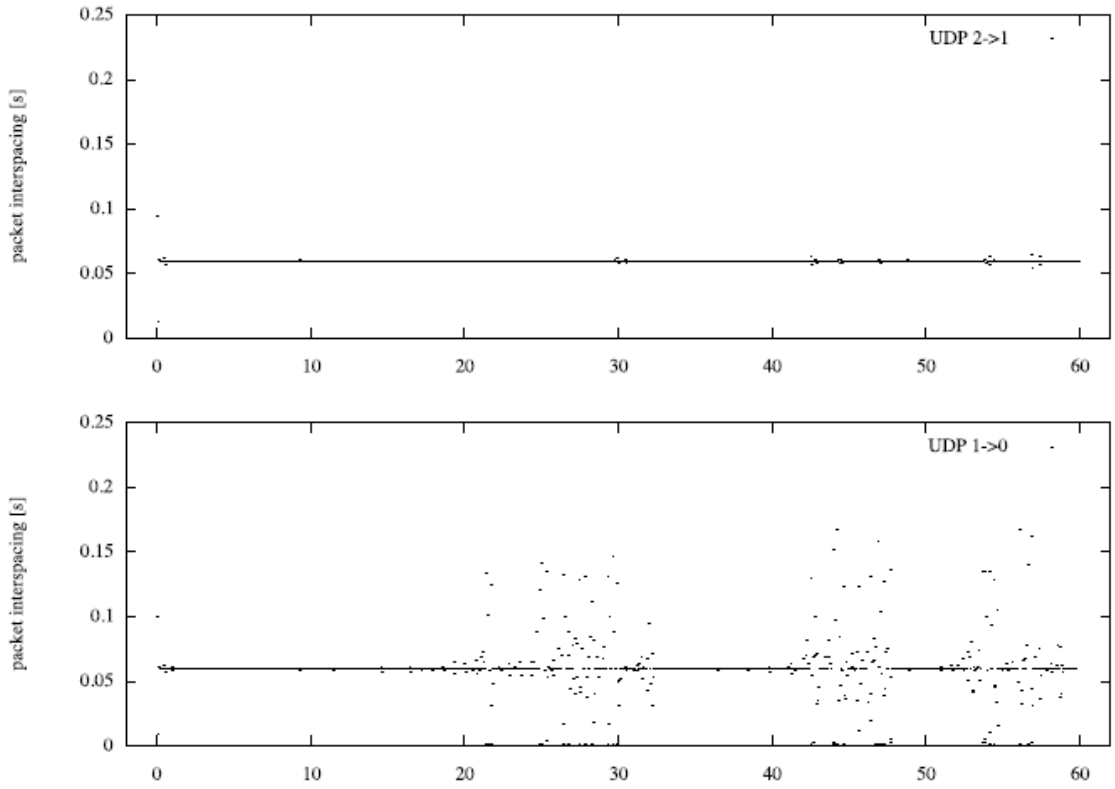


Figure 2.2 Example of UDP inter packet deliverytime in the static multi-hop scenario.

(Xu and Saadawi, 2001)

As we shown in Figure 2.2 we see the UDP inter packet delivery time on the link between node 2 and 1 and 1 and 0 respectively, along with the TCP time sequence graph for the TCP flow between node 3 and 0, for one of the experiments. Although variations are apparent throughout all experiments, Figure 2.2 is representative for a typical test run.

It can be seen that most of the packet loss on the UDP flow again occurs on the link between node 1 and 0. After node 3 has moved and is in direct contact with node 1, the jitter between node 2 and 1 is also more apparent, indicating that the 3. TCP flow is interfering with the UDP flow at that link. An interesting observation can also be made at time 50, where there is an increase in the TCP throughput. This increase comes at a cost, inducing a higher amount of jitter at the link between node 2 and1. This behavior is persistent in most of the experiments. It is not clear why TCP increases its throughput in this situation. One possible explanation could be the capture effect(Xu and Saadawi, 2001)

2.4 TCP Vegas vs. TCP Reno

According to (Jeonghoon et al, 2004) they analyzed the performance of TCP Vegas in comparison with TCP Reno. They reached to the TCP Vegas does lead to a fair allocation of bandwidth and explained some of its other characteristics, also demonstrated through both analysis and simulations that TCP Vegas does not suffer from the delay bias as TCP Reno does. TCP Vegas achieves better performance than TCP Reno since its bandwidth estimation does not rely on packet losses in order to estimate the available bandwidth in the network. However, when competing with other TCP Reno connections, TCP Vegas gets penalized due to the aggressive nature of TCP Reno and TCP Vegas does lead to a fair allocation of bandwidth and explained some of its other characteristics. They demonstrated through both analysis and simulations that TCP Vegas does not suffer from the delay bias as TCP Reno does. TCP Vegas achieves better performance than TCP Reno since its bandwidth estimation does not rely on packet

losses in order to estimate the available bandwidth in the network. However, when competing with other TCP Reno connections, TCP Vegas gets penalized due to the aggressive nature of TCP Reno (Jeonghoon M. et al, 2004).

2.5 TCP Startup Performance in Large Bandwidth

Next generation networks with large bandwidth and long delay pose a major challenge to TCP performance, especially during the startup period. In this paper we evaluate the performance of TCP Reno/Newreno, Vegas and Hoe's modification in large bandwidth delay networks. We propose a modified Slow-start mechanism, called Adaptive Start (Astart), to improve the startup performance in such networks. When a connection initially begins or re-starts after a coarse timeout, Astart adaptively and repeatedly resets the Slow-start Threshold (ssthresh) based on an eligible sending rate estimation mechanism proposed in TCP Westwood. By adapting to network conditions during the startup phase, a sender is able to grow the congestion window (cwnd) fast without incurring risk of buffer overflow and multiple losses. Simulation experiments show that Astart can significantly improve the link utilization under various bandwidth, buffer size and round-trip propagation times. The method avoids both under-utilization due to premature Slow start termination, as well as multiple losses due to initially setting ssthresh too high, or increasing cwnd too fast. Experiments also show that Astart achieves good fairness and friendliness toward TCP NewReno. Lab measurements using a FreeBSD Astart implementation are also reported in this paper, providing further evidence of the gains achievable via Astart.

According to (Singh, Guha & Francis, 2005) it is well-known that TCP Reno represents a performance bottleneck as the delay-bandwidth product increases. On high-bandwidth-delay links, the additive increase policy of one packet every RTT necessitates thousands of RTTs to reach full link utilization.

2.6 A Comparative Analysis of TCP Tahoe, Reno, New-Reno, and Vegas

2.6.1 TCP Tahoe

Tahoe refers to the TCP congestion control algorithm which was suggested by Van Jacobson in his paper (Floyd & Henderson, 1999). TCP is based on a principle of 'conservation of packets', i.e. if the connection is running at the available bandwidth capacity then a packet is not injected into the network unless a packet is taken out as well. TCP implements this principle by using the acknowledgements to clock outgoing packets because an acknowledgement means that a packet was taken off the wire by the receiver. It also maintains a congestion window CWO to reflect the network capacity (Floyd & Henderson, 1999). However there are certain issues, which need to be resolved to ensure this equilibrium.

1. Determination of the available bandwidth.
2. Ensuring that equilibrium is maintained.
3. How to react to congestion.

2.6.2 New-Reno

New RENO is a slight modification over TCP-RENO. It is able to detect multiple packet losses and thus is much more efficient that RENO in the event of multiple packet

losses.

Like Reno, New-Reno also enters into fast-retransmit when it receives multiple duplicate packets, however it differs from RENO in that it doesn't exit fast-recovery until all the data which was out standing at the time it entered fast recovery is acknowledged. Thus it overcomes the problem faced by Reno of reducing the CWD multiples times. The fast transmit phase is the same as in Reno. The difference in the fast recovery phase which allows for multiple re-transmissions in new-Reno. Whenever new-Reno enters fast recovery it notes the maximums segment which is outstanding. The fast-recovery phase proceeds as in Reno, however when a fresh ACK is received then there are two cases:

1.If it ACK' s all the segments which were outstanding when we enter fast recovery then it exits fast recovery and sets CWD to ssthresh and continues congestion avoidance like Tahoe.

2. If the ACK is a partial ACK then it deduces that the next segment in line was lost and it re-transmits that segment and sets the number of duplicate ACKS received to zero. It exits Fast recovery when all the data in the window is acknowledged.

2.6.3 Vegas

Vegas is a TCP implementation which is a modification of Reno. It builds on the fact that proactive measures to encounter congestion are much more efficient than reactive ones. It tried to get around the problem of coarse grain timeouts by suggesting an algorithm which checks for timeouts at a very efficient schedule. Also it overcomes the problem of requiring enough duplicate acknowledgements to detect a packet loss, and it also suggest a modified slow start algorithm which prevent it from congesting the

network. It does not depend solely on packet loss as a sign of congestion. It detects congestion before the packet losses occur. However it still retains the other mechanism of Reno and Tahoe, and a packet loss can still be detected by the coarse grain timeout of the other mechanisms fail.

2.6.4 TCP RENO

This Reno retains the basic principle of Tahoe, such as slow starts and the coarse grain re-transmit timer. However it adds some intelligence over it so that lost packets are detected earlier and the pipeline is not emptied every time a packet is lost. Reno requires that we receive immediate acknowledgement whenever a segment is received. The logic behind this is that whenever we receive a duplicate acknowledgment, then his duplicate acknowledgment could have been received if the next segment in sequence expected, has been delayed in the network and the segments reached there out of order or else that the packet is lost. If we receive a number of duplicate acknowledgements then that means that sufficient time has passed and even if the segment had taken a longer path, it should have gotten to the receiver by now. There is a very high probability that it was lost. So

Reno suggest an algorithm called 'Fast Re-transmition. Whenever we receive 3 duplicate ACK's we take it as a sign that the segment was lost, so we re-transmit the segment without waiting for timeout. Thus we manage to re-transmit the segment with the pipe almost full. Another modification that RENO makes is in that after a packet loss, it does not reduce the congestion window to I. Since this empties the pipe. It enters into a algorithm which we call' Fast-Re- Transmit'. The basic algorithm is presented as under:

1. Each time we receive 3 duplicate ACK's we take that to mean that the segment was lost and we re-transmit the segment immediately and enter 'Fast Recovery'

2. Set SSthresh to half the current window size and also set CWO to the same value.
3. For each duplicate ACK receive increase CWO by one. If the increase CWO is greater than the amount of data in the pipe then transmit a new segment else wait.

If there are 'w' segments in the window and one is lost, then we will receive (w-I) duplicate ACK's. Since CWO is reduced to W/2, therefore half a window of data is acknowledged before we can send a new segment. Once we retransmit a segment, we would have to wait for at least one RTT before we would receive a fresh acknowledgement. Whenever we receive a fresh ACK we reduce the CWND to SSthresh. If we had previously received (w-I) duplicate ACK's then at this point we should have exactly w/2 segments in the pipe which is equal to what we set the CWND to be at the end of fast recovery. Thus we don't empty the pipe, we just reduce the flow. We continue with congestion avoidance phase of Tahoe after that.

2.7 TCP Tahoe /Reno

The two most common TCP distributions, TCP Tahoe and TCP Reno, have mechanisms to compensate for the efficiency drop due to the congestion related packet loss. These mechanisms are proven to be successful in the wired networks. In fact, both varieties of TCP are designed with only wired networks in mind. The importance of this assumption is that the only major type of packet loss the wired networks experience is caused by network congestion (Todorovic, 2005).

2.8 TCP vs. UDP Performance Evaluation for CBR Traffic on Wireless Multihop Networks

We present a comprehensive set of measurements of a 2.4 GHz DSSS (Direct-Sequence Spread Spectrum) wireless LAN and analyze its behavior. We examine issues such as host and interface heterogeneity, bidirectional (TCP) traffic and error modeling that have not been previously analyzed. We uncover multiple problems with TCP and UDP performance in this system. We investigate the causes of these problems (radio hardware, device drivers, network protocols) and discuss the effectiveness of proposed improvements (Xylomenos and Polyzos, 1999)

The transport layer behavior regarding TCP and UDP protocols was evaluated. Several simulation scenarios in ns2, over different network topologies and data flow patterns were carried out. QoS characteristics are evaluated for low and heavy traffic in order to characterize throughput, delay and power consumption behavior of the above protocols Simulation results show that TCP suffers on multihop wireless routes, managing to deliver minimum amount of packets on destination. Therefore, fewer data packets can be sent over the multi hop wireless routes compared to UDP protocol. Furthermore, delay is short, because the transmit window of TCP was minimal. On the other hand, UDP achieved better results in throughput, although its mean delay was higher compared to TCP. The reason UDP is faster than TCP is because there is no form of flow control or error correction which also explains the fact that delay over UDP is higher compared to TCP. Finally, as power is concerned, TCP is a more power consuming protocol than UDP, due to the complexity of TCP's structure, i.e. (acknowledgment packets must be send)as well as packet retransmissions are forced (S. Giannoulis, et al,).

According to (Rohner, et al 1998) through a set of simple experiments in a real world setting the effect of constant bit rate (CBR) UDP traffic on adaptive TCP and vice versa, is investigated.

2.9 Behavior of TCP in variable-bandwidth environments

Available bandwidth is the most useful measurement to network adaptive applications and transports. Unfortunately, it is also very difficult to measure in a network that cannot be well approximated by a weighted fair queuing model, such as the Internet (Kazantzidis, 2001).

(Lee ,et al, 2001) they found the following results First, increasing the bandwidth-delay product leads to performance degradation regardless of TCP versions and the bottleneck buffer size. Second, NewReno outperforms Reno and SACK when no packet losses occur during the slow-start phase. Finally, increasing the bottleneck buffer size can lead to improve the link utilization, especially as the bandwidth-delay product gets larger.

According to (Fang, et al, 2005) they define the available bandwidth over one link as the link bandwidth minus the unutilized bandwidth. The estimation scheme is passive measurement, not using any probing messages that may interfere with the networks. The method need not wait too long for data convergence.

Most of the studies on TCP have assumed than the bottleneck bandwidth remains constant over time. In wired networks, the available bandwidth for best-effort trace is usually constant but major changes are foreseen as many telco operators and Internet providers (ISP) are beginning to deploy Quality of Service (QoS) features with

reservation like or priority-like mechanisms in their networks. These new technologies, along with the strong desire to provide bandwidth-on-demand features, may turn wired networks into highly variable-bandwidth environments (VBE) for the best-effort traffic,

In this paper we present a preliminary study of TCP in VBE and show simulation results to better understand the behavior and the performances of TCP in such environments. Both sine-based and step-based bandwidth variations models are used. Both the study and the simulation results show that YBE are challenging networking environments for TCP where packet losses are very frequently encountered. In this case it is tempting to let the TCP congestion control performs the regulation of bandwidth which would limit the growth of cwnd, but there is at the same time the need to grab very quickly the bandwidth when it increases. There are new propositions that increases the TCP increase slope in congestion avoidance phase (Jin, Wei & Low, 2004) for very high bandwidth delay product networks, but then a bandwidth drop is very costly and some preliminary simulations of such approaches suggest that the inefficiency problem is still not completely solved. Regarding the buffer size, the results suggest that the larger the buffer, the lower the inefficiency. However, as we did not look at the end-to-end performance issues such as the transmission completion time, it is still early to say that large buffers are better because large buffers also means more timeouts. We are currently performing more simulations to investigate this issue and also fairness issues.

2.10 Reno TCP

According to (Eddy, 2004) Reno TCP's features include slow-start, congestion avoidance, and fast retransmit which were carried over from the previous Net/1 Tahoe release, and additionally the fast recovery algorithm. These features are all congestion

control related. Originally TCP used the window advertised by the receiver as the amount of data it would send unacknowledged. This was problematic in that the receiver advertises how much buffer space it has made available in memory. Since bytes of end-host memory might be plentiful while router memory may not be, the advertised window can be greater than the buffering capacity of the network.

UDP is able to send almost all the required number of messages within due time whereas throughput for SCTP varies greatly, being on the lower side with lesser buffer sizes and increases with increasing buffer sizes.

Again it is visible the UDP performs better and the reason in quiet obvious for this that UOP has no transport overheads, no flow control, no congestion control, no slow starts. It works in a flat out manner, keeping the thing very simple and straight forward, although not friendly with internet traffic of today with TCP's share of more than 80% (Gill, 2008).

The performance of UOP applications over wireless links has not been extensively studied in the past, mainly due to their diversity. UDP applications were also perceived as LAN oriented, a situation challenged by UDP-based multimedia streaming on the Internet. Considerable work has been devoted to TCP however. Most TCP enhancement schemes try to avoid triggering congestion recovery due to wireless errors. General purpose TCP enhancements such as Selective Acknowledgments improve TCP performance by reducing the number of redundant TCP retransmissions, without reducing their delay though (Polyzos, 2003).

CHAPTER THREE

RESEARCH METHODOLOGY

3.1 Introduction

This chapter discusses the methodology that has been used in this project to achieve the project objectives. It is more than just a collection of methods to handle the research; it is a semantic way proposed by the researcher to solve the research problems. The researcher depend on a simulation model that used by (Yaacob, 2003) to proposed a simulation test-bed environment (Figure 3.1) which contains six phases. It is involves the testing of specific values of the decision or uncontrollable variables in the model and observing the impact on the output values. It also involves setting up a model of a real system and conducting repetitive experiments on it.

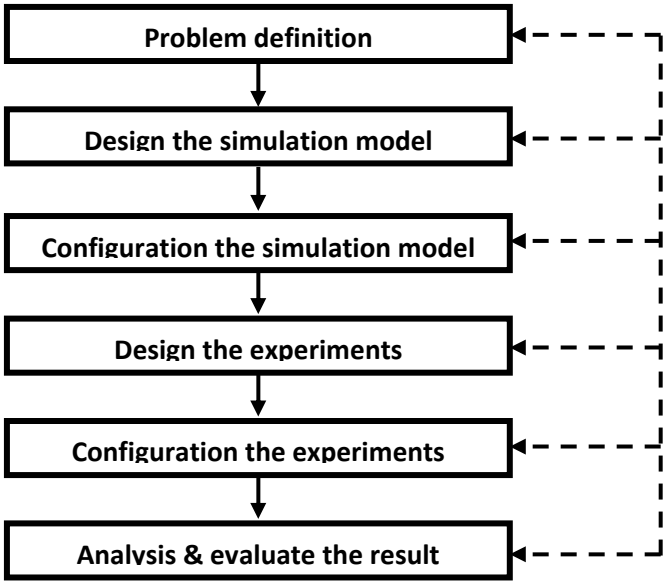


Figure 3.1: Simulation Test-Bed Model (Yaacob, 2003)

3.2 Description of experiments

The main goal of this project is investigate the effects of TCP and UDP Performance in term of bandwidth usage. So far, the traditional methods for examining the performance of TCP and UDP have been measurement, implementation and simulation. However, measurement and experimentation (Floyd and Henderson, 1999) have limitations in that they can only be used to explore the existing Internet. Even.

According to (Bajaj,et al, 1999) Simulation is a vital tool to quickly and inexpensively explore the behavior of new protocols across the range of topologies, and interactions that might occur in the Internet.

Furthermore, simulation (Floyd and Henderson, 1999) plays vital role in attempting to characterize both the behavior of the current Internet and the possible effects of proposed changes to its operation, and also in helping researchers to develop intuition. In particular, the complexities of the Internet topologies and traffic and the central role of adaptive congestion control make simulation the most promising tool for addressing many of the questions about Internet traffic dynamics (Floyd and Henderson, 1999). Not to forget that many successful TCP and UDP performance evaluations have been conducted via simulation (Fall & Floyd, 1996). The following is a list of the advantages of using a simulator to evaluate TCP's performance:

- Simulators are not equipment intensive, as only a single basic workstation is needed to run the simulations and analyze the data.
- Simulators allow a researcher to easily examine a wide range of scenarios in a relatively short amount of time.
- Simulation also provides a means of testing TCP performance across "rare"

networks that a researcher does not have good access to use.

- Complex topologies can be easily created via simulation, whereas such topologies would not be easy to replicate in a test bed environment.
- Simulators give the researcher access to data about all the traffic transmitted in the simulation.

Therefore, the performance investigation of TCP and UDP in this project involves using simulation Scenarios.

3.3 Simulation steps

3.3.1 Problem Definition

Problem definition is an important phase in every study. In this study, the researcher tried to study and collect the necessary information that related to the performance of TCP and UDP in term of bandwidth usage when their applications share the same link. This phase represent the backbone for this study. Information have been gathered and collected from books, journals, proceedings, white papers, reports.

This phase aim to specify the research problem(s), scope, domain, and the limitation after reviewing the collected information and the previous work done by other researchers to start from where they stopped.

3.3.2 Design the Simulation Model

To test the performance effect, of the TCP and UDP in term of bandwidth usage, create the simulation will be needed to run through network simulation. In order to create the topology and run the protocols, we use Network Simulator 2 (NS2) to create and run the coding. NS2 only work in UBUNTU, in our simulation model we will create (8) nodes and send two applications (FTP and CBR) to represent our protocols TCP and UDP. (refer to Figure 3.2)

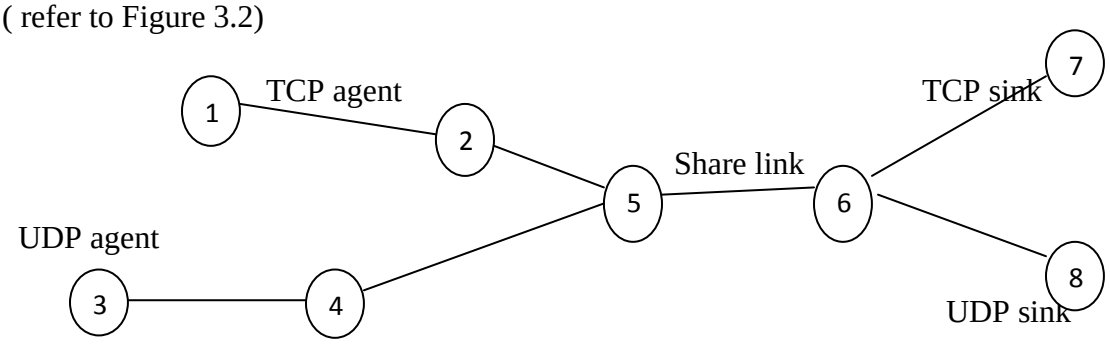


Figure 3.2: Simulation Model

3.3.3 Configuration the Simulation Model

After the simulation model has been designed, the configuration starting, to use NS2 for performance evaluation of TCP and UDP we will create (8) nodes and send two applications (FTP and CBR) to represent our protocols (TCP and UDP), for each application we will create agent and sink (e.g. source and destination for each protocol).

3.3.4 Design the Experiments

To run the codes, we use command which use a few code in running the coding. Then we will get a topology, which is the flow of the packet within the source and destination, base on the coding. A graph also appears after the command.

3.3.5 Conduct the Experiments

This phase of the methodology applying the experiments designed in the previous phase. By depending on our codes, for each experiment need to change the type of TCP (Tahoe, Reno, Newreno and Vegas) and the rate of UDP. Also for each scenario we will get the graph that will use in performance evaluation of UDP and TCP in term of bandwidth usage.

3.3.6 Analysis & Evaluation the Results

The amount of bandwidth usage by TCP and UDP will be used as an indicator of performance of TCP and UDP, this amount we will find it from the graph that will appear after each code running (with each scenario). Beside the graph we will get the trace file for each scenario also which contain the data of bandwidth usage.

CHAPTER 4

SIMULATION RESULTS

4.1 TCP Tahoe Simulation results:

We have been run two scenarios with TCP Tahoe once with 200Kb rate of UDP and other with 4Mb rate of UDP.

4.1.1 TCP Tahoe with 200Kb rate of CBR

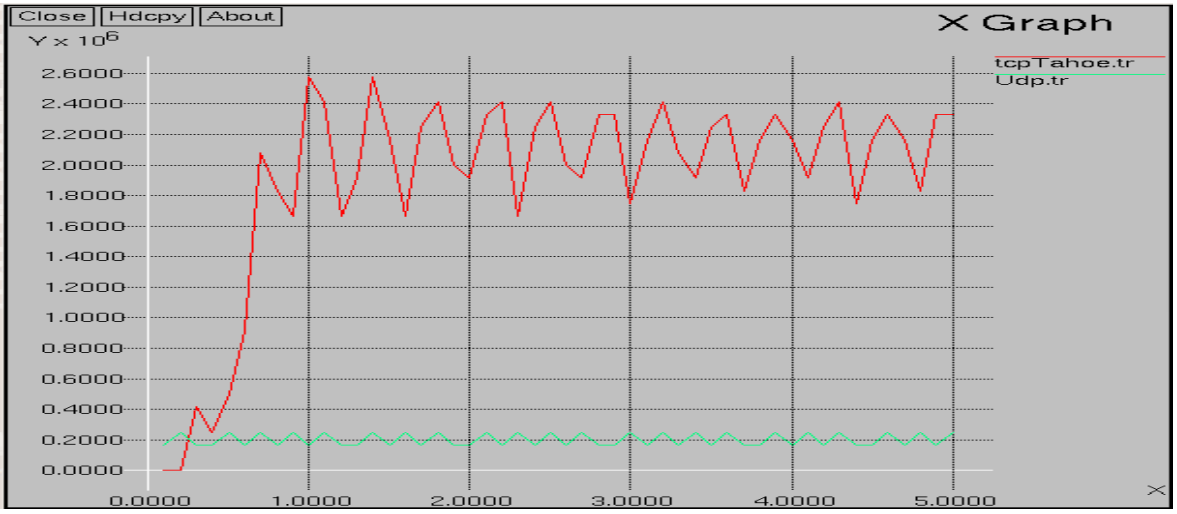


Figure 4.1: The bandwidth usage of TCP Tahoe with 200Kb of UDP

As shown in figure 4.1 TCP Tahoe is consumed bandwidth more than UDP because the CBR is small (200Kb) and the maximum value of usage bandwidth was (2.6*10^6).

4.1.2 TCP Tahoe with 4Mb rate of CBR

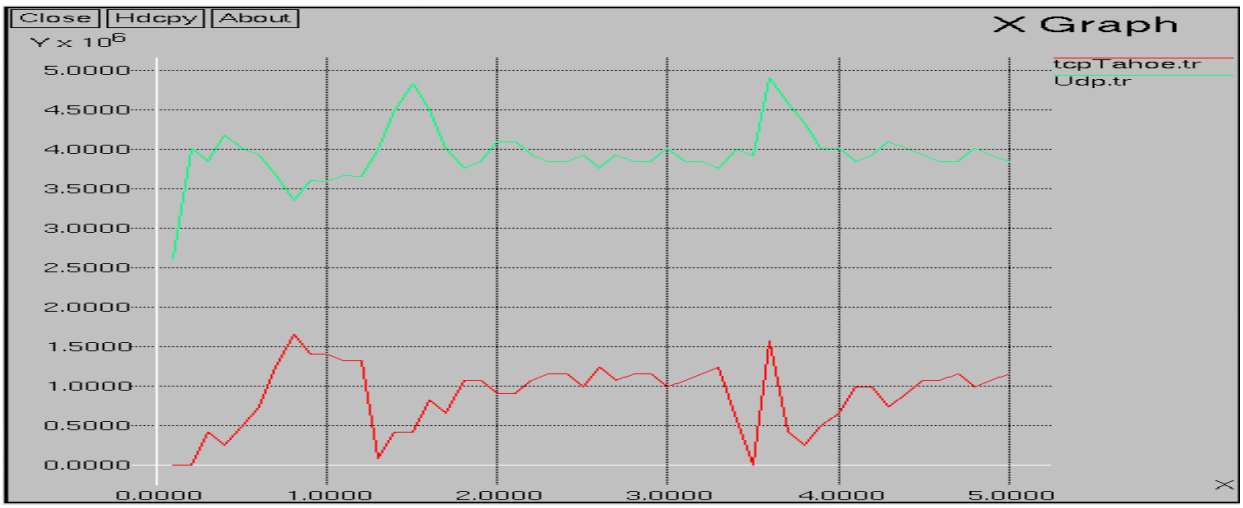


Figure 4.2: The bandwidth usage of TCP Tahoe with 4Mb of UDP

As we shown in figure 4.2 the bandwidth usage by TCP Tahoe is less than bandwidth usage by UDP when the CBR is big (4Mb), and we note the UDP is started with big value of bandwidth usage (2.5*10^6) while the TCP is started with zero value.

4.2 TCP Reno Simulation results:

We have been run two scenarios also with TCP Reno once with 200 Kb rates of UDP and other with 4Mb rate of UDP.

4.2.1 TCP Reno with 200Kb rate of CBR

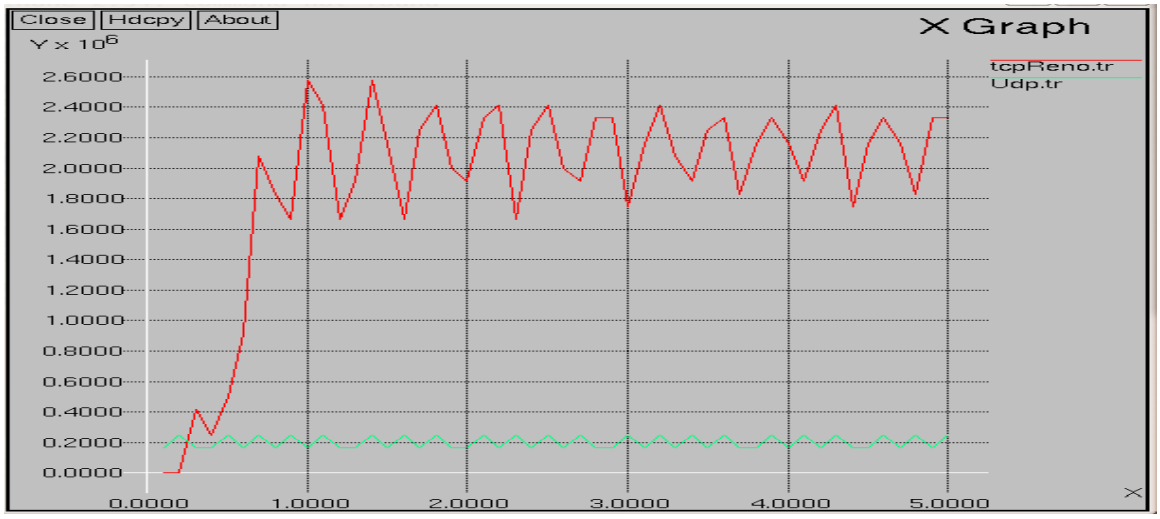


Figure 4.3: The bandwidth usage of TCP Reno with 200Kb of UDP

As shown in figure 4.3 TCP Reno is consumed bandwidth more than UDP because the CBR is small (200Kb) and the maximum value of usage bandwidth was (2.6*10^6).

4.2.2 TCP Reno with 4Mb rate of CBR

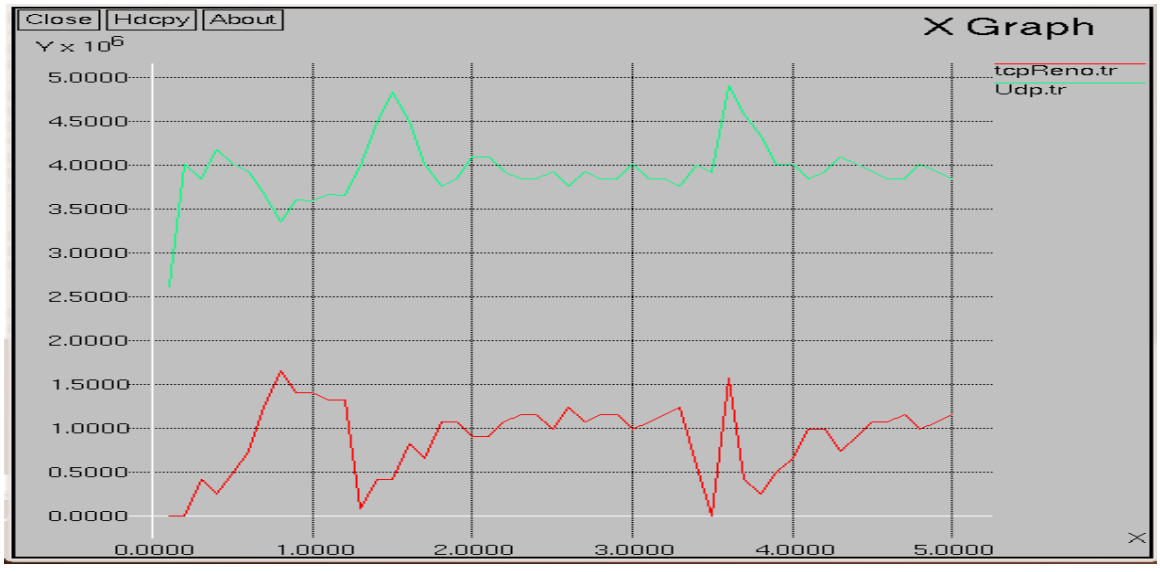


Figure 4.4: The bandwidth usage of TCP Reno with 4Mb of UDP

As we shown in figure 4.4 the bandwidth usage by TCP Reno is less than bandwidth usage by UDP when the CBR is big (4Mb), and we note the UDP is started with big value of bandwidth usage (2.5*10^6) while the TCP is started with zero value.

4.3 TCP Newreno Simulation results:

We have been run two scenarios also with TCP Newreno once with 200Kb rate of UDP and other with 4Mb rate of UDP.

4.3.1 TCP Newreno with 200Kb rate of CBR

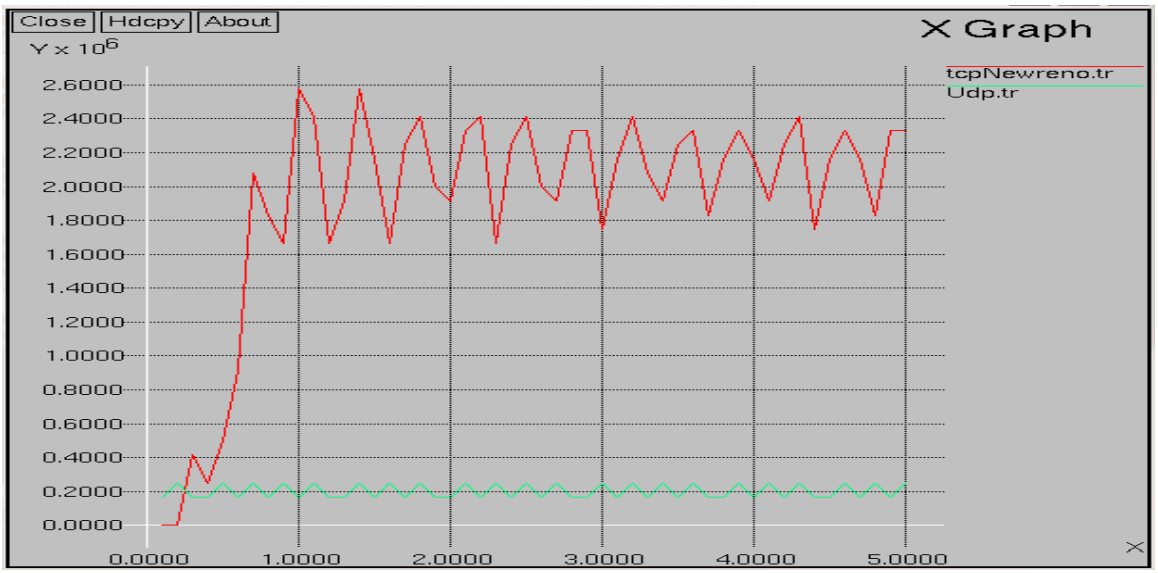


Figure 4.5: The bandwidth usage of TCP Newreno with 200Kb of UDP

As shown in figure 4.5 TCP Newreno is consumed bandwidth more than UDP because the CBR is small (200Kb) and the maximum value of usage bandwidth was (2.6*10^6).

. And the bandwidth usage by UDP still small compared with bandwidth usage by TCP Tahoe, Reno and Newreno.

4.3.2 TCP Newreno with 4Mb rate of CBR

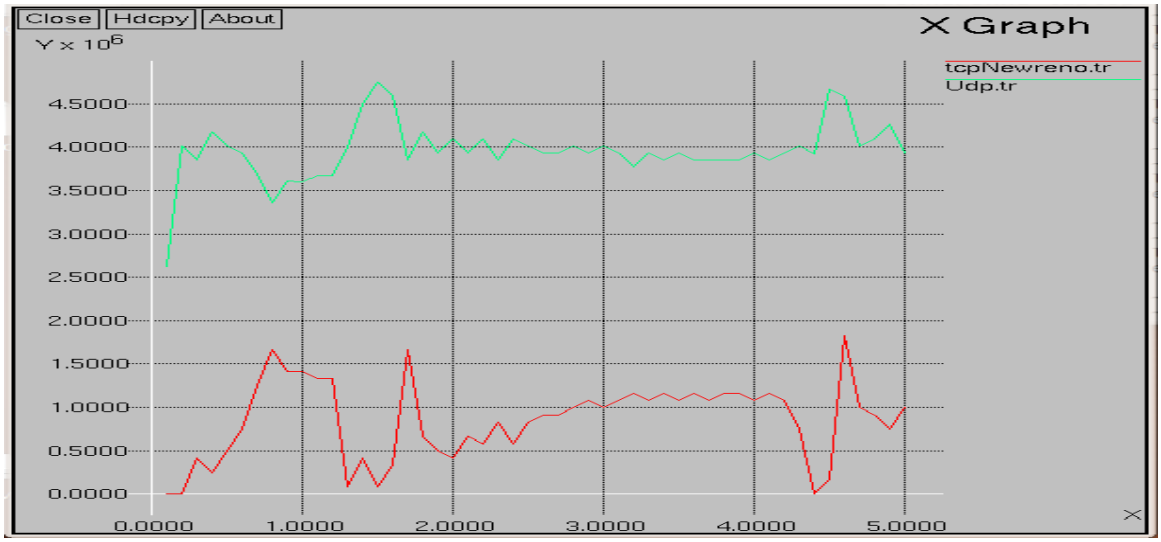


Figure 4.6: The bandwidth usage of TCP Newreno with 4Mb of UDP

As we shown in figure 4.6 the bandwidth usage by TCP Newreno is less than bandwidth usage by UDP when the CBR is big (4Mb), and we note the UDP is started with big value of bandwidth usage (2.5*10^6) while the TCP is started with zero value. Also we note the maximum value of bandwidth usage by TCP Newreno is more than maximum value in first and second scenario.

4.4 TCP Vegas Simulation results:

We have been run two scenario also with TCP Reno once with 200Kb rate of UDP and other with 4Mb rate of UDP.

4.4.1 TCP Vegas with 200Kb rate of CBR

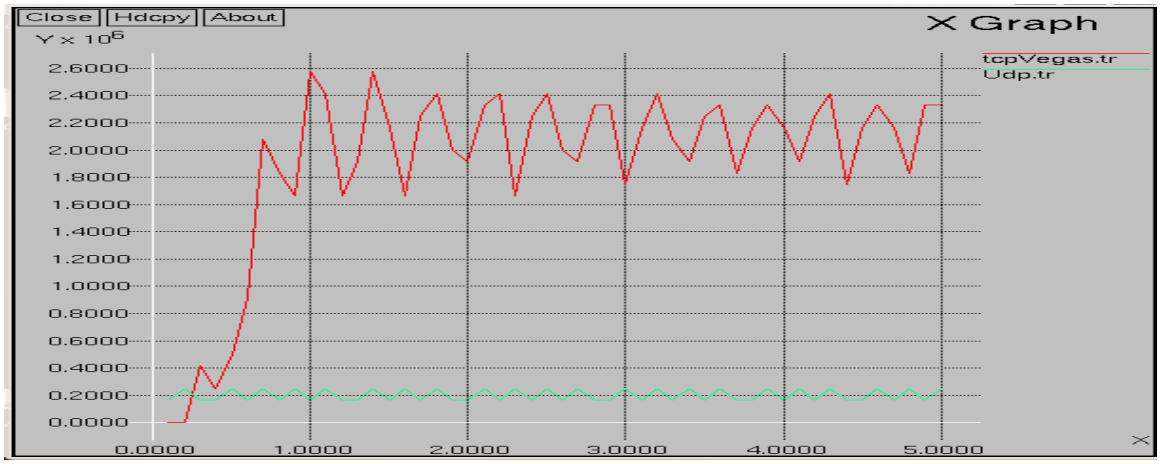


Figure 4.7: The bandwidth usage of TCP Vegas with 200Kb of UDP

As shown in figure 4.7 TCP Newreno is consumed bandwidth more than UDP because the CBR is small (200Kb) and the maximum value of usage bandwidth was (2.6*10^6). And the bandwidth usage by UDP still small compared with bandwidth usage by TCP Tahoe, Reno, Newreno and Vegas.

4.4.2 TCP Vegas with 4Mb rate of CBR

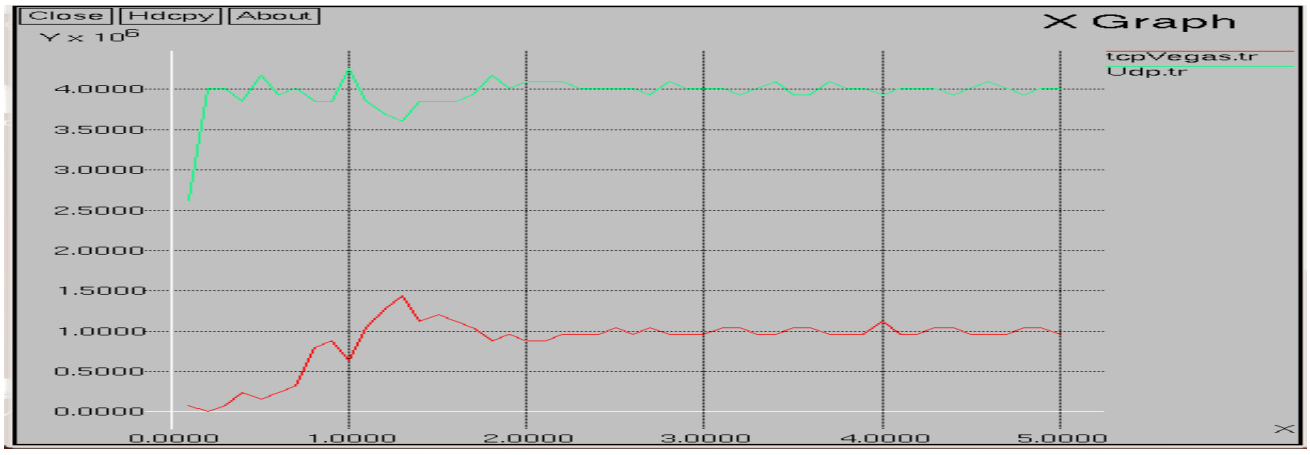


Figure 4.8: The bandwidth usage of TCP Vegas with 4Mb of UDP

As we shown in figure 4.8 the bandwidth usage by 4Mb rate of UDP is more than bandwidth usage by TCP Vegas, but in this scenario the maximum value of bandwidth usage by TCP Vegas which is (4.2*10^6) is less among previous scenarios.

CHAPTER 5

DISCUSSION AND CONCLUSION

5.1 Introduction

This chapter provides an overall review of the results, discussions, implications, and limitations. It will begin with a analyzing and discussion on the research findings, followed by a discussion on the implications of the study. Finally, The study limitations based on the subject of this study.

5.2 Discussions of the finding

The purpose of this research was to investigate the performance of TCP and UDP in term of bandwidth usage. In addition, the study also aimed to help researchers who are interested in improving TCP or UDP performance by giving them an accurate insight about TCP and UDP performance in term of bandwidth usage.

The research focused on study TCP and UDP performance in terms of bandwidth usage, using simulation tool NS2. Also two types of traffic (FTP and CBR) were used during the simulation course, and two scenarios were run for each CBR rate (200Kb and 4Mb) on the same network topology, and for each one to this scenarios we used one of the four TCP's types (Tahoe, Reno, Newreno, and Vegas) which we used in our experiments.

Chapter5: Discussion And Conclusion

Based on our project, we can say that the average bandwidth available to different versions of TCP are not the same. But for this network, we prefer to adopt TCP Vegas because the average bandwidth is higher than other TCP versions.

Regarding the bandwidth share of TCP and UDP, we agree that UDP traffic is suppressing the bandwidth share of TCP traffic. To prevent that, the UDP rate should be decrease if the UDP rate is higher than TCP rate. We can see the differences between 200KB rate of UDP with 4 MB rate of UDP in the graph above.

Based on the graph, there is no difference between all versions of TCP except for TCP Vegas which the average bandwidth of the TCP Vegas is the highest among all the TCP versions.

From our observation, we know that TCP protocol will establish the connection and has acknowledgement to guarantee that the packet send can arrive in orders, and there is no duplicates between the packets. Furthermore, UDP flows do not have any congestion avoidance mechanisms; they do not slow down when their packets are dropped at the routers. In the current internet, as TCP implements congestion avoidance algorithm, it is at a disadvantage (Sudha, et al, 2008). Unlike TCP, UDP protocol does not provide flow and congestion control, its only offer best effort policy. For the speed, TCP can tolerate some packet loss but require a minimum send rate. On the other hand, UDP can send data faster than TCP because there is no connection establishment, no connection state and unregulated send rate. Based on the reasons, UDP is suitable for real-time application such as video conferencing. The real-time application can tolerate

with data loss but at the same time it needs the data to be sent as fast as possible. In this case, UDP fulfill the application's requirement.

5.3 Implications of the study

This study has attempted to contribute to the body of work on simulation, modeling, analyzing and improving TCP and UDP protocol.

This research also help anybody wants to model a student's lab or internet café that will be when our study helps the manager to estimate the bandwidth he need to build success project, because the user satisfaction is important in term of economic.

This study contributes in the development of a network topology. This network topology may form a useful simulation topology for utilizing by other researchers in order to assess, study, and test the TCP and UDP performance.

This research also provides guide to TCP and UDP researchers whom thinking about improving TCP or UDP performance. Since, the findings reveal that TCP behaves better than UDP when CBR rate was big in the same link, and this issue needs to enhance UDP performance with big rate in term of bandwidth usage.

5.5 Conclusion

Accurate network bandwidth effect is important to a variety of network applications. In spite of the recent sudden increase in accessibility of broadband Internet access, the best part of home users have fairly small-bandwidth links in comparison with the sites hosting beloved content. It is quite common for users to run multiple

networking applications on a single connection, and the growing popularity of recent peer-to-peer file-sharing services. Consequently there are many circumstances in which the last hop link becomes a bottleneck resulting in congestion among a user's applications.

Based on our project, we can conclude that different type of TCP can give different result on those simulations. In this case, TCP Vegas seems to be the best compared to other types of TCP. Besides, packets transfer also influenced by the rate of the bandwidth.

We also notice that the rate of UDP has the possibility to affect TCP send rates, depends on the UDP's rate. More higher the UDP's rate, more suppressed the TCP traffic will be at the shared bandwidth.

REFERENCES

A study of the behavior of TCP in variable-bandwidth environments. from http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.59.8428 2/8

A Comparative Analysis of TCP Tahoe, Reno, New-Reno, SACK and Vegas. (1998). from http://inst.eecs.berkeley.edu/~ee122/fa05/projects/Project2/SACKRENEVEGAS.pdf

A.K.Aggarwal, A. (2003). ACC – ABCD Compliant NS-2.

Allman, M., and Falk, A. (1999). On the Effective Evaluation of TCP. *ACM Computer Communication Review*.

Allman, M., Paxson, V., and Stevens, W. (1999). TCP Congestion Control. *RFC 2581, IETF*

Bajaj, s., Breslau, L., Estrin, D., Fall, K., Floyd, S., Haldar, P., et al. (1999). Improving Simulation for Network Research. *IEEE*.

Barbeau, J. G. M. (2000). "Comparison of Bandwidth Usage: Service Location Protocol and Jini."

Chohan, N. (2006). An Analysis of TCP through Simulation.

Eddy, W. M. (2004). *Improving Transmission Control Protocol Performance With Path Error Rate Information.* College of Engineering and Technology of Ohio

University.

Fall, K.andFloyd,S.1996. Simulation-based Comparison of Taho, Reno and SSACK TCP.

Fang, Q., Jia, W., and Wu, J. (2005). Available Bandwidth Detection with Improved. Transport Control Algorithm for Heterogeneous Networks. *ACM*, 656 - 659.

Floyd, S., and Fall, K. (1999). Promoting the use of end-to-end congestion control in the Internet. *IEEE/ACM Transactions on Networking, 7*(4).

Floyd, S., and Fall, K.2001. Why we don't know how to simulate the Internet.

Floyd, S., and Fall, K.1997. Difficulties in Simulating the Internet.

Forouzan, B. (2000). *TCP/IP Protocol Suite*. Boston: MCGrawHill.

Giannoulis, S., Antonopoulos, C., Topalis, E., Athanasopoulos, A., Prayati, A., and Koubias, S. TCP vs. UDP Performance Evaluation for CBR Traffic On Wireless Multihop Networks. from http://www.wcl.ee.upatras.gr/csndsp/CD/contents/Sessions/Presentations/A7%20-%20Wireless%20LAN/A7.4.pdf

Gill, M. and. Zafar. M. S. (2008). Evaluation of UDP and SCTP for SIP-T and TCP, UDP and SCTP with constant traffic.

Govea, J., and Barbeau, M. (2000). Comparison of Bandwidth Usage: Service Location Protocol and Jini. from www.scs.carleton.ca/~barbeau/Publications/2000/TR_00_06.pdf

Hassan, M., and Jain, R. (2004). *High Performance TCP/IP Networking Concepts,*

Issues, and solutions. London: Prentice Hall.

Hossain, T. I. E. (2009). Introduction to Network Simulator NS2.

Huston, G., and Telstra. (2009). TCP Performance. *The Internet Protocol Journal, 3*(2).

Issariyakul, T., and Hossain, E. (2009). *Introduction to Network Simulator NS2*. New York, USA: Springer Science+Business Media.

Jacobson, V. (1988). *Congestion avoidance and control.* Paper presented at the ACM SIGCOMM Special Interest Group on Data Communications.

Jeonghoon Mo, R. J. L., Venkat Anantharam, and Jean Walrand. (2004). Analysis and Comparison of TCP Reno and Vegas [Electronic Version].

Jin, C., X.Wei, D., and Low, S. H. (2004). FAST TCP: Motivation, Architecture, Algorithms, Performance. *IEEE Infocom*

Kazantzidis, M. (2001). How to measure available bandwidth on the Internet. from http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.25.2565

Khorashadi, B., Chen, A., Ghosal, D., Chuah, C.-N., and Zhang, M. (2007). Impact of Transmission Power on the Performance of UDP in Vehicular Ad Hoc Networks. *IEEE*, 3698-3703.

Kurose, J., and Ross, K. (2005). *Networking: A Top-Down Approach Featuring the Internet* (3 ed.). Boston: Addison-Wesley.

Lai, Y. and Yao, C. (2000). The Performance Comparison between TCP Reno and TCP Vegas. *IEEE*, 61 – 66.

References

Lee, H., Lee, S.-h., and Choi, Y. (2001). The Influence of the Large Bandwidth-Delay Product on TCP Reno, NewReno, and SACK. *IEEE*, 327-334.

Mattsson, N.-E. (2004). *A DCCP module for NS-2.* Lulea Tekniska University.

Mehra, P., and Zakhor, A. (2003). Receiver-Driven Bandwidth Sharing for TCP. *IEEE, 7*(4), 740- 752.

Mo, J., La, R. J., Anantharam, V., and Walrand, J. (1999). Analysis and Comparison of TCP Reno and Vegas. *IEEE, 3*, 1556-1563

Peterson, L. L., and Davide, B. S. (2003). *Network Simulation Experiments Manual*. San Francisco, USA: Morgan Kaufmann Publishers.

Polyzos, G. X. a. G. C. (2003). "Wireless link layer enhancements for TCP and UDP applications."

Postel, J. (1981). Transmission Control Protocol. *RFC 793*.

Ren Wang, G. P., Kenshin Yamada, M.Y. Sanadidi, and Mario Gerla. (2004). TCP Startup Performance in Large Bandwidth Delay Networks [Electronic Version].

Rohner, C., Nordstrm, E., Gunningberg, P., & Tschudinn, C. (1998). Interactions between TCP, UDP and routing protocols in wireless multi-hop ad hoc *networks*.ROSS, J. F. K. K. W. (2008). Computer network.

Ross, J. F. K. K. W. (2008). Computer network.

S.Floyd, and T.Henderson. (1999). The NewReno Modification to TCP's Fast Recovery Algorithm. *RFC 2582*.

References

S. Giannoulis, C. A., E. Topalis, A. Athanasopoulos, A. Prayati, S. Koubias.TCP vs. UDP Performance Evaluation for CBR Traffic On Wireless Multihop Networks [Electronic Version] from http://www.google.com.my/search?hl=enandei=aGKoSvPFBIme6gPN3uWYBgandsa =1.

S.Sudha, Maddipati, S., and Ammasaigounden, N. (2008). A new adaptive marker for bandwidth fairness between TCP and UDP traffic in DiffServ. *IEEE*, 1-5.

Singh, H., and Singh, S. (2002). Energy Consumption of TCP Reno, Newreno, and SACK in Multi-Hop Wireless Networks. *ACM*, 206 - 216.

Singh, M., Guha, S., and Francis, P. (2005). Utilizing spare network bandwidth to improve TCP performance. *ACM*.

Stevens, W. (1997). TCP Slow Start, Congestion Avoidance, Fast Retransmit, and Fast Recovery Algorithms. *RFC 2001, IEEE*.

Stevens ,W.(1994).TCP/IP illustrated.

Todorovic, M. (2005). *Comparative Study Of The End-To-End Compliant Tcp Protocols For Wireless Networks.* Texas Tech University, USA.

Wang, R., Pau, G., Yamada, K., Sanadidi, M. Y., and Gerla, M. (2002). TCP Startup Performance in Large Bandwidth Delay Networks. from www.ieee-infocom.org/2004/Papers/16_5.PDF

Xu, S., and Saadawi, T. (2001). Does the ieee 802.11 mac protocol work well in multihop wireless ad hoc networks? . *IEEE Communications Magazine, 39*(4).

References

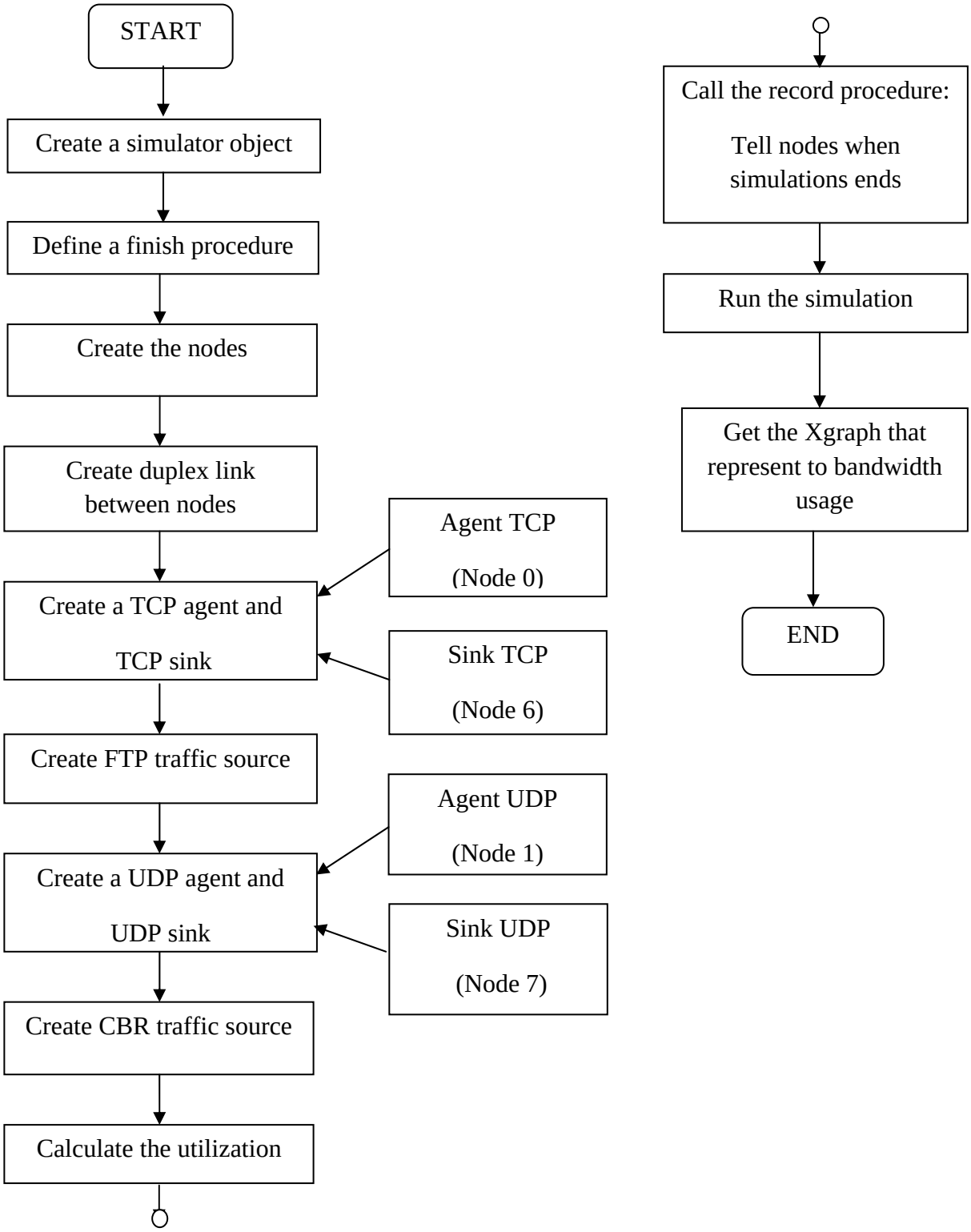
Xylomenos, G., and Polyzos, G. C. (1999). TCP and UDP Performance over a Wireless LAN. *IEEE*, 439–446.

Yaacob, N. A. (2003). Utilizing Snort in the analysis of intrusion Detection System. University Utara Malaysia.

Zafar, M. S., and Gill, M. S. (2008). *Evaluation of UDP and SCTP for SIP-T and TCP, UDP and SCTP with constant traffic.* Blekinge Institute of Technology.

Zakhor, P.M. (2003). Receiver-Driven Bandwidth Sharing for TCP [Electronic Version] from http://www.google.com.my/search?hl=enandq=Receiver-Driven+Bandwidth+Sharing+for+TCP.6/8

Appendix A Flow chart



Appendix B: NS2 SCOURCE CODE

1.TAHOE.

1.1 Script(200Kb rate of UDP)

```
#Create a simulator object
set ns [new Simulator]
$ns use-newtrace
#Define different colors for data flows (for NAM)
$ns color 1 Red
$ns color 2 Cyan
#Open the ns trace file
set nf [open out.nam w]
$ns namtrace-all $nf
set files(0)      [open tcpTahoe.tr w]
set files(1)      [open Udp.tr w]
set val(stop)   200  ;# time of simulation end
set val(nn)     8    ;# number of nodes
# Define a 'finish' procedure
proc stop {} {
   global ns nf files
   $ns flush-trace
# Close the NAM trace file
   close $nf
# Close the output file
   close $files(0)
   close $files(1)
```

Appendix

```
#Execute xgraph tp display the result
exec xgraph tcpTahoe.tr Udp.tr -geometry 800*400 &
#Execute NAM  on trace file
exec nam out.nam &
   exit 0
}
#Create 8 nodes
for {set i 0} {$i < $val(nn)} {incr i} {
   set n($i) [$ns node]
}
#Create a duplex link between the nodes
$ns duplex-link $n(0) $n(2) 5Mb 10ms DropTail
$ns duplex-link $n(2) $n(4) 10Mb 10ms DropTail
$ns duplex-link $n(4) $n(5) 5Mb 5ms DropTail
$ns duplex-link $n(5) $n(6) 5Mb 10ms DropTail
$ns duplex-link $n(1) $n(3) 10Mb 5ms DropTail
$ns duplex-link $n(3) $n(4) 10Mb 10ms DropTail
$ns duplex-link $n(5) $n(7) 5Mb 10ms DropTail
#Set node position in nam
$ns duplex-link-op $n(0) $n(2) orient right-down
$ns duplex-link-op $n(2) $n(4) orient right-down
$ns duplex-link-op $n(4) $n(5) orient right
$ns duplex-link-op $n(5) $n(6) orient right-up
$ns duplex-link-op $n(1) $n(3) orient rightp-u
$ns duplex-link-op $n(3) $n(4) orient right-up
$ns duplex-link-op $n(5) $n(7) orient right-down
#Create a TCP agent and attach it to node n0.The flow between n0 and n6
set src_tcp [new Agent/TCP]
```

Appendix

```
$ns attach-agent $n(0) $src_tcp
$src_tcp set class_ 1
$src_tcp set maxcwnd_ 64  ;# max congestion window size
#Create sink node and attach it to node 6 (n6)
set sink_tcp [new Agent/TCPSink/DelAck]
$ns attach-agent $n(6) $sink_tcp
$ns connect $src_tcp $sink_tcp
#Setup FTP over TCP connection
set ftp [new Application/FTP]
$ftp set packetSize_ 1024
$ftp attach-agent $src_tcp
#$ns at 0.5 "$tcp advanceby 1"
$ns at 0.0 "$ftp start"
# Create UDP agent and attach it to node 1.UDP flow between n1 and n7
set src_udp [new Agent/UDP]
$ns attach-agent $n(1) $src_udp
$src_udp set class_ 2
#Create sink node abd attach to node 7 (n7)
set sink_udp [new Agent/LossMonitor]
$ns attach-agent $n(7) $sink_udp
$ns connect $src_udp $sink_udp
#Setup a CBR over UDP connection
set cbr [new Application/Traffic/CBR]
$cbr set packetSize_ 1024
$cbr set rate_ 0.2Mb
$cbr attach-agent $src_udp
$ns at 0.0 "$cbr start"
# printing throughput of each client and channel utilization
```

Appendix

```
set last_udp 0.0
set last_tcp 0.0
proc plotFiles {} {
 global ns files last_udp last_tcp sink_udp sink_tcp val
    set time 0.1
    set now [$ns now]
    set total 0.0
     set cur_tcp [$sink_tcp set bytes_]
    set temp [expr $cur_tcp-$last_tcp]
    puts $files(0) "$now [expr $temp*8/$time]"
    set total [expr $total+$temp]
  set cur_udp [$sink_udp set bytes_]
    set temp [expr $cur_udp-$last_udp]
    puts $files(1) "$now [expr $temp*8/$time]"
    set total [expr $total+$temp]
   set last_udp $cur_udp
    set last_tcp $cur_tcp
       set nextTime [expr $now+$time]
    if { $nextTime <= $val(stop) } {
        $ns at $nextTime "plotFiles"
    }
}
$ns at 0.1 "plotFiles"
# telling nodes when the simulation ends
for {set i 0} {$i < $val(nn)} {incr i} {
    $ns at $val(stop) "$n($i) reset";
}
$ns at $val(stop) "$ftp stop"
```

```
$ns at $val(stop) "$cbr stop"
$ns at $val(stop) "stop"
$ns at 200.1 "puts \"end simulation\"; $ns halt"
$ns at 5.0 "stop"
#Run the simulation
$ns run
```

1.2 Script(4Mb rate of UDP)

```
#Create a simulator object
set ns [new Simulator]
$ns use-newtrace
 #Define different colors for data flows (for NAM)
$ns color 1 red
$ns color 2 Cyan
#Open the ns trace file
set nf [open out.nam w]
$ns namtrace-all $nf
set files(0)      [open tcpTahoe.tr w]
set files(1)      [open Udp.tr w]
set val(stop)   200  ;# time of simulation end
set val(nn)     8    ;# number of nodes
# Define a 'finish' procedure
proc stop {} {
   global ns nf files
   $ns flush-trace
```

Appendix

```
    # Close the NAM trace file
    close $nf
    # Close the output file
    close $files(0)
    close $files(1)
#Execute xgraph tp display the result
exec xgraph tcpTahoe.tr Udp.tr -geometry 800*400 &
 #Execute NAM  on trace file
exec nam out.nam &
    exit 0
}
#Create 8 nodes
for {set i 0} {$i < $val(nn)} {incr i} {
   set n($i) [$ns node]
}
#Create a duplex link between the nodes
$ns duplex-link $n(0) $n(2) 5Mb 10ms DropTail
$ns duplex-link $n(2) $n(4) 10Mb 10ms DropTail
$ns duplex-link $n(4) $n(5) 5Mb 5ms DropTail
$ns duplex-link $n(5) $n(6) 5Mb 10ms DropTail
$ns duplex-link $n(1) $n(3) 10Mb 5ms DropTail
$ns duplex-link $n(3) $n(4) 10Mb 10ms DropTail
$ns duplex-link $n(5) $n(7) 5Mb 10ms DropTail
 #Set node position in nam
$ns duplex-link-op $n(0) $n(2) orient right-down
$ns duplex-link-op $n(2) $n(4) orient right-down
```

Appendix

```
$ns duplex-link-op $n(4) $n(5) orient right
$ns duplex-link-op $n(5) $n(6) orient right-up
$ns duplex-link-op $n(1) $n(3) orient rightp-u
$ns duplex-link-op $n(3) $n(4) orient right-up
$ns duplex-link-op $n(5) $n(7) orient right-down
#Create a TCP agent and attach it to node n0.The flow between n0 and n6
set src_tcp [new Agent/TCP/Tahoe]
$ns attach-agent $n(0) $src_tcp
$src_tcp set class_ 1
$src_tcp set maxcwnd_ 64  ;# max congestion window size
 #Create sink node and attach it to node 6 (n6)
set sink_tcp [new Agent/TCPSink/DelAck]
$ns attach-agent $n(6) $sink_tcp
$ns connect $src_tcp $sink_tcp
 #Setup FTP over TCP connection
set ftp [new Application/FTP]
$ftp set packetSize_ 1024
$ftp attach-agent $src_tcp
 #$ns at 0.5 "$tcp advanceby 1"
$ns at 0.0 "$ftp start"
 # Create UDP agent and attach it to node 1.UDP flow between n1 and n7
set src_udp [new Agent/UDP]
$ns attach-agent $n(1) $src_udp
$src_udp set class_ 2
#Create sink node abd attach to node 7 (n7)
set sink_udp [new Agent/LossMonitor]
```

Appendix

```
$ns attach-agent $n(7) $sink_udp
$ns connect $src_udp $sink_udp
#Setup a CBR over UDP connection
set cbr [new Application/Traffic/CBR]
$cbr set packetSize_ 1024
$cbr set rate_ 4Mb
$cbr attach-agent $src_udp
$ns at 0.0 "$cbr start"
  # printing throughput of each client and channel utilization
set last_udp 0.0
   set last_tcp 0.0
     proc plotFiles {} {
   global ns files last_udp last_tcp sink_udp sink_tcp val
   set time 0.1
   set now [$ns now]
   set total 0.0
    set cur_tcp [$sink_tcp set bytes_]
   set temp [expr $cur_tcp-$last_tcp]
   puts $files(0) "$now [expr $temp*8/$time]"
   set total [expr $total+$temp]
    set cur_udp [$sink_udp set bytes_]
   set temp [expr $cur_udp-$last_udp]
   puts $files(1) "$now [expr $temp*8/$time]"
   set total [expr $total+$temp]
    set last_udp $cur_udp
   set last_tcp $cur_tcp
```

```
        set nextTime [expr $now+$time]
        if { $nextTime <= $val(stop) } {
            $ns at $nextTime "plotFiles"
        }
    }
    $ns at 0.1 "plotFiles"
# telling nodes when the simulation ends
    for {set i 0} {$i < $val(nn)} {incr i} {
        $ns at $val(stop) "$n($i) reset";
    }
    $ns at $val(stop) "$ftp stop"
    $ns at $val(stop) "$cbr stop"
    $ns at $val(stop) "stop"
    $ns at 200.1 "puts \"end simulation\"; $ns halt"
    $ns at 5.0 "stop"
#Run the simulation
    $ns run
```

2.RENO.

1.1 Script(200Kb rate of UDP)

```
#Create a simulator object
set ns [new Simulator]
$ns use-newtrace
#Define different colors for data flows (for NAM)
$ns color 1 Red
```

Appendix

```
$ns color 2 Cyan
#Open the ns trace file
set nf [open out.nam w]
$ns namtrace-all $nf
set files(0)      [open tcpReno.tr w]
set files(1)      [open Udp.tr w]
set val(stop)   200  ;# time of simulation end
set val(nn)     8    ;# number of nodes
# Define a 'finish' procedure
proc stop {} {
   global ns nf files
   $ns flush-trace
# Close the NAM trace file
   close $nf
# Close the output file
   close $files(0)
   close $files(1)
#Execute xgraph tp display the result
exec xgraph tcpReno.tr Udp.tr -geometry 800*400 &
#Execute NAM  on trace file
exec nam out.nam &
   exit 0
}
#Create 8 nodes
for {set i 0} {$i < $val(nn)} {incr i} {
   set n($i) [$ns node]
}
#Create a duplex link between the nodes
```

Appendix

```
$ns duplex-link $n(0) $n(2) 5Mb 10ms DropTail
$ns duplex-link $n(2) $n(4) 10Mb 10ms DropTail
$ns duplex-link $n(4) $n(5) 5Mb 5ms DropTail
$ns duplex-link $n(5) $n(6) 5Mb 10ms DropTail
$ns duplex-link $n(1) $n(3) 10Mb 5ms DropTail
$ns duplex-link $n(3) $n(4) 10Mb 10ms DropTail
$ns duplex-link $n(5) $n(7) 5Mb 10ms DropTail
#Set node position in nam
$ns duplex-link-op $n(0) $n(2) orient right-down
$ns duplex-link-op $n(2) $n(4) orient right-down
$ns duplex-link-op $n(4) $n(5) orient right
$ns duplex-link-op $n(5) $n(6) orient right-up
$ns duplex-link-op $n(1) $n(3) orient rightp-u
$ns duplex-link-op $n(3) $n(4) orient right-up
$ns duplex-link-op $n(5) $n(7) orient right-down
#Create a TCP agent and attach it to node n0.The flow between n0 and n6
set src_tcp [new Agent/TCP]
$ns attach-agent $n(0) $src_tcp
$src_tcp set class_ 1
$src_tcp set maxcwnd_ 64  ;# max congestion window size
#Create sink node and attach it to node 6 (n6)
set sink_tcp [new Agent/TCPSink/DelAck]
$ns attach-agent $n(6) $sink_tcp
$ns connect $src_tcp $sink_tcp
#Setup FTP over TCP connection
set ftp [new Application/FTP]
$ftp set packetSize_ 1024
$ftp attach-agent $src_tcp
```

Appendix

```
#$ns at 0.5 "$tcp advanceby 1"
$ns at 0.0 "$ftp start"
# Create UDP agent and attach it to node 1.UDP flow between n1 and n7
set src_udp [new Agent/UDP]
$ns attach-agent $n(1) $src_udp
$src_udp set class_ 2
#Create sink node abd attach to node 7 (n7)
set sink_udp [new Agent/LossMonitor]
$ns attach-agent $n(7) $sink_udp
$ns connect $src_udp $sink_udp
#Setup a CBR over UDP connection
set cbr [new Application/Traffic/CBR]
$cbr set packetSize_ 1024
$cbr set rate_ 0.2Mb
$cbr attach-agent $src_udp
$ns at 0.0 "$cbr start"
# printing throughput of each client and channel utilization
set last_udp 0.0
set last_tcp 0.0
proc plotFiles {} {
   global ns files last_udp last_tcp sink_udp sink_tcp val
   set time 0.1
   set now [$ns now]
   set total 0.0
   set cur_tcp [$sink_tcp set bytes_]
   set temp [expr $cur_tcp-$last_tcp]
   puts $files(0) "$now [expr $temp*8/$time]"
   set total [expr $total+$temp]
```

```
 set cur_udp [$sink_udp set bytes_]
    set temp [expr $cur_udp-$last_udp]
    puts $files(1) "$now [expr $temp*8/$time]"
    set total [expr $total+$temp]
    set last_udp $cur_udp
    set last_tcp $cur_tcp
       set nextTime [expr $now+$time]
    if { $nextTime <= $val(stop) } {
        $ns at $nextTime "plotFiles"
    }
}
$ns at 0.1 "plotFiles"
# telling nodes when the simulation ends
for {set i 0} {$i < $val(nn)} {incr i} {
    $ns at $val(stop) "$n($i) reset";
}
$ns at $val(stop) "$ftp stop"
$ns at $val(stop) "$cbr stop"
$ns at $val(stop) "stop"
$ns at 200.1 "puts \"end simulation\"; $ns halt"
$ns at 5.0 "stop"
#Run the simulation
$ns run
```

2.2 Script(4Mb rate of UDP)

```
#Create a simulator object
```

Appendix

```
set ns [new Simulator]
$ns use-newtrace
 #Define different colors for data flows (for NAM)
$ns color 1 red
$ns color 2 Cyan
#Open the ns trace file
set nf [open out.nam w]
$ns namtrace-all $nf
 set files(0)      [open tcpReno.tr w]
set files(1)      [open Udp.tr w]
set val(stop)   200  ;# time of simulation end
set val(nn)     8    ;# number of nodes
 # Define a 'finish' procedure
proc stop {} {
    global ns nf files
    $ns flush-trace
  # Close the NAM trace file
    close $nf
     # Close the output file
    close $files(0)
    close $files(1)
#Execute xgraph tp display the result
exec xgraph tcpReno.tr Udp.tr -geometry 800*400 &
 #Execute NAM  on trace file
exec nam out.nam &
  exit 0
```

```
}
#Create 8 nodes
for {set i 0} {$i < $val(nn)} {incr i} {
   set n($i) [$ns node]
}
 #Create a duplex link between the nodes
$ns duplex-link $n(0) $n(2) 5Mb 10ms DropTail
$ns duplex-link $n(2) $n(4) 10Mb 10ms DropTail
$ns duplex-link $n(4) $n(5) 5Mb 5ms DropTail
$ns duplex-link $n(5) $n(6) 5Mb 10ms DropTail
$ns duplex-link $n(1) $n(3) 10Mb 5ms DropTail
$ns duplex-link $n(3) $n(4) 10Mb 10ms DropTail
$ns duplex-link $n(5) $n(7) 5Mb 10ms DropTail
 #Set node position in nam
$ns duplex-link-op $n(0) $n(2) orient right-down
$ns duplex-link-op $n(2) $n(4) orient right-down
$ns duplex-link-op $n(4) $n(5) orient right
$ns duplex-link-op $n(5) $n(6) orient right-up
$ns duplex-link-op $n(1) $n(3) orient rightp-u
$ns duplex-link-op $n(3) $n(4) orient right-up
$ns duplex-link-op $n(5) $n(7) orient right-down

#Create a TCP agent and attach it to node n0.The flow between n0 and n6
set src_tcp [new Agent/TCP/Reno]
$ns attach-agent $n(0) $src_tcp
$src_tcp set class_ 1
```

```
$src_tcp set maxcwnd_ 64  ;# max congestion window size
  #Create sink node and attach it to node 6 (n6)
set sink_tcp [new Agent/TCPSink/DelAck]
$ns attach-agent $n(6) $sink_tcp
$ns connect $src_tcp $sink_tcp
  #Setup FTP over TCP connection
set ftp [new Application/FTP]
$ftp set packetSize_ 1024
$ftp attach-agent $src_tcp
#$ns at 0.5 "$tcp advanceby 1"
$ns at 0.0 "$ftp start"
 # Create UDP agent and attach it to node 1.UDP flow between n1 and n7
set src_udp [new Agent/UDP]
$ns attach-agent $n(1) $src_udp
$src_udp set class_ 2
  #Create sink node abd attach to node 7 (n7)
set sink_udp [new Agent/LossMonitor]
$ns attach-agent $n(7) $sink_udp
$ns connect $src_udp $sink_udp
 #Setup a CBR over UDP connection
set cbr [new Application/Traffic/CBR]
$cbr set packetSize_ 1024
$cbr set rate_ 4Mb
$cbr attach-agent $src_udp
$ns at 0.0 "$cbr start"
   # printing throughput of each client and channel utilization
```

Appendix

```
set last_udp 0.0
set last_tcp 0.0
 proc plotFiles {} {
global ns files last_udp last_tcp sink_udp sink_tcp val
set time 0.1
set now [$ns now]
set total 0.0
 set cur_tcp [$sink_tcp set bytes_]
set temp [expr $cur_tcp-$last_tcp]
puts $files(0) "$now [expr $temp*8/$time]"
set total [expr $total+$temp]
  set cur_udp [$sink_udp set bytes_]
set temp [expr $cur_udp-$last_udp]
puts $files(1) "$now [expr $temp*8/$time]"
set total [expr $total+$temp]
set last_udp $cur_udp
set last_tcp $cur_tcp
  set nextTime [expr $now+$time]
if { $nextTime <= $val(stop) } {
   $ns at $nextTime "plotFiles"
}
}
$ns at 0.1 "plotFiles"
# telling nodes when the simulation ends
for {set i 0} {$i < $val(nn)} {incr i} {
$ns at $val(stop) "$n($i) reset";
```

```
}
$ns at $val(stop) "$ftp stop"
$ns at $val(stop) "$cbr stop"
$ns at $val(stop) "stop"
$ns at 200.1 "puts \"end simulation\"; $ns halt"
$ns at 5.0 "stop"
#Run the simulation
$ns run
```

3.NEWRENO.

3.1 Script(200Kb rate of UDP)

```
#Create a simulator object
set ns [new Simulator]
$ns use-newtrace
#Define different colors for data flows (for NAM)
$ns color 1 Red
$ns color 2 Cyan
#Open the ns trace file
set nf [open out.nam w]
$ns namtrace-all $nf
set files(0)      [open tcpNewreno.tr w]
set files(1)      [open Udp.tr w]
set val(stop)   200  ;# time of simulation end
set val(nn)     8    ;# number of nodes
# Define a 'finish' procedure
```

Appendix

```
proc stop {} {
   global ns nf files
   $ns flush-trace
# Close the NAM trace file
   close $nf
# Close the output file
   close $files(0)
   close $files(1)
#Execute xgraph tp display the result
exec xgraph tcpNewreno.tr Udp.tr -geometry 800*400 &
#Execute NAM  on trace file
exec nam out.nam &
   exit 0
}
#Create 8 nodes
for {set i 0} {$i < $val(nn)} {incr i} {
   set n($i) [$ns node]
}
#Create a duplex link between the nodes
$ns duplex-link $n(0) $n(2) 5Mb 10ms DropTail
$ns duplex-link $n(2) $n(4) 10Mb 10ms DropTail
$ns duplex-link $n(4) $n(5) 5Mb 5ms DropTail
$ns duplex-link $n(5) $n(6) 5Mb 10ms DropTail
$ns duplex-link $n(1) $n(3) 10Mb 5ms DropTail
$ns duplex-link $n(3) $n(4) 10Mb 10ms DropTail
$ns duplex-link $n(5) $n(7) 5Mb 10ms DropTail
#Set node position in nam
$ns duplex-link-op $n(0) $n(2) orient right-down
```

Appendix

```
$ns duplex-link-op $n(2) $n(4) orient right-down
$ns duplex-link-op $n(4) $n(5) orient right
$ns duplex-link-op $n(5) $n(6) orient right-up
$ns duplex-link-op $n(1) $n(3) orient rightp-u
$ns duplex-link-op $n(3) $n(4) orient right-up
$ns duplex-link-op $n(5) $n(7) orient right-down
#Create a TCP agent and attach it to node n0.The flow between n0 and n6
set src_tcp [new Agent/TCP]
$ns attach-agent $n(0) $src_tcp
$src_tcp set class_ 1
$src_tcp set maxcwnd_ 64  ;# max congestion window size
#Create sink node and attach it to node 6 (n6)
set sink_tcp [new Agent/TCPSink/DelAck]
$ns attach-agent $n(6) $sink_tcp
$ns connect $src_tcp $sink_tcp
#Setup FTP over TCP connection
set ftp [new Application/FTP]
$ftp set packetSize_ 1024
$ftp attach-agent $src_tcp
#$ns at 0.5 "$tcp advanceby 1"
$ns at 0.0 "$ftp start"
# Create UDP agent and attach it to node 1.UDP flow between n1 and n7
set src_udp [new Agent/UDP]
$ns attach-agent $n(1) $src_udp
$src_udp set class_ 2
#Create sink node abd attach to node 7 (n7)
set sink_udp [new Agent/LossMonitor]
$ns attach-agent $n(7) $sink_udp
```

Appendix

```
$ns connect $src_udp $sink_udp
#Setup a CBR over UDP connection
set cbr [new Application/Traffic/CBR]
$cbr set packetSize_ 1024
$cbr set rate_ 0.2Mb
$cbr attach-agent $src_udp
$ns at 0.0 "$cbr start"

# printing throughput of each client and channel utilization
set last_udp 0.0
set last_tcp 0.0
proc plotFiles {} {
    global ns files last_udp last_tcp sink_udp sink_tcp val
    set time 0.1
    set now [$ns now]
    set total 0.0
    set cur_tcp [$sink_tcp set bytes_]
    set temp [expr $cur_tcp-$last_tcp]
    puts $files(0) "$now [expr $temp*8/$time]"
    set total [expr $total+$temp]
    set cur_udp [$sink_udp set bytes_]
    set temp [expr $cur_udp-$last_udp]
    puts $files(1) "$now [expr $temp*8/$time]"
    set total [expr $total+$temp]
    set last_udp $cur_udp
    set last_tcp $cur_tcp
    set nextTime [expr $now+$time]
    if { $nextTime <= $val(stop) } {
```

```
        $ns at $nextTime "plotFiles"
    }
}
$ns at 0.1 "plotFiles"
# telling nodes when the simulation ends
for {set i 0} {$i < $val(nn)} {incr i} {
    $ns at $val(stop) "$n($i) reset";
}
$ns at $val(stop) "$ftp stop"
$ns at $val(stop) "$cbr stop"
$ns at $val(stop) "stop"
$ns at 200.1 "puts \"end simulation\"; $ns halt"
$ns at 5.0 "stop"
#Run the simulation
$ns run
```

3.2 Script(4Mb rate of UDP)

```
#Create a simulator object
set ns [new Simulator]
$ns use-newtrace
#Define different colors for data flows (for NAM)
$ns color 1 red
$ns color 2 Cyan
#Open the ns trace file
set nf [open out.nam w]
$ns namtrace-all $nf
```

Appendix

```
 set files(0)      [open tcpNewreno.tr w]
set files(1)      [open Udp.tr w]
 set val(stop)   200  ;# time of simulation end
set val(nn)     8    ;# number of nodes
# Define a 'finish' procedure
proc stop {} {
    global ns nf files
    $ns flush-trace
   # Close the NAM trace file
    close $nf
     # Close the output file
    close $files(0)
    close $files(1)
#Execute xgraph tp display the result
exec xgraph tcpNewreno.tr Udp.tr -geometry 800*400 &
#Execute NAM  on trace file
exec nam out.nam &
  exit 0
 }
 #Create 8 nodes
for {set i 0} {$i < $val(nn)} {incr i} {
   set n($i) [$ns node]
} #Create a duplex link between the nodes
$ns duplex-link $n(0) $n(2) 5Mb 10ms DropTail
$ns duplex-link $n(2) $n(4) 10Mb 10ms DropTail
$ns duplex-link $n(4) $n(5) 5Mb 5ms DropTail
```

Appendix

```
$ns duplex-link $n(5) $n(6) 5Mb 10ms DropTail
$ns duplex-link $n(1) $n(3) 10Mb 5ms DropTail
$ns duplex-link $n(3) $n(4) 10Mb 10ms DropTail
$ns duplex-link $n(5) $n(7) 5Mb 10ms DropTail
 #Set node position in nam
$ns duplex-link-op $n(0) $n(2) orient right-down
$ns duplex-link-op $n(2) $n(4) orient right-down
$ns duplex-link-op $n(4) $n(5) orient right
$ns duplex-link-op $n(5) $n(6) orient right-up
$ns duplex-link-op $n(1) $n(3) orient rightp-u
$ns duplex-link-op $n(3) $n(4) orient right-up
$ns duplex-link-op $n(5) $n(7) orient right-down
#Create a TCP agent and attach it to node n0.The flow between n0 and n6
set src_tcp [new Agent/TCP/Newreno]
$ns attach-agent $n(0) $src_tcp
$src_tcp set class_ 1
$src_tcp set maxcwnd_ 64  ;# max congestion window size
#Create sink node and attach it to node 6 (n6)
set sink_tcp [new Agent/TCPSink/DelAck]
$ns attach-agent $n(6) $sink_tcp
$ns connect $src_tcp $sink_tcp
 #Setup FTP over TCP connection
set ftp [new Application/FTP]
$ftp set packetSize_ 1024
$ftp attach-agent $src_tcp
 #$ns at 0.5 "$tcp advanceby 1"
```

Appendix

```
$ns at 0.0 "$ftp start"
 # Create UDP agent and attach it to node 1.UDP flow between n1 and n7
set src_udp [new Agent/UDP]
$ns attach-agent $n(1) $src_udp
$src_udp set class_ 2
#Create sink node abd attach to node 7 (n7)
set sink_udp [new Agent/LossMonitor]
$ns attach-agent $n(7) $sink_udp
$ns connect $src_udp $sink_udp
#Setup a CBR over UDP connection
set cbr [new Application/Traffic/CBR]
$cbr set packetSize_ 1024
$cbr set rate_ 4Mb
$cbr attach-agent $src_udp
$ns at 0.0 "$cbr start"
# printing throughput of each client and channel utilization
set last_udp 0.0
set last_tcp 0.0
  proc plotFiles {} {
    global ns files last_udp last_tcp sink_udp sink_tcp val
    set time 0.1
    set now [$ns now]
    set total 0.0
    set cur_tcp [$sink_tcp set bytes_]
    set temp [expr $cur_tcp-$last_tcp]
    puts $files(0) "$now [expr $temp*8/$time]"
```

Appendix

```
    set total [expr $total+$temp]
    set cur_udp [$sink_udp set bytes_]
    set temp [expr $cur_udp-$last_udp]
    puts $files(1) "$now [expr $temp*8/$time]"
    set total [expr $total+$temp]
  set last_udp $cur_udp
    set last_tcp $cur_tcp
     set nextTime [expr $now+$time]
    if { $nextTime <= $val(stop) } {
        $ns at $nextTime "plotFiles"
    }
}
$ns at 0.1 "plotFiles"
 # telling nodes when the simulation ends
for {set i 0} {$i < $val(nn)} {incr i} {
    $ns at $val(stop) "$n($i) reset";
}
 $ns at $val(stop) "$ftp stop"
$ns at $val(stop) "$cbr stop"
$ns at $val(stop) "stop"
$ns at 200.1 "puts \"end simulation\"; $ns halt"
 $ns at 5.0 "stop"
#Run the simulation
$ns run
```

4.VEGAS.

4.1 Script(200Kb rate of UDP)

```
#Create a simulator object
set ns [new Simulator]
$ns use-newtrace
#Define different colors for data flows (for NAM)
$ns color 1 Red
$ns color 2 Cyan
#Open the ns trace file
set nf [open out.nam w]
$ns namtrace-all $nf

set files(0)      [open tcpVegas.tr w]
set files(1)      [open Udp.tr w]
set val(stop)   200  ;# time of simulation end
set val(nn)     8    ;# number of nodes
# Define a 'finish' procedure
proc stop {} {
   global ns nf files
   $ns flush-trace
# Close the NAM trace file
   close $nf
# Close the output file
   close $files(0)
   close $files(1)
#Execute xgraph tp display the result
```

Appendix

```
exec xgraph tcpVegas.tr Udp.tr -geometry 800*400 &
#Execute NAM  on trace file
exec nam out.nam &
 exit 0
}
#Create 8 nodes
for {set i 0} {$i < $val(nn)} {incr i} {
    set n($i) [$ns node]
}
#Create a duplex link between the nodes
$ns duplex-link $n(0) $n(2) 5Mb 10ms DropTail
$ns duplex-link $n(2) $n(4) 10Mb 10ms DropTail
$ns duplex-link $n(4) $n(5) 5Mb 5ms DropTail
$ns duplex-link $n(5) $n(6) 5Mb 10ms DropTail
$ns duplex-link $n(1) $n(3) 10Mb 5ms DropTail
$ns duplex-link $n(3) $n(4) 10Mb 10ms DropTail
$ns duplex-link $n(5) $n(7) 5Mb 10ms DropTail
#Set node position in nam
$ns duplex-link-op $n(0) $n(2) orient right-down
$ns duplex-link-op $n(2) $n(4) orient right-down
$ns duplex-link-op $n(4) $n(5) orient right
$ns duplex-link-op $n(5) $n(6) orient right-up
$ns duplex-link-op $n(1) $n(3) orient rightp-u
$ns duplex-link-op $n(3) $n(4) orient right-up
$ns duplex-link-op $n(5) $n(7) orient right-down
#Create a TCP agent and attach it to node n0.The flow between n0 and n6
set src_tcp [new Agent/TCP]
$ns attach-agent $n(0) $src_tcp
```

Appendix

```
$src_tcp set class_ 1
$src_tcp set maxcwnd_ 64  ;# max congestion window size
#Create sink node and attach it to node 6 (n6)
set sink_tcp [new Agent/TCPSink/DelAck]
$ns attach-agent $n(6) $sink_tcp
$ns connect $src_tcp $sink_tcp
#Setup FTP over TCP connection
set ftp [new Application/FTP]
$ftp set packetSize_ 1024
$ftp attach-agent $src_tcp
#$ns at 0.5 "$tcp advanceby 1"
$ns at 0.0 "$ftp start"
# Create UDP agent and attach it to node 1.UDP flow between n1 and n7
set src_udp [new Agent/UDP]
$ns attach-agent $n(1) $src_udp
$src_udp set class_ 2
#Create sink node abd attach to node 7 (n7)
set sink_udp [new Agent/LossMonitor]
$ns attach-agent $n(7) $sink_udp
$ns connect $src_udp $sink_udp
#Setup a CBR over UDP connection
set cbr [new Application/Traffic/CBR]
$cbr set packetSize_ 1024
$cbr set rate_ 0.2Mb
$cbr attach-agent $src_udp
$ns at 0.0 "$cbr start"
# printing throughput of each client and channel utilization
   set last_udp 0.0
```

Appendix

```
    set last_tcp 0.0
    proc plotFiles {} {
    global ns files last_udp last_tcp sink_udp sink_tcp val
    set time 0.1
    set now [$ns now]
    set total 0.0
    set cur_tcp [$sink_tcp set bytes_]
    set temp [expr $cur_tcp-$last_tcp]
    puts $files(0) "$now [expr $temp*8/$time]"
    set total [expr $total+$temp]
     set cur_udp [$sink_udp set bytes_]
    set temp [expr $cur_udp-$last_udp]
    puts $files(1) "$now [expr $temp*8/$time]"
    set total [expr $total+$temp]
 set last_udp $cur_udp
 set last_tcp $cur_tcp
    set nextTime [expr $now+$time]
    if { $nextTime <= $val(stop) } {
        $ns at $nextTime "plotFiles"
    }
}
$ns at 0.1 "plotFiles"
# telling nodes when the simulation ends
for {set i 0} {$i < $val(nn)} {incr i} {
    $ns at $val(stop) "$n($i) reset";
}
$ns at $val(stop) "$ftp stop"
$ns at $val(stop) "$cbr stop"
```

```
$ns at $val(stop) "stop"
$ns at 200.1 "puts \"end simulation\"; $ns halt"
$ns at 5.0 "stop"
#Run the simulation
$ns run
```

4.2 Script(4Mb rate of UDP)

```
        #Create a simulator object
        set ns [new Simulator]
$ns use-newtrace
        #Define different colors for data flows (for NAM)
        $ns color 1 red
        $ns color 2 Cyan
         #Open the ns trace file
        set nf [open out.nam w]
        $ns namtrace-all $nf
         set files(0)      [open tcpVegas.tr w]
        set files(1)      [open Udp.tr w]
         set val(stop)   200  ;# time of simulation end
        set val(nn)     8    ;# number of nodes
        # Define a 'finish' procedure
        proc stop {} {
         global ns nf files
         $ns flush-trace
           # Close the NAM trace file
```

```
  close $nf
   # Close the output file
   close $files(0)
 close $files(1)
#Execute xgraph tp display the result
exec xgraph tcpVegas.tr Udp.tr -geometry 800*400 &
 #Execute NAM  on trace file
exec nam out.nam &
 exit 0
}
 #Create 8 nodes
for {set i 0} {$i < $val(nn)} {incr i} {
    set n($i) [$ns node]
}
 #Create a duplex link between the nodes
$ns duplex-link $n(0) $n(2) 5Mb 10ms DropTail
$ns duplex-link $n(2) $n(4) 10Mb 10ms DropTail
$ns duplex-link $n(4) $n(5) 5Mb 5ms DropTail
$ns duplex-link $n(5) $n(6) 5Mb 10ms DropTail
$ns duplex-link $n(1) $n(3) 10Mb 5ms DropTail
$ns duplex-link $n(3) $n(4) 10Mb 10ms DropTail
$ns duplex-link $n(5) $n(7) 5Mb 10ms DropTail
 #Set node position in nam
$ns duplex-link-op $n(0) $n(2) orient right-down
$ns duplex-link-op $n(2) $n(4) orient right-down
$ns duplex-link-op $n(4) $n(5) orient right
```

```
$ns duplex-link-op $n(5) $n(6) orient right-up
$ns duplex-link-op $n(1) $n(3) orient rightp-u
$ns duplex-link-op $n(3) $n(4) orient right-up
$ns duplex-link-op $n(5) $n(7) orient right-down
#Create a TCP agent and attach it to node n0.The flow between n0 and n6
set src_tcp [new Agent/TCP/Vegas]
$ns attach-agent $n(0) $src_tcp
$src_tcp set class_ 1
$src_tcp set maxcwnd_ 64  ;# max congestion window size
  #Create sink node and attach it to node 6 (n6)
set sink_tcp [new Agent/TCPSink/DelAck]
$ns attach-agent $n(6) $sink_tcp
$ns connect $src_tcp $sink_tcp
 #Setup FTP over TCP connection
set ftp [new Application/FTP]
$ftp set packetSize_ 1024
$ftp attach-agent $src_tcp
  #$ns at 0.5 "$tcp advanceby 1"
$ns at 0.0 "$ftp start"
  # Create UDP agent and attach it to node 1.UDP flow between n1 and n7
set src_udp [new Agent/UDP]
$ns attach-agent $n(1) $src_udp
$src_udp set class_ 2
 #Create sink node abd attach to node 7 (n7)
set sink_udp [new Agent/LossMonitor]
$ns attach-agent $n(7) $sink_udp
```

```
$ns connect $src_udp $sink_udp
  #Setup a CBR over UDP connection
set cbr [new Application/Traffic/CBR]
$cbr set packetSize_ 1024
$cbr set rate_ 4Mb
$cbr attach-agent $src_udp
$ns at 0.0 "$cbr start"
 # printing throughput of each client and channel utilization
set last_udp 0.0
set last_tcp 0.0
  proc plotFiles {} {
 global ns files last_udp last_tcp sink_udp sink_tcp val
   set time 0.1
    set now [$ns now]
    set total 0.0
     set cur_tcp [$sink_tcp set bytes_]
    set temp [expr $cur_tcp-$last_tcp]
    puts $files(0) "$now [expr $temp*8/$time]"
    set total [expr $total+$temp]
    set cur_udp [$sink_udp set bytes_]
    set temp [expr $cur_udp-$last_udp]
    puts $files(1) "$now [expr $temp*8/$time]"
    set total [expr $total+$temp]
   set last_udp $cur_udp
    set last_tcp $cur_tcp
     set nextTime [expr $now+$time]
```

Appendix

```
    if { $nextTime <= $val(stop) } {
        $ns at $nextTime "plotFiles"
    }
}
$ns at 0.1 "plotFiles"
# telling nodes when the simulation ends
for {set i 0} {$i < $val(nn)} {incr i} {
    $ns at $val(stop) "$n($i) reset";
}
  $ns at $val(stop) "$ftp stop"
$ns at $val(stop) "$cbr stop"
$ns at $val(stop) "stop"
$ns at 200.1 "puts \"end simulation\"; $ns halt"
 $ns at 5.0 "stop"
 #Run the simulation
$ns run
```

Appendix C installation steps of NS2

Ns2.34 installation steps.

1. “tar xvzf ns-allinone-2.34-gcc32.tar.gz”,

   followed by “cd ns-allinone-2.34”
2. Suppose you are in “/home/shhd/ns-allinone-2.34”.

   Execute “./install”,
3. Input the following lines to the “bashrc” file. NSALL=/home/shhd/ns-allinone-2.34

   otcl=/home/shhd/ns-allinone-2.34/otcl-1.0a5 tclcl=/home/shhd/ns-allinone-2.34/tclcl-1.0b9 TCL=/home/shhd/ns-allinone-2.34/tclbox TK=/home/shhd/ns-allinone-2.34/tkbox export PATH=“$PATH:/home/shhd/ns-allinone-2.34/bin:/home/shhd/ns-allinone-2.34/tcl8.0.4/unix:/home/shhd/ns-allinone-2.34/tk8.0.4/unix”

export LD_LIBRARY_PATH=“$LD_LIBRARY_PATH:/home/shhd/ns-allinone-2.34/otcl-1.0a5:/home/shhd/ns-allinone-2.34/lib” export TCL_LIBRARY=“$TCL_LIBRARY:/home/shhd/ns-allinone-2.34/tcl8.0.4/lbrary” Save then quit the “bashrc” file. Enter “/home/shhd/ns-allinone-2.34/ns-2.34” and run “./validate” to validate if the installation is complete.

4.Validate it

$ cd ns-2.33

$ ./validate

Thus ns-allinone-2.34 installation has come to an end.