



Policy-based Adaptation Rate Control for TCP Traffic

By

Ms. Dujdao Krisboonchu

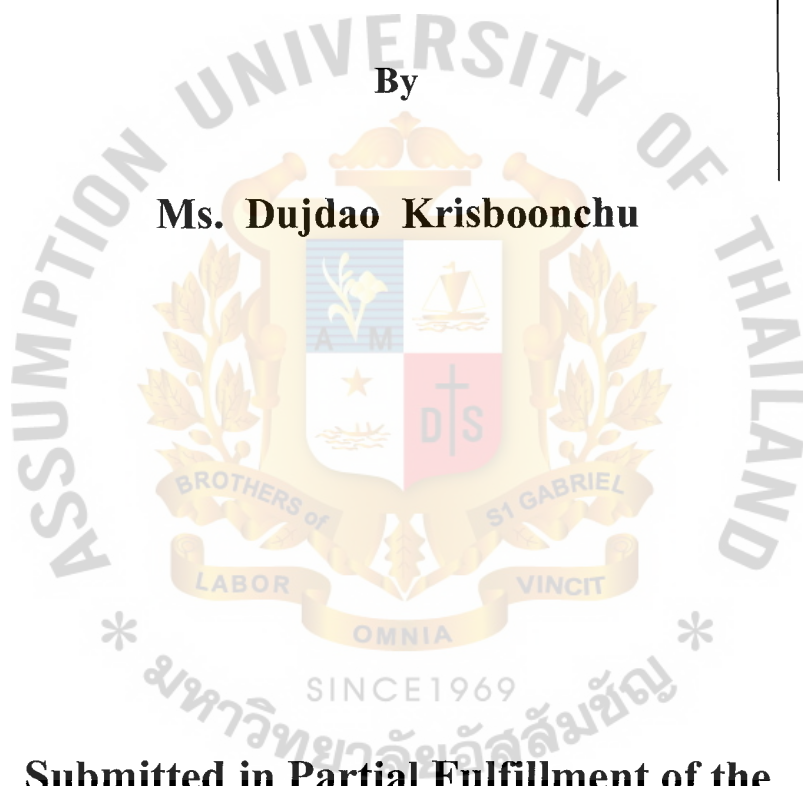
Submitted in Partial Fulfillment of the
Requirements for the Degree of
Master of Science
in Telecommunications Science
Assumption University

October, 2003

Policy-based Adaptation Rate Control for TCP Traffic

By

Ms. Dujdao Krisboonchu



**Submitted in Partial Fulfillment of the
Requirement for the Degree of
Master of Science in
Telecommunications Science
Assumption University**

October , 2003

The Faculty of Science and Technology


Master Thesis Approval


Thesis Title Policy-based Adaptation Rate Control for TCP Traffic

By Ms. Dujdao Krisboonchu
Thesis Advisor Dr. Surat Tanterdtid
Academic Year 1/2003


The Department of Telecommunications Science, Faculty of Science and Technology of Assumption University has approved this final report of the **twelve** credits course. **TS7000 Master Thesis**, submitted in partial fulfillment of the requirements for the degree of Master of Science in Telecommunications Science.

Approval Committee:

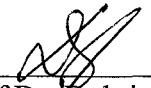

(Dr. Surat Tanterdtid)
Advisor

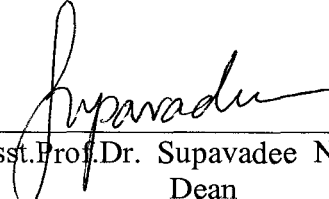

(Asst.Prof.Dr. Chanintorn J. Nukoon)
Committee Member


(Asst.Prof.Dr. Dobri Batovski)
Committee Member


(Asst.Prof.Dr. Surapong Auwatanamongkol)
Commission of Higher Education
University Affairs

Faculty Approval:


(Asst.Prof.Dr. Dobri Batovski)
Program Director


(Asst.Prof.Dr. Supavadee Nontakao)
Dean

October / 2003

ABSTRACT

TCP rate control is a new technique for transparently augmenting end-to-end TCP performance by controlling the sending rate of a TCP source. The sending rate of TCP source is determined by its window size, the round trip time and the rate of acknowledgment. It controls the rate of TCP packets by controlling window size and the rate of acknowledgment based on congestion environment.

This thesis presented the comparison of the simulation of TCP rate control and standard TCP. This thesis separated the simulation into 2 categories, single hop and multi hop topology with various link speeds. In the simulation of both topologies with 14 cases, the adjustment of the bandwidth is presented to make a congested environment. The result of both topologies has shown that using TCP rate control technology give higher performance than standard TCP in term of sending more TCP packets in congested environment.

ACKNOWLEDGMENT

The author is very much indebted to Dr. Surat Tanterdtid, the thesis advisor, for his help, suggestions and encouragement during my study. The author owes a great deal to Asst.Prof.Dr.Chanintorn J. Nukoon and Asst.Prof.Dr.Dobri Batovski for their helpful recommendations.

The author would like to thank Telecommunication Science Department, Graduate School of Assumption University for supporting the research.

Special thanks are due to Mr. Chanwit Chavalitkitjaroen for all his support.

The author would like to dedicate all of the expected benefits that are obtained from this dissertation to his father, Mr. Vinij Krisboonchu, who has been a great moral supporter during the author's study.

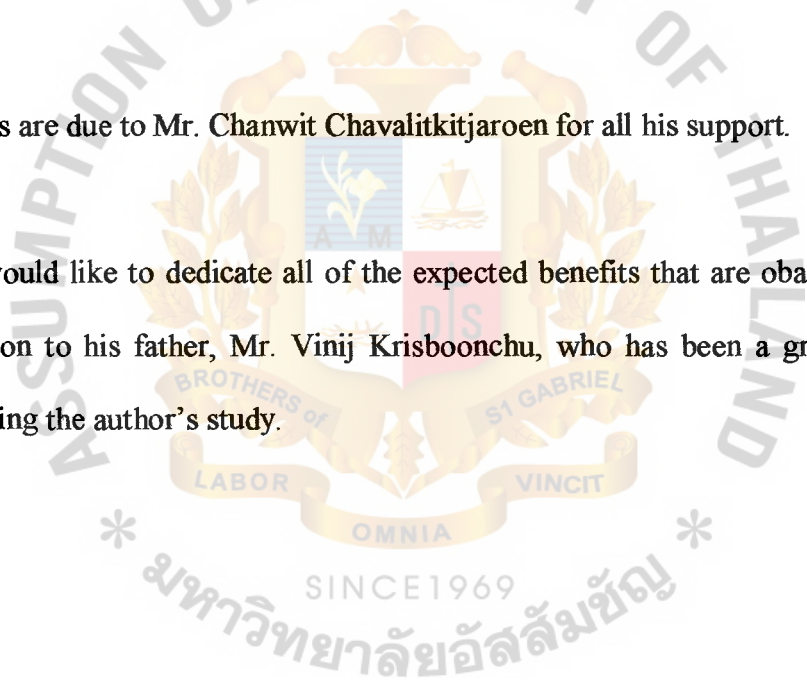


TABLE OF CONTENTS

ABSTRACT	i
ACKNOWLEDGMENT	ii
TABLE OF CONTENTS	iii
LIST OF FIGURES	v
LIST OF TABLES	viii
CHAPTER	
CHAPTER 1 INTRODUCTION	
1.1 TCP overview	1
1.2 Background	13
1.3 Motivation	14
1.4 Problem Statement	14
1.5 Goals and objectives	15
CHAPTER 2 LITERATURE REVIEWS	
2.1 Transmission Control Protocol	16
2.2 TCP Slow Start, Congestion Avoidance, Fast Retransmit, and Fast Recovery	21
2.3 Simulation-based Comparison of Tahoe, Reno and SACK TCP	24
2.4 Selective Acknowledgment	26
2.5 Comparison Study of RED, ECN and TCP Rate Control	27
2.6 TCP Rate Control	28
CHAPTER 3 PROPOSED SYSTEM	
3.1 Proposed Algorithm	29
3.2 Scope and Limitation	34
3.3 Methodology	35

TABLE OF CONTENTS (CONTINUE)

CHAPTER 4 RESULTS

4.1 Tested Topologies and Parameters for Single Hop and Multi Hop Topology	36
4.2 Evaluation of Single Hop Topology for Fast Ethernet (100BaseT)	40
4.3 Effect of Reducing the Bandwidth for Single Hop Topology	44
4.3.1 Evaluation of Reducing the Bandwidth to Standard Ethernet (10BaseT)	44
4.3.2 Evaluation of Reducing the Bandwidth to Low Bandwidth	47
4.3.3 Discussion of Effect of Reducing the Bandwidth for Single Hop Topology	49
4.4 Evaluation of Multi Hop Topology for Standard Ethernet (10BaseT)	50
4.5 Effect of Reducing the Bandwidth for Multi Hop Topology	53
4.5.1 Evaluation of Reducing the Bandwidth to Low Bandwidth	53
4.5.2 Discussion of Effect of Reducing the Bandwidth for Multi Hop Topology	58
4.6 Comparative Results of TCP Rate Control and Standard TCP	
4.6.1 Discussion of Single Hop Topology	60
4.6.2 Discussion of Multi Hop Topology	62
4.6.3 Summary Discussion of Results	63

CHAPTER 5 CONCLUSION AND RECOMMENDATION	64
---	----

REFERENCE	66
-----------	----

APPENDIX	68
----------	----

LIST OF FIGURES

Figure 1-1	TCP Flow Control Mechanisms	2
Figure 1-2	TCP Header	4
Figure 1-3	TCP State Transition Diagram	7
Figure 1-4	TCP Sliding Window	9
Figure 1-5	Jacobson/Karels Algorithm	12
Figure 2-1	TCP Packet Format	16
Figure 2-2	TCP Three Way Handshake	20
Figure 2-3	Packet Loss and Retransmission	20
Figure 2-4	Slow Start	22
Figure 2-5	Selective Acknowledgment (SACK)	27
Figure 3-1	Single Hop Topology	34
Figure 3-2	Multi Hop Topology	34
Figure 4-1	Topology A – The switch is connected to the server at 100 Mbps	37
Figure 4-2	Topology B - The switch is connected to the server at 10 Mbps	37
Figure 4-3	Topology C - The switch is connected to the server at 1 Mbps	38
Figure 4-4	Topology D - The bandwidth between switches is 10Mbps	38
Figure 4-5	Topology E -The bandwidth between switches is 5 Mbps	38
Figure 4-6	Topology F - The bandwidth between switches is 1 Mbps	39
Figure 4-7	Comparison of total sending and receiving packets for standard TCP and TCP Rate Control of case 1	41
Figure 4-8	Comparison of total sending and receiving packets for standard TCP and TCP Rate Control of case 2	42
Figure 4-9	Comparison of total sending and receiving packets for standard TCP and TCP Rate Control of case 3	42

LIST OF FIGURES

Figure 4-10	Comparison of total sending and receiving packets for standard TCP and TCP Rate Control of case 4	45
Figure 4-11	Comparison of total sending and receiving packets for standard TCP and TCP Rate Control of case 5	46
Figure 4-12	Comparison of total sending and receiving packets for standard TCP and TCP Rate Control of case 6	46
Figure 4-13	Comparison of total sending and receiving packets for standard TCP and TCP Rate Control of case 7	48
Figure 4-14	Comparison of total sending and receiving packets for standard TCP and TCP Rate Control of case 8	49
Figure 4-15	Comparison of total sending and receiving packets for standard TCP and TCP Rate Control of case 9	51
Figure 4-16	Comparison of total sending and receiving packets for standard TCP and TCP Rate Control of case 10	52
Figure 4-17	Comparison of total sending and receiving packets for standard TCP and TCP Rate Control of case 11	54
Figure 4-18	Comparison of total sending and receiving packets for standard TCP and TCP Rate Control of case 12	55
Figure 4-19	Comparison of total sending and receiving packets for standard TCP and TCP Rate Control of case 13	57
Figure 4-20	Comparison of total sending and receiving packets for standard TCP and TCP Rate Control of case 14	57

LIST OF FIGURES

Figure 4-21 The percentage coefficient differentiation between TCP rate control 61 and standard TCP for case study 9-14

Figure 4-22 The percentage coefficient differentiation between TCP rate control 63 and standard TCP for case study 9-14



LIST OF TABLES

Table 4-1	Number of TCP and UDP sources for Case 1-3	40
Table 4-2	Percentage of TCP and UDP sources and comparison percentage of TCP and UDP received packet for Case 1-3	40
Table 4-3	Result of standard TCP for case 1-3	41
Table 4-4	Result of TCP Rate Control for case 1-3	41
Table 4-5	Number of TCP and UDP sources for Case 4-6	44
Table 4-6	Percentage of TCP and UDP sources and comparison percentage of TCP and UDP received packet for Case 4-6	44
Table 4-7	Result of standard TCP for case 4-6	45
Table 4-8	Result of TCP Rate Control for case 4-6	45
Table 4-9	Number of TCP and UDP sources for Case 7-8	47
Table 4-10	Percentage of TCP and UDP sources and comparison percentage of TCP and UDP received packet for Case 7-8	47
Table 4-11	Result of standard TCP for case 7-8	48
Table 4-12	Result of TCP Rate Control for case 7-8	48
Table 4.13	Number of TCP and UDP sources for Case 9-10	50
Table 4-14	Percentage of TCP and UDP sources and comparison percentage of TCP and UDP received packet for Case 9-10	50
Table 4-15	Result of standard TCP for case 9-10	51
Table 4-16	Result of TCP Rate Control for case 9-10	51
Table 4-17	Number of TCP and UDP sources for Case 11-12	53
Table 4-18	Percentage of TCP and UDP sources and comparison percentage of TCP and UDP received packet for Case 11-12	53
Table 4-19	Result of standard TCP for case 11-12	54

LIST OF TABLES (CONTINUE)

Table 4-20	Result of TCP Rate Control for case 11-12	54
Table 4-21	Number of TCP and UDP sources for Case 13-14	56
Table 4-22	Percentage of TCP and UDP sources and comparison percentage of TCP and UDP received packet for Case 13-14	56
Table 4-23	Result of standard TCP for case 13-14	56
Table 4-24	Result of TCP Rate Control for case 13-14	57
Table 4-25	TCP received packet comparison for case 1-8	60
Table 4-26	TCP received packet comparison for case 9 – 14	62



1. INTRODUCTION

1.1 TCP Overview

The TCP provides reliable transmission of data in an IP environment. TCP corresponds to the transport layer (Layer 4) of the OSI reference model. Among the services TCP provides are stream data transfer, reliability, efficient flow control, full-duplex operation, and multiplexing. With stream data transfer, TCP delivers an unstructured stream of bytes identified by sequence numbers. This service benefits applications because they do not have to chop data into blocks before handing it off to TCP. Instead, TCP groups bytes into segments and passes them to IP for delivery.

TCP offers reliability by providing connection-oriented, end-to-end reliable packet delivery through an inter network. It does this by sequencing bytes with a forwarding acknowledgment number that indicates to the destination the next byte the source expects to receive. Bytes not acknowledged within a specified time period are retransmitted. The reliability mechanism of TCP allows devices to deal with lost, delayed, duplicate, or misread packets. A time-out mechanism allows devices to detect lost packets and request retransmission. TCP offers efficient flow control, which means that, when sending acknowledgments back to the source, the receiving TCP process indicates the highest sequence number it can receive without overflowing its internal buffers. Full-duplex operation means that TCP processes can both send and receive at the same time. Finally, TCP's multiplexing means that numerous simultaneous upper-layer conversations can be multiplexed over a single connection.

TCP Summary

TCP provides a connection oriented, reliable, byte stream service. The term connection-oriented means the two applications using TCP must establish a TCP connection with each other before they can exchange data. It is a full duplex protocol, meaning that each TCP connection supports a pair of byte streams, one flowing in each direction. TCP includes a flow-control mechanism for each of these byte streams that allows the receiver to limit how much data the sender can transmit. TCP also implements a congestion-control mechanism.

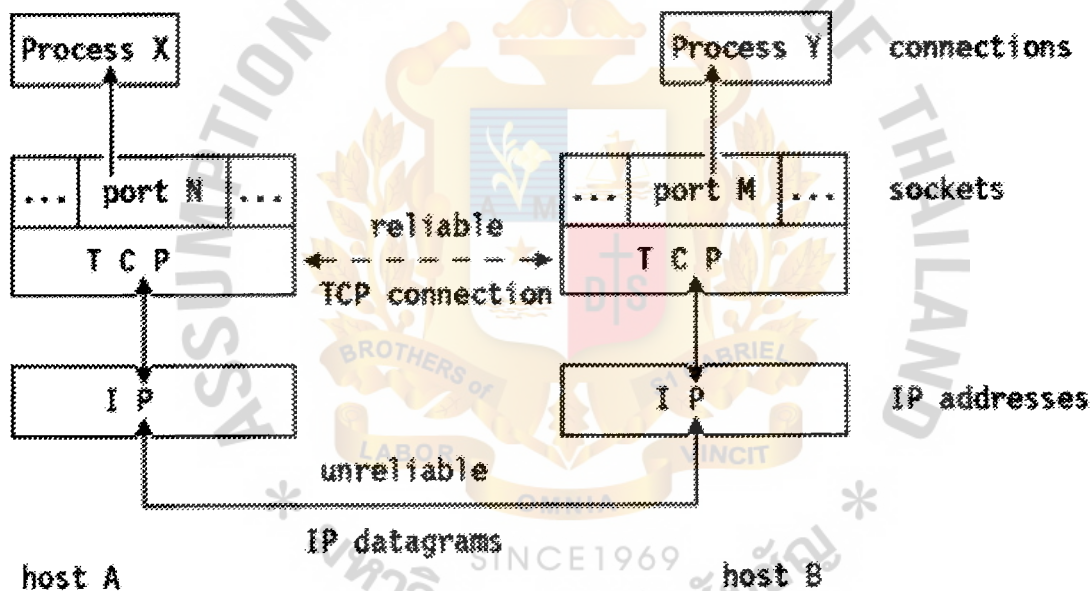


Figure 1-1: TCP Flow Control Mechanisms

Two processes communicating via TCP sockets. Each side of a TCP connection has a socket which can be identified by the pair $\langle IP_address, port_number \rangle$. Two processes communicating over TCP form a logical connection that is uniquely identifiable by the two sockets involved, that is by the combination $\langle local_IP_address, local_port, remote_IP_address, remote_port \rangle$.

TCP provides the following facilities to:

Stream Data Transfer

From the application's viewpoint, TCP transfers a contiguous stream of bytes. TCP does this by grouping the bytes in TCP segments, which are passed to IP for transmission to the destination. TCP itself decides how to segment the data and it may forward the data at its own convenience.

Reliability

TCP assigns a sequence number to each byte transmitted, and expects a positive acknowledgment (ACK) from the receiving TCP. If the ACK is not received within a timeout interval, the data is retransmitted. The receiving TCP uses the sequence numbers to rearrange the segments when they arrive out of order, and to eliminate duplicate segments.

Flow Control

The receiving TCP, when sending an ACK back to the sender, also indicates to the sender the number of bytes it can receive beyond the last received TCP segment, without causing overrun and overflow in its internal buffers. This is sent in the ACK in the form of the highest sequence number it can receive without problems.

Multiplexing

To allow for many processes within a single host to use TCP communication facilities simultaneously, the TCP provides a set of addresses or ports within each host. Concatenated with the network and host addresses from the internet communication layer, this forms a socket. A pair of sockets uniquely identifies each connection.

Logical Connections

The reliability and flow control mechanisms described above require that TCP initializes and maintains certain status information for each data stream. The combination of this status, including sockets, sequence numbers and window sizes, is called a logical connection. Each connection is uniquely identified by the pair of sockets used by the sending and receiving processes.

Full Duplex

TCP provides for concurrent data streams in both directions.

TCP Header

TCP data is encapsulated in an IP datagram. The figure shows the format of the TCP header. Its normal size is 20 bytes unless options are present. Each of the fields is discussed below:

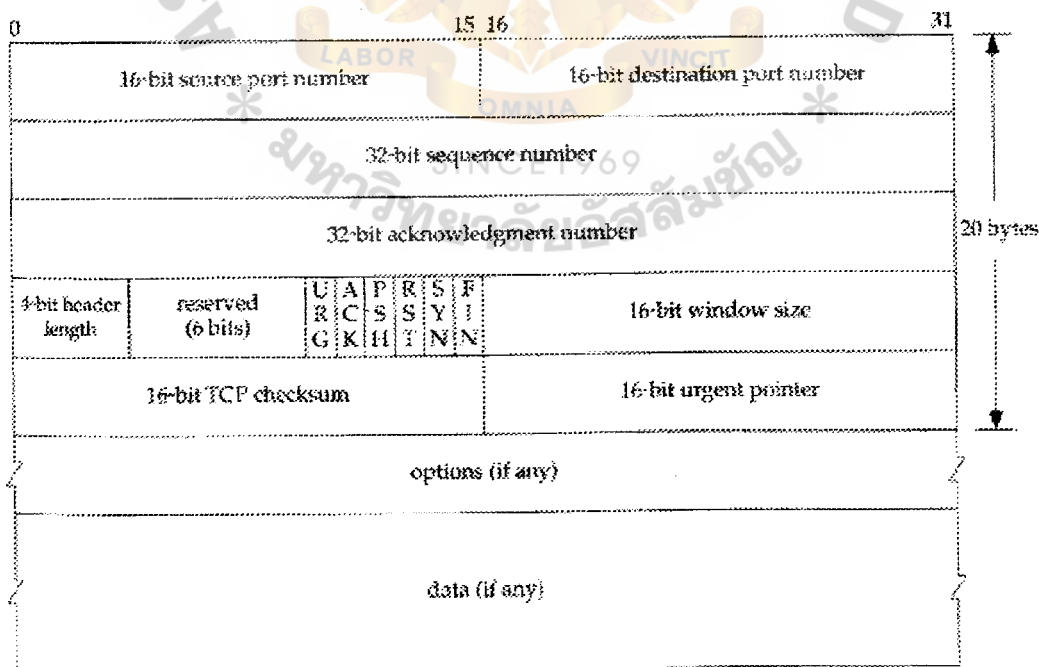


Figure 1-2: TCP Header

The **SrcPort** and **DstPort** fields identify the source and destination ports, respectively. These two fields plus the source and destination IP addresses, combine to uniquely identify each TCP connection.

The **sequence number** identifies the byte in the stream of data from the sending TCP to the receiving TCP that the first byte of data in this segment represents.

The **Acknowledgement number** field contains the next sequence number that the sender of the acknowledgement expects to receive. This is therefore the sequence number plus 1 of the last successfully received byte of data. This field is valid only if the ACK flag is on. Once a connection is established the ACK flag is always on.

The **Acknowledgement**, **SequenceNum**, and **AdvertisedWindow** fields are all involved in TCP's sliding window algorithm. The Acknowledgement and AdvertisedWindow fields carry information about the flow of data going in the other direction. In TCP's sliding window algorithm, the receiver advertises a window size to the sender. This is done using the AdvertisedWindow field. The sender is then limited to having no more than a value of AdvertisedWindow bytes of an acknowledged data at any given time. The receiver sets a suitable value for the AdvertisedWindow based on the amount of memory allocated to the connection for the purpose of buffering data.

The **header length** gives the length of the header in 32-bit words. This is required because the length of the options field is variable.

The 6-bit **Flags field** is used to relay control information between TCP peers.

The possible flags include SYN, FIN, RESET, PUSH, URG, and ACK.

- The SYN and Fin flags are used when establishing and terminating a TCP connection, respectively.

- The ACK flag is set any time the Acknowledgement field is valid, implying that the receiver should pay attention to it.
- The URG flag signifies that this segment contains urgent data. When this flag is set, the UrgPtr field indicates where the non-urgent data contained in this segment begins.
- The PUSH flag signifies that the sender invoked the push operation, which indicates to the receiving side of TCP that it should notify the receiving process of this fact.
- Finally, the RESET flag signifies that the receiver has become confused and so wants to abort the connection.

The **Checksum** covers the TCP segment: the TCP header and the TCP data. This is a mandatory field that must be calculated by the sender, and then verified by the receiver.

The **Option field** is the maximum segment size option, called the MSS. Each end of the connection normally specifies this option on the first segment exchanged. It specifies the maximum sized segment the sender wants to receive.

The **data** portion of the TCP segment is optional.

TCP State Transition Diagram

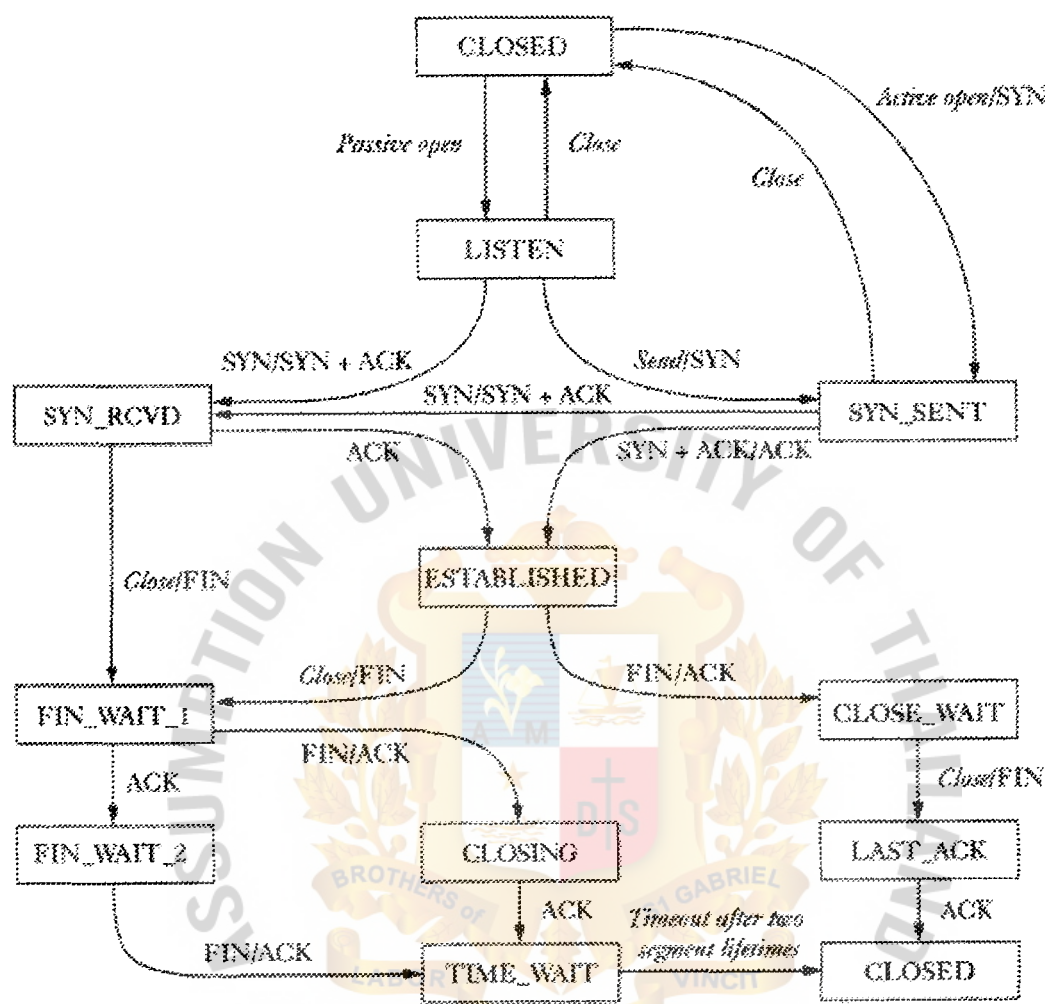


Figure 1-3: TCP State Transition Diagram

The two transitions leading to the ESTABLISHED state correspond to the opening of a connection, and the two transitions leading from the ESTABLISHED state are for the termination of a connection. The ESTABLISHED state is where data transfer can occur between the two ends in both the directions.

If a connection is in the LISTEN state and a SYN segment arrives, the connection makes a transition to the SYN_RCVD state and takes the action of replying with an ACK+SYN segment. The client does an active open which causes its end of the

connection to send a SYN segment to the server and to move to the SYN_SENT state. The arrival of the SYN+ACK segment causes the client to move to the ESTABLISHED state and to send an ack back to the server. When this ACK arrives the server finally moves to the ESTABLISHED state. In other words, we have just traced the THREE-WAY HANDSHAKE.

In the process of terminating a connection, the important thing to keep in mind is that the application process on both sides of the connection must independently close its half of the connection. Thus, on any one side there are three combinations of transition that get a connection from the ESTABLISHED state to the CLOSED state:

- This side closes first:

ESTABLISHED -> FIN_WAIT_1-> FIN_WAIT_2 -> TIME_WAIT -> CLOSED.

- The other side closes first:

ESTABLISHED -> CLOSE_WAIT -> LAST_ACK -> CLOSED.

- Both sides close at the same time:

ESTABLISHED -> FIN_WAIT_1-> CLOSING ->TIME_WAIT -> CLOSED.

The main thing to recognize about connection teardown is that a connection in the TIME_WAIT state cannot move to the CLOSED state until it has waited for two times the maximum amount of time an IP datagram might live in the Internet. The reason for this is that while the local side of the connection has sent an ACK in response to the other side's FIN segment, it does not know that the ACK was successfully delivered. As a consequence, this other side might retransmit its FIN segment, and this second FIN segment might be delayed in the network. If the

connection were allowed to move directly to the CLOSED state, then another pair of application processes might come along and open the same connection, and the delayed FIN segment from the earlier incarnation of the connection would immediately initiate the termination of the later incarnation of that connection.

Sliding Window

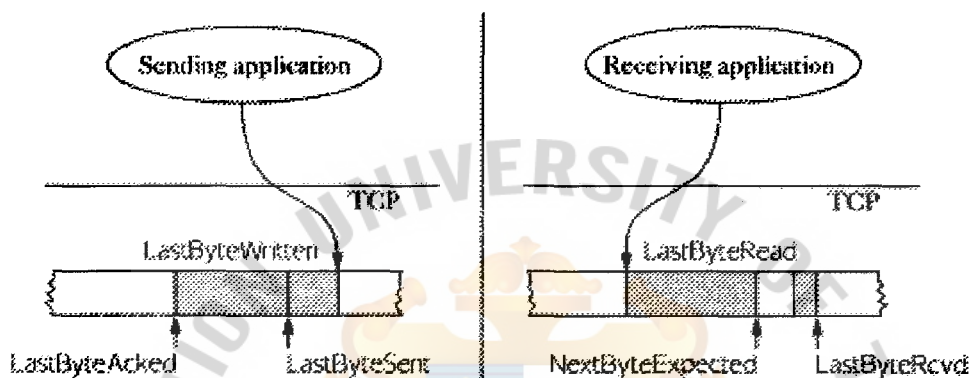


Figure 1-4: TCP Sliding Window

The sliding window serves several purposes:

- (1) it guarantees the reliable delivery of data
- (2) it ensures that the data is delivered in order,
- (3) it enforces flow control between the sender and the receiver.

Reliable and ordered delivery

The sending and receiving sides of TCP interact in the following manner to implement reliable and ordered delivery:

Each byte has a sequence number.

ACKs are cumulative.

Sending side

- $\text{LastByteAcked} \leq \text{LastByteSent}$
- $\text{LastByteSent} \leq \text{LastByteWritten}$
- bytes between LastByteAcked and LastByteWritten must be buffered.

Receiving side

- $\text{LastByteRead} < \text{NextByteExpected}$
- $\text{NextByteExpected} \leq \text{LastByteRcvd} + 1$
- bytes between NextByteRead and LastByteRcvd must be buffered.

Flow Control

Sender buffer size : MaxSendBuffer

Receive buffer size : MaxRcvBuffer

Receiving side

- $\text{LastByteRcvd} - \text{NextByteRead} \leq \text{MaxRcvBuffer}$
- $\text{AdvertisedWindow} = \text{MaxRcvBuffer} - (\text{LastByteRcvd} - \text{NextByteRead})$

Sending side

- $\text{LastByteSent} - \text{LastByteAcked} \leq \text{AdvertisedWindow}$
- $\text{EffectiveWindow} = \text{AdvertisedWindow} - (\text{LastByteSent} - \text{LastByteAcked})$

- $\text{LastByteWritten} - \text{LastByteAcked} \leq \text{MaxSendBuffer}$
- Block sender if $(\text{LastByteWritten} - \text{LastByteAcked}) + y > \text{MaxSendBuffer}$

Always send ACK in response to an arriving data segment

Persist when $\text{AdvertisedWindow} = 0$

Adaptive Retransmission

TCP guarantees reliable delivery and so it retransmits each segment if an ACK is not received in a certain period of time. TCP sets this timeout as a function of the RTT it expects between the two ends of the connection. Unfortunately, given the range of possible RTT's between any pair of hosts in the Internet, as well as the variation in RTT between the same two hosts over time, choosing an appropriate timeout value is not that easy. To address this problem, TCP uses an adaptive retransmission mechanism. We describe this mechanism and how it has evolved over time.

Original Algorithm

Measure SampleRTT for each segment/ACK pair

Compute weighted average of RTT

$\text{EstimatedRTT} = a * \text{EstimatedRTT} + b * \text{SampleRTT}$, where $a + b = 1$

a between 0.8 and 0.9

b between 0.1 and 0.2

Set timeout based on $\text{EstimatedRTT} * \text{TimeOut} = 2 * \text{EstimatedRTT}$

Karn/Partridge Algorithm

Do not sample RTT when retransmitting

Double timeout after each retransmission

Jacobson/Karels Algorithm

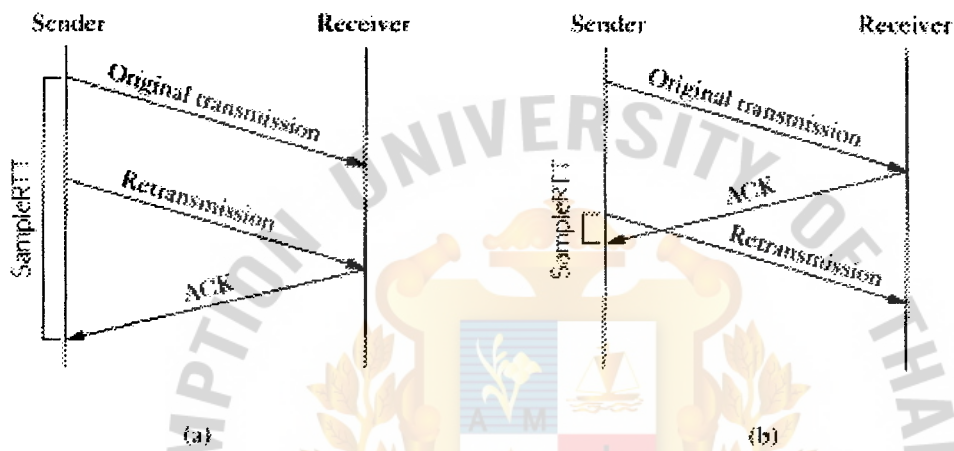


Figure 1-5: Jacobson/Karels Algorithm

New calculation for average RTT

Difference = SampleRTT - EstimatedRTT

EstimatedRTT = EstimatedRTT + (d * Difference)

Deviation = Deviation + d (|Difference| - Deviation)), where d is a fraction between 0 and 1

Consider variance when setting timeout value

Timeout = u * EstimatedRTT + q * Deviation, where u = 1 and q = 4

1.2 Background

TCP Rate Control represents a more fundamental and precise approach by applying explicit rate-based flow control to both individual and classes of traffic flows. TCP rate control increases network efficiency by avoiding retransmissions and packet loss.

Imagine putting fine sand, rather than gravel, through a network pipe. Sand can pass through the pipe more evenly and quickly than chunks. TCP Rate Control conditions traffics so that it becomes more like sand than gravel. By using rate-based flow control instead of queuing, TCP Rate Control evenly distributes packet transmissions by controlling TCP acknowledgments to the sender, causing the sender to throttle back, and avoiding packet tossing when there is insufficient bandwidth.

TCP Rate Control is a new technique for transparently augmenting end-to-end TCP performance by controlling the sending rate of a TCP source. The sending rate of a TCP source is determined by its window size, the round trip time and the rate of acknowledgements. TCP Rate Control affects these aspects by modifying the ack number and receiver window fields in acknowledgements and by modulating the acknowledgement rate. From a performance viewpoint, a key benefit of TCP rate control is to avoid adverse performance effects due to packet losses such as reduced goodput and unfairness or large spread in per-user goodput.

1.3 Motivation

TCP congestion control is designed for network stability, robustness and opportunistic use of network resources on an end-to-end basis. Using a robust technique to detect packet loss (timeout or triple-duplicate acks), TCP infers congestion and trades off per-user goodput for network stability. Specifically, TCP throughput is known to be a function, which is inversely proportional to the round trip time, the timeout delays and the square root of loss probability (Ignoring effects of small windows and timeout).

Given this equation, we can view the function of any buffer management algorithm managing TCP flows as assigning loss probabilities and queuing delays (which affect the round trip time) to competing TCP flows in order to meet performance requirements such as utilization, queuing delays, spread of per-user goodputs etc. However, this equation assumes that the TCP receiver window is not a limiting factor, which is not necessarily the case. Therefore if the TCP receiver window were the primary limiting factor, we could design a buffer management algorithm in which TCP throughput would not depend primarily on loss rate (or the round trip time) under controlled operating system.

1.4 Problem Statement

Computer networks have experienced an explosive growth and with that growth have come severe congestion problems. Now TCP uses slow-start algorithm for congestion control. With TCP slow-start, when a connection opens, only one packet is sent until an ACK is received. For each ACK receives ACK, the sender can double the transmission size. Note that this is exponential growth rate. But eventually packets are dropped.

With normal transmission of TCP, the sending rate is not based on congestion environment and when network is under the problem of loss packets and retransmission can occur. Dropped and retransmission packet can increase latency. This is the significant role of this thesis to point out TCP rate control based on congestion environment can reduce loss and retransmit packets and can decrease latency too.

1.5 Goals and Objectives

The goal for this thesis is to propose the methodology of TCP rate control that can improve the performance when transmitting TCP/IP packet and reduce the loss and retransmission packets. TCP rate control is a new technique for transparently augmenting end-to-end TCP performance by controlling the sending rate of a TCP source. The sending rate of TCP source is determined by its window size, the round trip time and the rate of acknowledgement. So this thesis will control the rate of TCP packets by controlling window size and the rate of acknowledgement based on congestion environment.

From the performance viewpoint, a key objective of TCP rate control is to avoid adverse performance effects due to packet losses such as reduced goodput and unfairness or large spread in per-user goodputs.

2. LITERATURE REVIEW

These are 6 major literatures which describe the TCP concept and TCP congestion control algorithms that are used in traditional TCP and their comparisons. The literatures describe the TCP congestion control and comparisons of Tahoe, Reno and SACK and describe the characteristic of SACK TCP.

2.1 Transmission Control Protocol [8]

This literature describes the protocol specification of Transmission Control Protocol (TCP).

TCP Connection Establishment

To use reliable transport services, TCP hosts must establish a connection-oriented session with one another. Connection establishment is performed by using a “three-way handshake” mechanism. A three-way handshake synchronizes both ends of a connection by allowing both sides to agree upon initial sequence numbers. This mechanism also guarantees that both sides are ready to transmit data and know that the other side is ready to transmit as well. This is necessary so that packets are not transmitted or retransmitted during session establishment or after session termination.

Each host randomly chooses a sequence number used to track bytes within the stream it is sending and receiving. Then, the three-way handshake proceeds in the following manner: The first host (Host A) initiates a connection by sending a packet with the initial sequence number (X) and SYN bit set to indicate a connection request. The second host (Host B) receives the SYN, records the sequence number X, and replies by

acknowledging the SYN (with an $ACK = X + 1$). Host B includes its own initial sequence number ($SEQ = Y$). An $ACK = 20$ means the host has received bytes 0 through 19 and expects byte 20 next. This technique is called *forward acknowledgment*. Host A then acknowledges all bytes Host B sent with a forward acknowledgment indicating the next byte Host A expects to receive ($ACK = Y + 1$). Data transfer then can begin.

Positive Acknowledgment and Retransmission

A simple transport protocol might implement a reliability-and-flow-control technique where the source sends one packet, starts a timer, and waits for an acknowledgment before sending a new packet. If the acknowledgment is not received before the timer expires, the source retransmits the packet. Such a technique is called *positive acknowledgment and retransmission* (PAR). By assigning each packet a sequence number, PAR enables hosts to track lost or duplicate packets caused by network delays that result in premature retransmission. The sequence numbers are sent back in the acknowledgments so that the acknowledgments can be tracked. PAR is an inefficient use of bandwidth, however, because a host must wait for an acknowledgment before sending a new packet, and only one packet can be sent at a time.

TCP Sliding Window

A *TCP sliding window* provides more efficient use of network bandwidth than PAR because it enables hosts to send multiple bytes or packets before waiting for an acknowledgment. In TCP, the receiver specifies the current window size in every packet. Because TCP provides a byte-stream connection, window sizes are expressed in bytes. This means that a window is the number of data bytes that the sender is allowed to send

before waiting for an acknowledgment. Initial window sizes are indicated at connection setup, but might vary throughout the data transfer to provide flow control. A window size of zero, for instance, means “Send no data.” In a TCP sliding-window operation, for example, the sender might have a sequence of bytes to send (numbered 1 to 10) to a receiver who has a window size of five. The sender then would place a window around the first five bytes and transmit them together. It would then wait for an acknowledgment. The receiver would respond with an ACK = 6, indicating that it has received bytes 1 to 5 and is expecting byte 6 next. In the same packet, the receiver would indicate that its window size is 5. The sender then would move the sliding window five bytes to the right and transmit bytes 6 to 10. The receiver would respond with an ACK = 11, indicating that it is expecting sequenced byte 11 next. In this packet, the receiver might indicate that its window size is 0 (because, for example, its internal buffers are full). At this point, the sender cannot send any more bytes until the receiver sends another packet with a window size greater than 0.

Source port		Destination port	
Sequence number			
Acknowledgment number			
Data offset	Reserved	Flags	Window
Checksum		Urgent pointer	
Options (+ padding)			
Data (variable)			

Figure 2-1: TCP Packet Format

TCP Packet Field Description

The following descriptions summarize the TCP packet fields:

- *Source Port* and *Destination Port*—Identifies points at which upper-layer source and destination processes receive TCP services.
- *Sequence Number*—Usually specifies the number assigned to the first byte of data in the current message. In the connection-establishment phase, this field also can be used to identify an initial sequence number to be used in an upcoming transmission.
- *Acknowledgment Number*—Contains the sequence number of the next byte of data the sender of the packet expects to receive.
- *Data Offset*—Indicates the number of 32-bit words in the TCP header.
- *Reserved*—Remains reserved for future use.
- *Flags*—Carries a variety of control information, including the SYN and ACK bits used for connection establishment, and the FIN bit used for connection termination.
- *Window*—Specifies the size of the sender's receive window (that is, the buffer space available for incoming data).
- *Checksum*—Indicates whether the header was damaged in transit.
- *Urgent Pointer*—Points to the first urgent data byte in the packet.
- *Options*—Specifies various TCP options.
- *Data*—Contains upper-layer information.

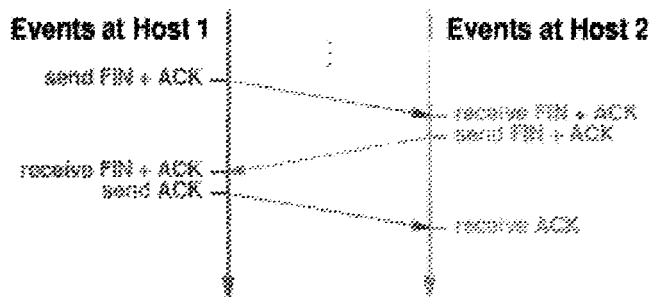


Figure 2-2: TCP Three Way Handshake

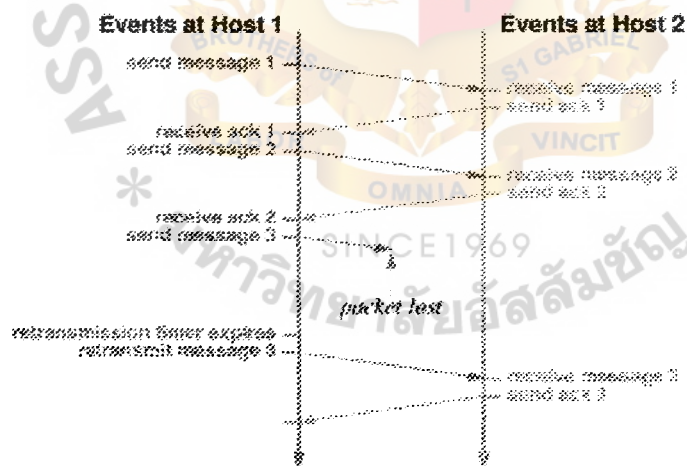


Figure 2-3: Packet Loss and Retransmission

Adaptive Retransmission

TCP guarantees reliable delivery and so it retransmits each segment if an ACK is not received in a certain period of time. TCP sets this timeout as a function of the RTT it expects between the two ends of the connection. Unfortunately, given the range of possible RTT's between any pair of hosts in the Internet, as well as the variation in RTT between the same two hosts over time, choosing an appropriate timeout value is not that easy. To address this problem, TCP uses an adaptive retransmission mechanism.

2.2 TCP Slow Start, Congestion Avoidance, Fast Retransmit, and Fast Recovery [1]

This literature describes the characteristics of congestion control algorithms for slow start, congestion avoidance, fast retransmit, and fast recovery.

Slow Start

- When a new connection is established, the congestion window, called “cwnd”, is initialized to one segment.
- The sender starts by transmitting one segment and waiting for its ACK.
- When that ACK is received, the congestion window is incremented from one to two, and two segments can be sent.
- When each of those two segments is acknowledged, the congestion window is increased to four.
- This provides an exponential growth.

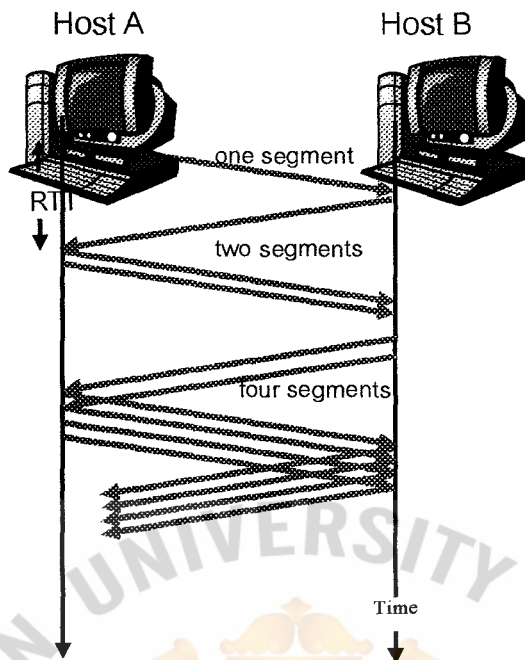


Figure 2-4 Slow Start

Congestion Avoidance

- Congestion avoidance and slow start are independent algorithms with different objectives. In practice they are implemented together.
- Two variables be maintained for each connection: a congestion window, *cwnd*, and a slow start threshold size, *ssthresh*.
- Initialization for a given connection sets *cwnd* to one segment and *ssthresh* to 65535 bytes.
- The TCP output routine never sends more than the minimum of *cwnd* and the receiver's advertised window.
- When congestion occurs, one half of the current window size is saved in *ssthresh*. Additionally, if the congestion is indicated by a timeout, *cwnd* is set to one segment.

Fast Retransmit

- TCP may generate an immediate acknowledgment (a duplicate ACK) when an out-of-order segment is received.
- The purpose of this duplicate ACK is to let the other end know that a segment was received out of order, and to tell it what sequence number is expected.
- Since TCP does not know whether a duplicate ACK is caused by a lost segment or just a reordering of segments, it waits for a small number of duplicate ACKs to be received.
- If there is just a reordering of the segments, there will be only one or two duplicate ACKs before the reordered segment is processed.
- If three or more duplicate ACKs are received in a row, it is a strong indication that a segment has been lost. TCP then performs a retransmission of what appears to be the missing segment without waiting for a retransmission timer to expire.

Fast Recovery

After fast retransmit sends what appears to be the missing segment, it uses congestion avoidance. The reason for not performing slow start is that the receipt of the duplicate ACKs tells TCP more than just a packet has been lost.

Since the receiver can only generate the duplicate ACK when another segment is received, that segment has left the network and is in the receiver's buffer.[1]

2.3 Simulation-based Comparisons of Tahoe, Reno and SACK TCP [4]

This literature uses simulation to explore the benefits of adding selective acknowledgements (SACK) to TCP. It compares Tahoe, Reno, New Reno and SACK TCP and show that SACK gives the best result.

Tahoe TCP

This follows a basic go-back-n model using slow-start, congestion avoidance and fast retransmit algorithms. With Fast Retransmit, after receiving a small number of duplicate acks for the same TCP segment, the data sender infers that the packet has been lost and retransmits the packet without waiting for the retransmission timer to expire.

Reno TCP

- Modification to the Tahoe TCP Fast Retransmit algorithm to include Fast Recovery; this prevents the pipe from going empty after Fast Retransmit, thereby avoiding the need to slow start after a single packet loss.
- A TCP sender enters Fast Recovery after receiving a threshold number of dup. ACKs. The sender retransmits one packet and reduces its congestion window by half. Instead of slow-starting, the Reno sender uses additional incoming dup. acks to clock subsequent outgoing packets.
- Reno TCP greatly improves performance in the face of single packet loss, but can suffer when multiple packets are lost.

New Reno TCP

In Reno, a partial ack (ack for some but not all of the packets that were outstanding at the start of the fast recovery period) takes TCP out of Fast Recovery. In New Reno, partial acks do not take TCP out of fast recovery; partial acks received during fast recovery are treated as an indication that the packet immediately following the ACK packet has been lost and should be retransmitted. Thus, when multiple packets are lost, New Reno can recover without a retransmission timeout.

SACK TCP

The TCP sender maintain a scoreboard which keeps track of acks from previous SACK packets. When the sender is allowed to send a packet, it retransmits the next packet from the list of packets inferred to be missing at the receiver. If there are no such packets and the receiver's advertised window is sufficiently large, the sender sends a new packet. RFC2018 [2].

Simulation Result

One packet loss: Tahoe TCP does badly due to slow-start after the packet loss. All other do relatively the same.

Two packet losses: Reno TCP fails to do as well as New Reno or SACK TCP, since its algorithm is tuned for single packet loss.

Multiple packet losses: Reno TCP performs miserably in the face of a large number of packet losses. SACK TCP continues to out-perform the rest of the algorithms.

2.4 Selective Acknowledgement Option [2]

TCP may experience poor performance when multiple packets are lost from one window of data. With the limited information available from cumulative acknowledgments, a TCP sender can only learn about a single lost packet per round trip time. An aggressive sender could choose to retransmit packets early, but such retransmitted segments may have already been successfully received.

A Selective Acknowledgment (SACK) mechanism, combined with a selective repeat retransmission policy, can help to overcome these limitations. The receiving TCP sends back SACK packets to the sender informing the sender of data that has been received. The sender can then retransmit only the missing data segments.

This memo proposes an implementation of SACK and discusses its performance and related issues

SACK allows the receiver to inform the sender about all segments that have been successfully received and allows the sender to retransmit only those segments that have been sent.

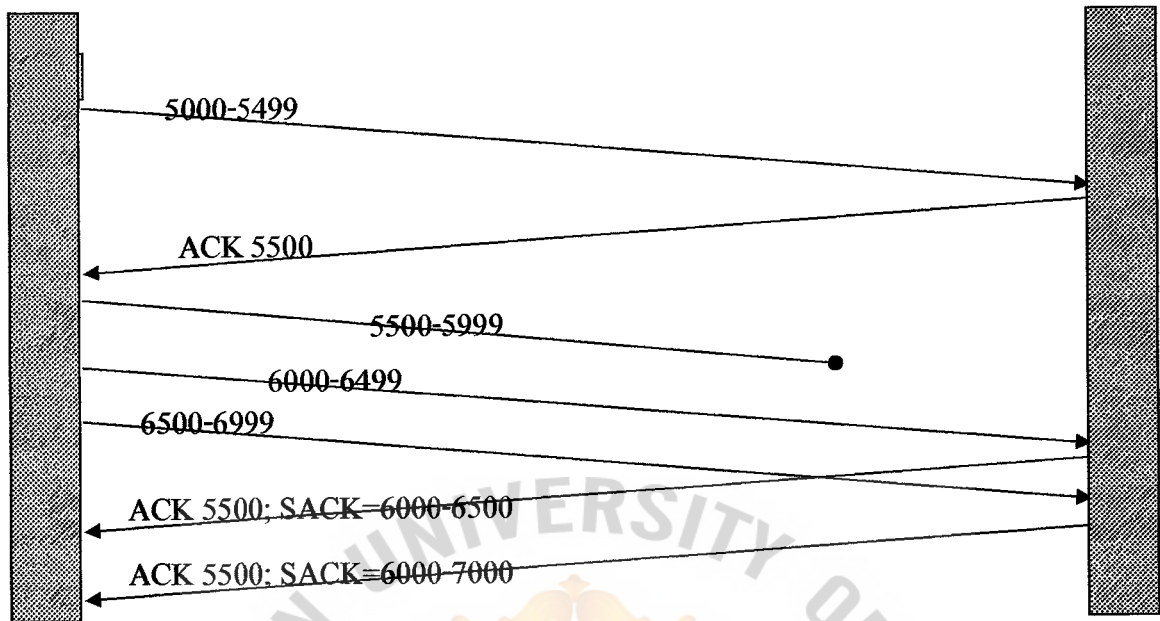


Figure 2-5 Selective Acknowledgment (SACK)

2.5 Comparative Study of RED, ECN and TCP Rate Control [7]

This literature evaluates the following network-based and end-to-end enhancements for addressing the issue of enhancing TCP performance.

- Random Early Detection (RED): an active queue management technique
- TCP-explicit congestion notification (ECN): which uses a one-bit explicit congestion notification instead of using packet drop as an implicit notification.
- Packeteer's TCP Rate Control: a network-based solution which controls the left and right edges of the TCP window, and shapes the TCP acknowledgement stream.

All schemes control bottleneck queuing delay, but trade off other measures such as drop rate, utilization and fairness, with TCP rate control exhibiting the best performance in terms of all metrics. In terms of deployment flexibility, TCP rate control and RED allow widespread and immediate deployment because they are transparent to hosts (ECN in not because it requires TCP protocol modifications). The minimal state requirements and protocol transparent of RED allows it a large deployment space.

2.6 TCP Rate Control [3]

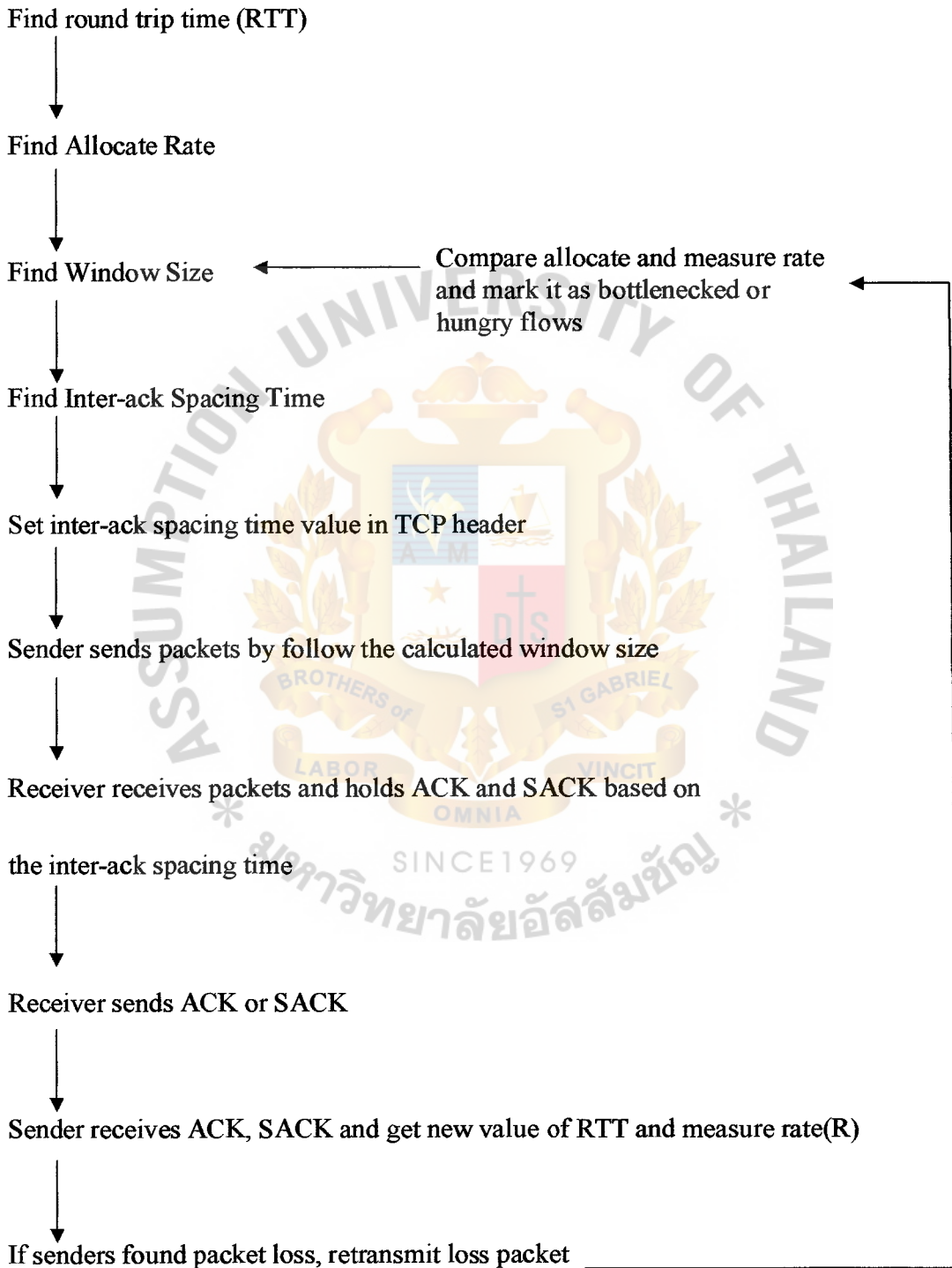
This paper presents TCP rate control, a new technique for transparently augmenting end-to-end TCP performance by controlling the sending rate of a TCP source. The sending rate of a TCP source is determined by its window size, the round trip time and the rate of acknowledgments. TCP rate control affects these aspects by modifying the ack number and receiver window fields in acknowledgments and by modulating the acknowledgment rate. From a performance viewpoint, a key benefit of TCP rate control is to avoid adverse performance effects due to packet losses such as reduced goodput and unfairness or large spread in per-user goodputs. Further, TCP rate control positively affects performance even if the bottleneck is non-local and the end-host TCP implementations are nonconforming.

These aspects are demonstrated through a comparative study of TCP rate control, RED and TCP-ECN. The TCP rate control approach has been implemented and patented by Packeteer Inc.

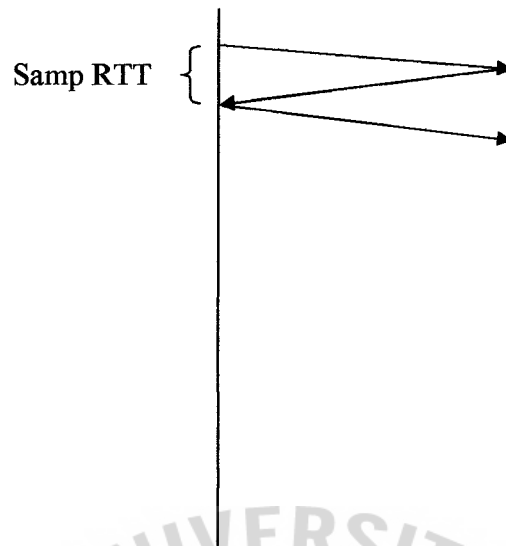
3. PROPOSED SYSTEM

3.1 Proposed Algorithm

TCP Rate Control Methodology Flow Chart



1.



Find RTT

Samp RTT = Sample value of RTT by noting the time difference between creation of the TCP segment and receipt of an ACK for it.

Est RTT = the estimate of RTT that TCP has after measuring.

$$Est\ RTT\ (n) = \alpha \times Est\ RTT\ (n-1) + (1-\alpha) \times Samp\ RTT\ (n) \quad ;(\alpha = 7/8)$$

2. Find Allocate Rate (A_i)

B = Bottlenecked capacity

N = Number of flows

$$A_i = B/N$$

3. Compare with Measure Rate (R_i)

If $R_i < A_i$, then mark flow i as *bottlenecked* else mark it as *hungry*

4. Calculate the new allocate rate

Hungry Flows

U = Aggregate residual bandwidth the bandwidth that remains unutilized by the bottlenecked flows. This value can be calculated by bottleneck capacity (B) minus measure rate (R)

H = Total Number of Hungry Flows This is the total number of flows in the link.

$$A_j = A_j + U/H$$

Bottlenecked Flows

$$A_k = (A_k + R_k) / 2$$

5. Find Window Size

W = the calculated window size in unit of packets

RTT = the round trip time, in seconds

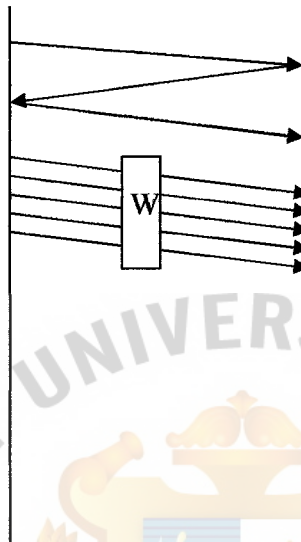
MSS = the maximum segment size, in bytes

A = the rate allocation in bytes/s

$$W = (A \times RTT) / MSS$$

Where $0 < W < 300$

6. Send the packets based on calculated window size (W)

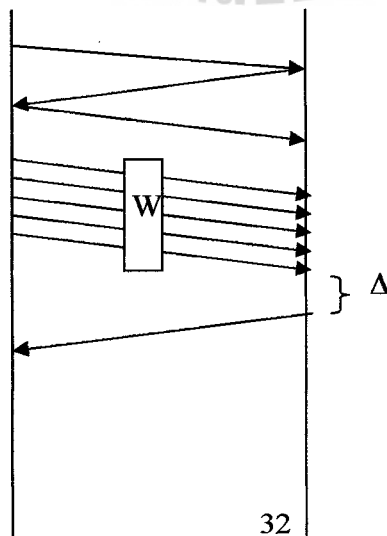


6. Find Inter-ack Spacing Time

Δ = the inter-ack spacing time, in seconds

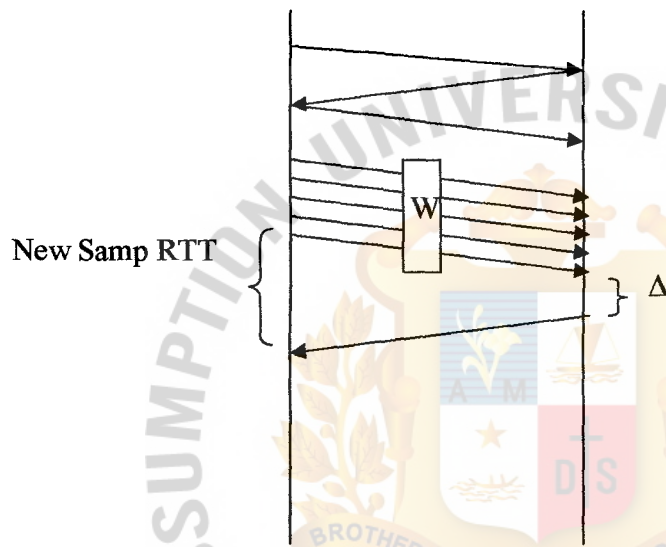
$$\Delta = RTT / W$$

7. Delay ACK and SACK based on inter-ack spacing time



8. If there is packet loss, retransmit loss packet

9. Get the new value of RTT and measure rate (R_i)



Measure rate (R_i) = size of flow (byte) / RTT (seconds)

10. Repeat to step 1 until all data is sent

Note: For the first flow, skip the step 3 and 4. Use the allocated rate (A_i), there is no need to mark the flow as hungry or bottlenecked flow

3.2 Scope and Limitation

The scope and limitation for this thesis will follow as:

1. TCP protocol will follow RFC 793 [5].
2. TCP SACK will follow RFC2018 [3].
3. TCP algorithms will follow RFC2001 [2].
4. This thesis will be working in the same LAN switch network.
5. The simulated scenario will be as following scenario

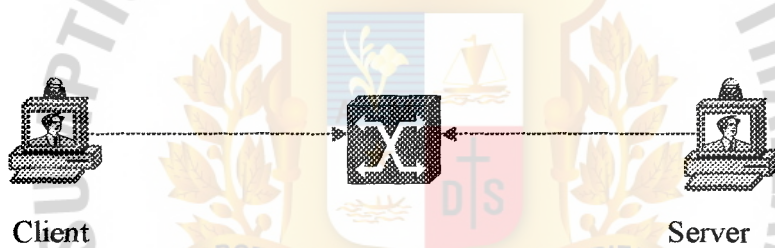


Figure 3-1: Single Hop Topology



Figure 3-2: Multi Hop Topology

Client side opens TCP ports to transmit data to server and the rate of TCP will be controlled based on the congestion by following the proposed algorithms.

3.3 Methodology

Network Simulator version 2 (NS2) or other coding C language is used to simulate the idea for this thesis. Network Simulator version2 is based on C++ and Otcl and developed by University of California at Berkeley (UCB).

This program uses metrics as follow:

- Sent packets for TCP and UDP (Bytes)
- Received packets for TCP and UDP (Bytes)
- Dropped packets for TCP and UDP (Bytes)

This thesis simulated 2 cases:

1. TCP with rte control by follow the proposed algorithm.
2. Standard TCP.

4. RESULTS

This chapter describes the performance results of TCP Rate Control and standard TCP. The simulation is separated into 2 categories, single hop topology and multi hop topology, with 14 cases. Section 4.6 presented the comparative results of TCP Rate Control and Standard TCP. A brief discussion of topologies and parameters are presented as follows:

4.1 Tested Topologies and Parameters for Single Hop and Multi Hop Topology

Topologies are classified into 2 categories: *single hop* and *mulie hop* as described below:

Single Hop Topology: All TCP, UDP sources and servers are connected directly to the switch. To get the various results, there is the adjustment of the number of TCP and UDP sources and the bandwidth of the link from switch to the server. The bandwidth from the switch to the server is adjusted from fast Ethernet (100 Mbps) to standard Ethernet (10 Mbps) and to low bandwidth (1 Mbps). The use of low bandwidth from the switch to server is unrealistic. However, it is easy to evaluate the algorithm and cleary show the result from the simulation. Figure 4.1-4.3 presented the diagram of Single Hop Topology. There are 8 study cases (case study 1-8) for the simulation of single hop topology. The detail of the number of TCP and UDP sources for each case is described in section 4.2-4.3. All TCP and UDP sources send the packet to the server by following the tested parameters.

Multi Hop Topology: All TCP, UDP sources and servers are connected directly to the switch and the server is connected to the another switch. These two switches are connected together. To get the various results, there is the adjustment of the number

of TCP and UDP sources and the bandwidth of the link from switch to the server. The bandwidth from the switch to the server is adjusted from standard Ethernet (10 Mbps) to low bandwidth 5 Mbps and to 1 Mbps. The use of low bandwidth between the switches is unrealistic. However, it is easy to evaluate the algorithm and clearly show the result from the simulation. Figure 4.4 -4.6 presented the diagram of Multi Hop Topology. There are 6 study cases (case study 9-14) for the simulation of multi hop topology. The detail of the number of TCP and UDP sources for each case is described in section 4.4-4.5. All TCP sources send the packets to one server and UDP sources send the packets to another server.

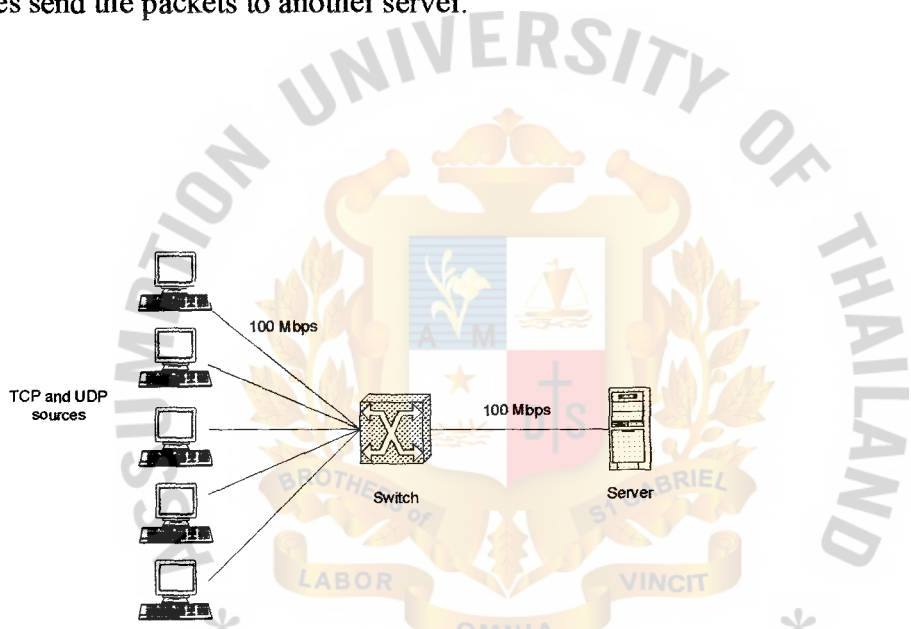


Figure 4-1: Topology A – The switch is connected to the server at 100 Mbps

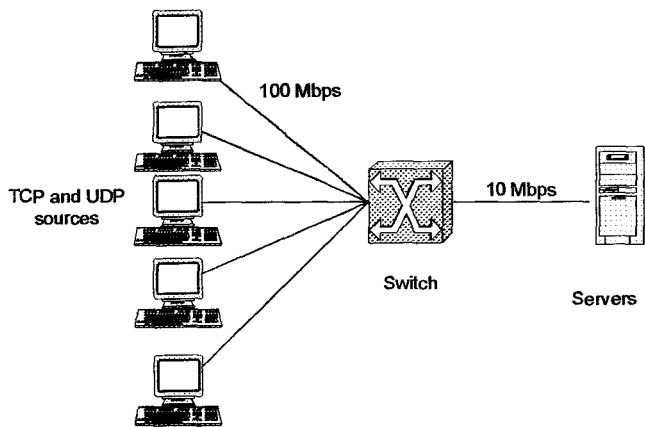


Figure 4-2: Topology B - The switch is connected to the server at 10 Mbps

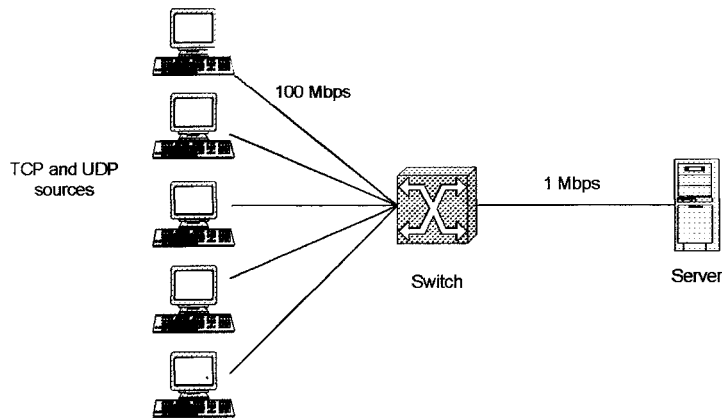


Figure 4-3: Topology C - The switch is connected to the server at 1 Mbps

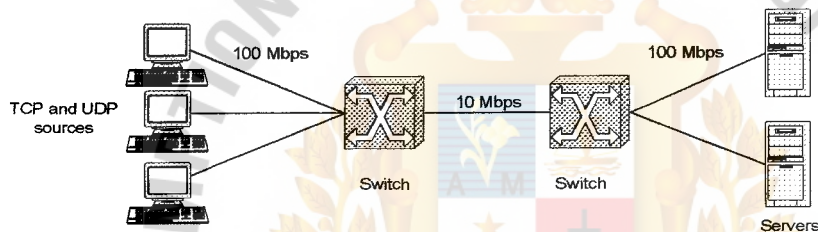


Figure 4-4: Topology D - The bandwidth between switches is 10 Mbps

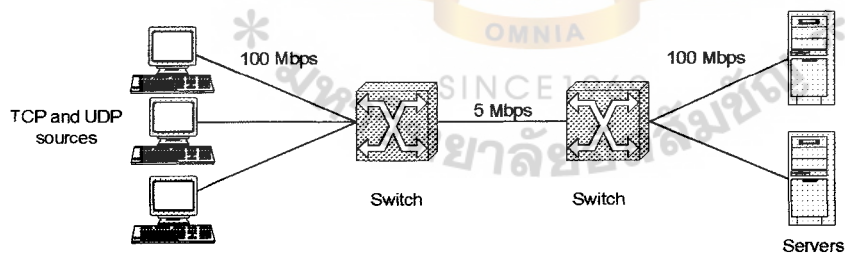


Figure 4-5: Topology E - The bandwidth between switches is 5 Mbps

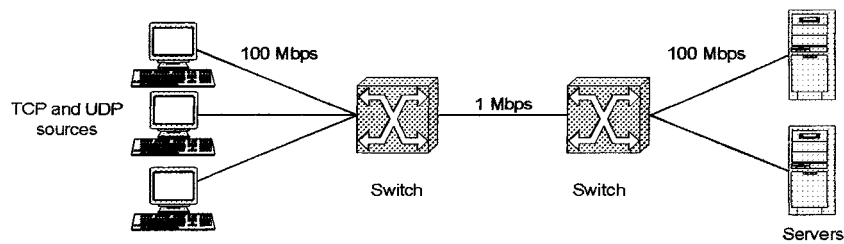


Figure 4-6: Topology F - The bandwidth between switches is 1 Mbps

The evaluation follows the following parameters:

- TCP selective acknowledgment (SACK),
- Tail Drop queue type
- 1000 bytes of packet size
- Duration of 180 milliseconds
- 448 Kbps of UDP incoming traffic (Constant Bit Rate, CBR)



4.2 Evaluation of Single Hop Topology for Fast Ethernet (100BaseT)

Refer to Figure 4.1, topology A, all TCP and UDP sources are connected to the switch at the bandwidth 100 Mbps and the switch is connected to the server at the bandwidth 100 Mbps. The evaluation of topology A is separated into 3 cases with different number of TCP and UDP sources. The adjustment of the number of TCP and UDP sources for topology A is presented as follows:

Table 4-1: Number of TCP and UDP sources for Case 1-3

Case	Number of TCP Sources	Number of UDP Sources
1	5	0
2	5	10
3	2	15

Table 4-2: Percentage of TCP and UDP sources and comparison percentage of TCP and UDP received packet for Case 1-3

Case	% of TCP Sources	% of UDP Sources	Standard TCP		TCP Rate Control	
			% of TCP Received	% of UDP Received	% of TCP Received	% of UDP Received
1	100	0	100	0	100	0
2	33	67	81.01	18.99	80.95	19.05
3	12	88	48.55	51.45	48.95	51.05

From the above configuration of the number of TCP and UDP sources, the measurement values are the number of sent, received and dropped packets for TCP and UDP packets. The results are presented as follow:

Table 4-3: Result of standard TCP for case 1-3

Case	TCP sent (bytes)	TCP Received (bytes)	UDP sent (bytes)	UDP Received (bytes)	TCP Dropped (bytes)
1	7,155,000	7,151,000	-	-	-
2	7,155,000	7,123,000	1,680,000	1,670,000	-
3	2,862,000	2,862,000	3,050,000	3,033,000	-

Table 4-4: Result of TCP Rate Control for case 1-3

Case	TCP sent (bytes)	TCP Received (bytes)	UDP sent (bytes)	UDP Received (bytes)	TCP Dropped (bytes)
1	7,105,000	7,105,000	-	-	-
2	7,123,000	7,096,000	1,680,000	1,670,000	-
3	2,786,000	2,786,000	2,920,000	2,905,000	-

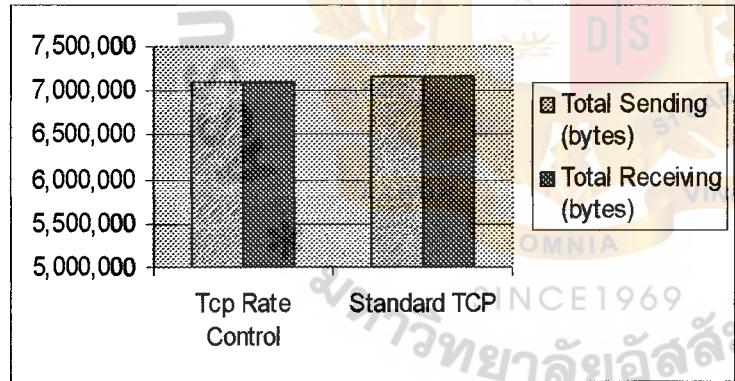


Figure 4.7: Comparison of total sending and receiving packets for standard TCP and TCP Rate Control of case 1

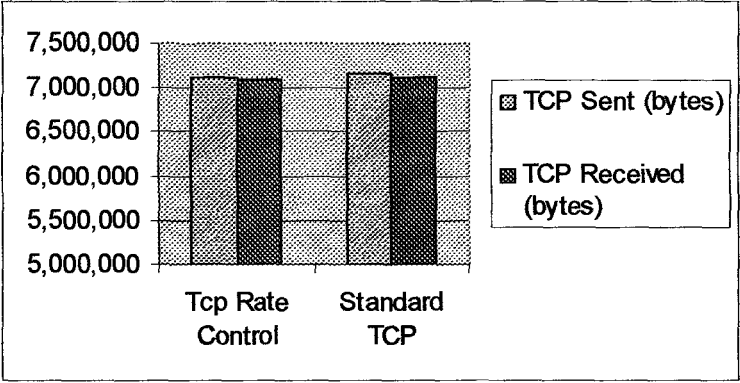


Figure 4-8: Comparison of total sending and receiving packets for standard TCP and TCP Rate Control of case 2

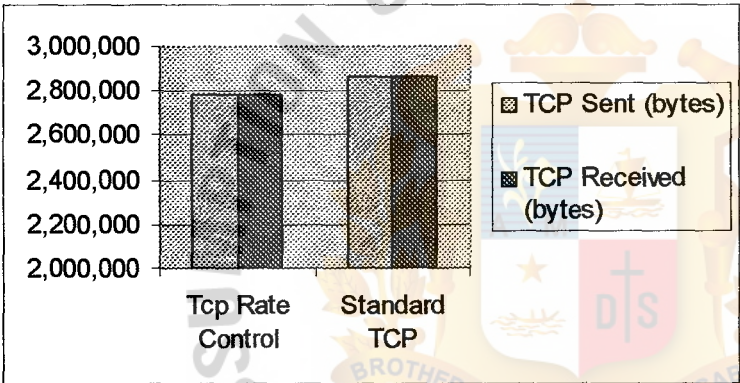


Figure 4-9: Comparison of total sending and receiving packets for standard TCP and TCP Rate Control of case 3

Figure 4.7 – Figure 4.9 show the comparison of standard TCP and TCP rate control by comparing the total number of TCP and UDP sent and received packets. From these results, it shows that the total of sent and received TCP and UDP packets of both algorithms is not much different. The different is less than 3 % for the total receiving TCP packets for case study 1-3. Due to this topology, the bandwidth from switch to server is Fast Ethernet, 100 Mbps, so the link is not congested. With noncongested link, the performance for sent and received packets for standard TCP

and TCP rate control is almost the same. Reducing the bandwidth to make congested link can get the different performances for both topologies. Next section describes the effect of reducing the bandwidth of the link for single hop topology to standard Ethernet 10BaseT and low bandwidth.



4.3 Effect of Reducing the Bandwidth for Single Hop Topology

This section describes the effect of reducing the bandwidth from the switch to server in single hop topology. Refer to figure 4.2 and 4.3 shown the topology B and C that reduce the bandwidth to 10 Mbps and 1 Mbps.

4.3.1 Evaluation of Reducing the Bandwidth to Standard Ethernet 10BaseT

Evaluation of reducing the bandwidth from the switch to server to standard Ethernet 10BaseT, 10 Mbps, is separated into 3 study cases (case 4 – 6). The adjustments of the number of TCP and UDP sources are presented as follow:

Table 4-5: Number of TCP and UDP sources for Case 4-6

Case	Number of TCP Sources	Number of UDP Sources
4	5	0
5	5	5
6	5	15

Table 4-6: Percentage of TCP and UDP sources and comparison percentage of TCP and UDP received packet for Case 4-6

Case	% of TCP Sources	% of UDP Sources	Standard TCP		TCP Rate Control	
			% of TCP Received	% of UDP Received	% of TCP Received	% of UDP Received
4	100	0	100	0	100	0
5	50	50	78.18	21.82	77.74	21.26
6	33	67	39.18	60.82	38.6	61.4

From the above configuration of the number of TCP and UDP sources, the measurement values are the number of sent, received and dropped packets for TCP and UDP packets. The results are presented as follow:

Table 4-7: Result of standard TCP for case 4-6

Case	TCP sent (bytes)	TCP Received (bytes)	UDP sent (bytes)	UDP Received (bytes)	TCP Dropped (bytes)
4	3,606,000	3,592,000	-	-	-
5	2,853,000	2,848,000	800,000	795,000	27,000
6	1,698,000	1,691,000	2,635,000	2,625,000	51,000

Table 4-8: Result of TCP Rate Control for case 4-6

TCP sent (bytes)	TCP Received (bytes)	UDP sent (bytes)	UDP Received (bytes)	TCP Dropped (bytes)
3,606,000	3,592,000	-	-	-
2,835,000	2,829,000	818,000	810,000	20,000
1,687,000	1,683,000	2,680,000	2,677,000	44,000

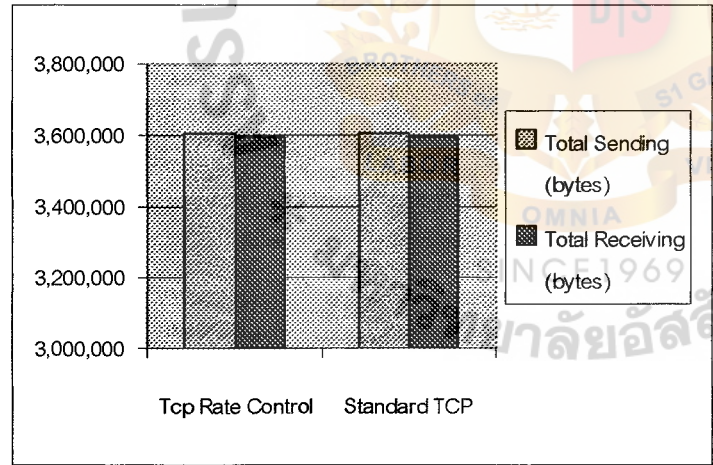


Figure 4-10: Comparison of total sending and receiving packets for standard TCP and TCP Rate Control of case 4

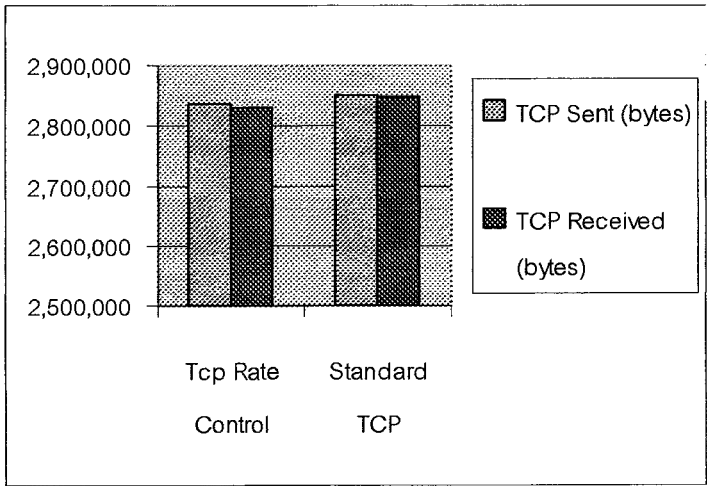


Figure 4-11: Comparison of total sending and receiving packets for standard TCP and TCP Rate Control of case 5

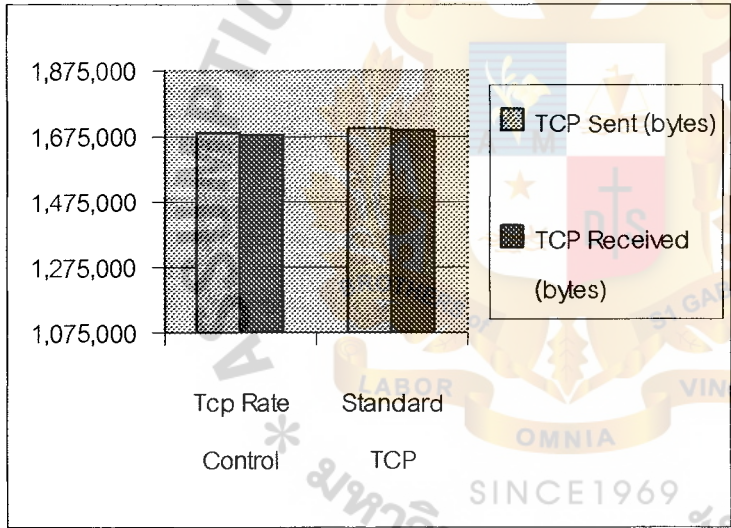


Figure 4.12: Comparison of total sending and receiving packets for standard TCP and TCP Rate Control of case 6

From the results in Figure 4.10 – 4.12, the TCP sent and received packets of standard TCP and TCP rate control are not much different. The difference between both algorithm is less than 1% for the total receiving TCP packets. The UDP sent and received packets of standard TCP and TCP rate control are also not much different. There is no effect of reducing the bandwidth to 10 Mbps because the link is not congested. Section 4.3.2 describes the evaluation of reducing the bandwidth to low bandwidth.

4.3.2 Evaluation of Reducing the Bandwidth to Low Bandwidth

The adjustment of the bandwidth from switch to server to low bandwidth, 1 Mbps, makes the link more congested. Evaluation of reducing the bandwidth from the switch to server to 1 Mbps is separated into 2 study cases (case 7 - 8). The adjustments of the number of TCP and UDP sources are presented as follow:

Table 4-9: Number of TCP and UDP sources for Case 7-8

Case	Number of TCP Sources	Number of UDP Sources	% of TCP	% of UDP
7	10	2	55	45
8	5	2	50	50

Table 4-10: Percentage of TCP and UDP sources and comparison percentage of TCP and UDP received packet for Case 7-8

Case	% of TCP Sources	% of UDP Sources	Standard TCP		TCP Rate Control	
			% of TCP Received	% of UDP Received	% of TCP Received	% of UDP Received
7	83	17	55.76	44.24	60.21	39.79
8	71	29	50.66	49.34	56.5	43.5

From the above configuration of the number of TCP and UDP sources, the measurement values are the number of sent, received and dropped packets for TCP and UDP packets. The results are presented as follow:

Table 4-11: Result of standard TCP for case 7-8

Case	TCP sent (bytes)	TCP Received (bytes)	UDP sent (bytes)	UDP Received (bytes)	TCP Dropped (bytes)
7	214,000	213,000	170,000	169,000	88,000
8	192,000	191,000	187,000	186,000	82,000

Table 4-12: Result of TCP Rate Control for case 7-8

Case	TCP sent (bytes)	TCP Received (bytes)	UDP sent (bytes)	UDP Received (bytes)	TCP Dropped (bytes)
7	232,000	230,000	152,000	152,000	96,000
8	214,000	213,000	165,000	164,000	128,000

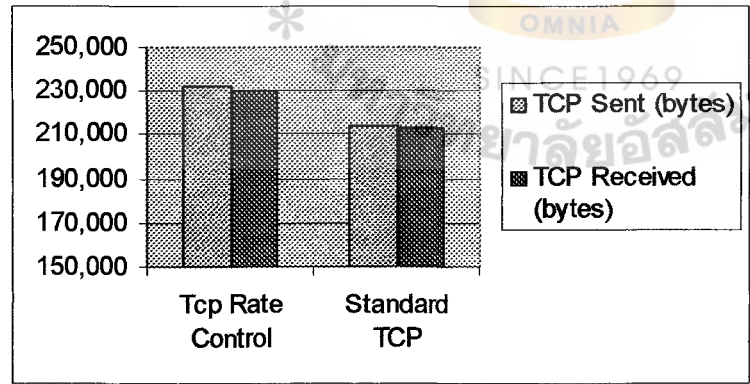


Figure 4-13: Comparison of total sending and receiving packets for standard TCP and TCP Rate Control of case 7

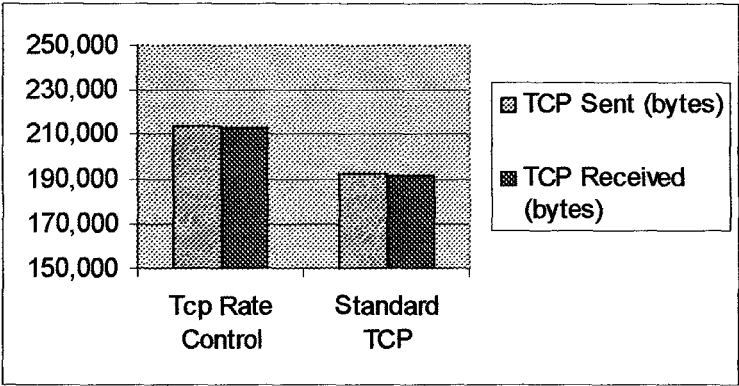


Figure 4-14: Comparison of total sending and receiving packets for standard TCP and TCP Rate Control of case 8

From the results in Table 4.8 – 4.9 and Figure 4.13 – 4.14, it shows that using TCP rate control algorithm is better than standard TCP in team of sending more TCP packets. The difference of the total receiving TCP packets between both algorithms is about 7% for case study 7 and about 10% for case study 8. In the opposite way in these 2 cases, using TCP rate control can send less UDP packets than standard TCP. From the result, it can be concluded that TCP rate control algorithm can sent more TCP packets than standard TCP in a low bandwidth environment.

4.3.3 Discussion of Effect of Reducing the Bandwidth for Single Hop Topology

Section 4.3.1 presents the evaluation of reducing the bandwidth to standard Ethernet,10 Mbps. From the results, there is not much difference in the performance between standard TCP and TCP rate control in terms of the total receiving TCP and UDP packets. The difference is only less than 1% for the total receiving TCP packets. Reducing the bandwidth to low bandwidth,1 Mpbs makes the link more congested. Section 4.3.2 presents the evaluation of reducing the bandwidth to 1 Mpbs. The results show TCP rate control sent more TCP packets than standard TCP. The difference of the total receiving TCP packets between both algorithms is about 7% for case study 7 and about 10% for case study 8. In the opposite way, using TCP rate

control sent less UDP packets than standard TCP in case study 7 and 8 or in a congested environment.

In summary, reducing the bandwidth to 10 Mbps as in section 4.3.1, the link is not very congested, so the performance of using TCP rate control and standard TCP are not much different. However, after reducing the bandwidth to 1 Mbps makes the link very congested, using TCP rate control has a higher performance than standard TCP. Thus, it can be concluded that TCP rate control has a higher performance than standard TCP for very congested link environment in single hop topology. Next section describes the evaluation of multi hop topology.



4.4 Evaluation of Multi Hop Topology for Standard Ethernet 10BaseT

Refer to Figure 4.10, topology D, all TCP and UDP sources are connected to the switch at the bandwidth 100 Mbps. The switches connect together at the bandwidth of standard Ethernet,10 Mbps, and the switch connects to the server at the bandwidth 100 Mbps. The evaluation of topology D is separated into 2 cases with different number of TCP and UDP sources. The adjustment of the number of TCP and UDP sources for topology A is presented as follows:

Table 4-13: Number of TCP and UDP sources for Case 9-10

Case	Number of TCP Sources	Number of UDP Sources
9	5	10
10	5	5

Table 4-14: Percentage of TCP and UDP sources and comparison percentage of TCP and UDP received packet for Case 9-10

Case	% of TCP Sources	% of UDP Sources	Standard TCP		TCP Rate Control	
			% of TCP Received	% of UDP Received	% of TCP Received	% of UDP Received
9	33	67	57.15	42.85	55.33	44.67
10	50	50	77	23	76.66	23.34

From the above configuration of the number of TCP and UDP sources, the measurement values are the number of packets sent, received and dropped packets for TCP and UDP packets. The results are presented as follow:

Table 4-15: Result of standard TCP for case 9-10

Case	TCP sent (bytes)	TCP Received (bytes)	UDP sent (bytes)	UDP Received (bytes)	TCP Dropped (bytes)
9	2,083,000	2,067,000	1,560,000	1,550,000	20,000
10	2,756,000	2,735,000	822,000	817,000	8,000

Table 4-16: Result of TCP Rate Control for case 9-10

Case	TCP sent (bytes)	TCP Received (bytes)	UDP sent (bytes)	UDP Received (bytes)	TCP Dropped (bytes)
9	2,013,000	1,998,000	1,630,000	1,613,000	17,000
10	2,743,000	2,723,000	835,000	829,000	6,000

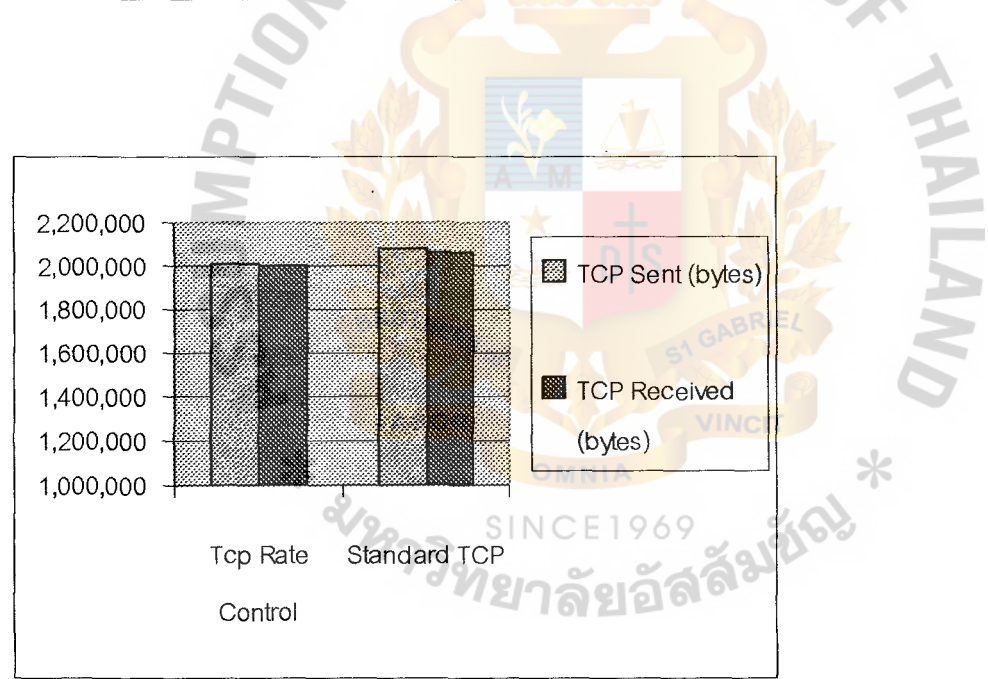


Figure 4-15: Comparison of total sent and received packets for standard TCP and TCP Rate Control of case 9

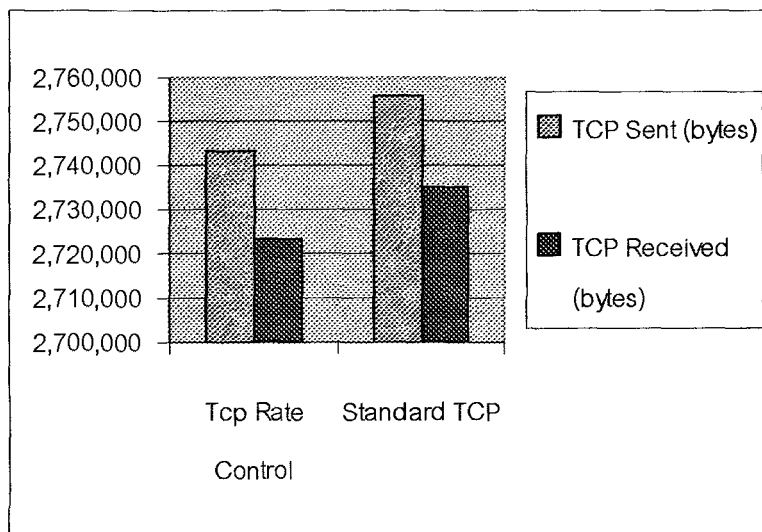


Figure 4-16: Comparison of total sent and received packets for standard TCP and TCP Rate Control of case 10

From the result in Figure 4.15 – 4.16, it is clear that using standard TCP algorithm can sent more TCP packets than using TCP rate control algorithm. For this multi hop topology, the bandwidth from switch to server is standard Ethernet 10BaseT, 10 Mbps. Standard TCP has a higher performance in terms of the total receiving TCP packets than TCP rate control. However, the difference for total receiving TCP packets for both algorithms is less than 5%. Refer to the results about receiving UDP packets; the difference between TCP rate control and standard TCP is not much different. Reducing the bandwidth of the link between the switches making the congested link is evaluated in the next section.

4.5 Effect of Reducing the Bandwidth for Multi Hop Topology

This section describes the effect of reducing the bandwidth between the switches for multi hop topology. Figures 4.5 and 4.6 show the topology E and F that reduce the bandwidth between switches to low bandwidth to make a congested link.

4.5.1 Evaluation of Reducing the Bandwidth to Low Bandwidth

Reducing the Bandwidth to 5 Mbps

Evaluation of reducing the bandwidth between the switches to 5 Mbps is separated into 2 cases (case 11 – 12). The adjustments of the number of TCP and UDP sources are presented as follow:

Table 4-17: Number of TCP and UDP sources for Case 11-12

Case	Number of TCP Sources	Number of UDP Sources
11	5	10
12	5	5

Table 4-18: Percentage of TCP and UDP sources and comparison percentage of TCP and UDP received packet for Case 11-12

			Standard TCP		TCP Rate Control	
Case	% of TCP Sources	% of UDP Sources	% of TCP Received	% of UDP Received	% of TCP Received	% of UDP Received
11	33	67	17.99	82.01	21.11	78.99
12	50	50	57.96	42.04	56.65	43.35

From the above configuration of the number of TCP and UDP sources, the measurement values are the number of sent, received and dropped packets for TCP and UDP packets. The results are presented as follow:

Table 4-19: Result of standard TCP for case 11-12

Case	TCP sent (bytes)	TCP Received (bytes)	UDP sent (bytes)	UDP Received (bytes)	TCP Dropped (bytes)
11	337,000	334,000	1,534,000	1,523,000	56,000
12	1,065,000	1,059,000	775,000	768,000	53,000

Table 4-20: Result of TCP Rate Control for case 11-12

Case	TCP sent (bytes)	TCP Received (bytes)	UDP sent (bytes)	UDP Received (bytes)	TCP Dropped (bytes)
11	393,000	392,000	1,478,000	1,465,000	74,000
12	1,043,000	1,035,000	797,000	792,000	46,000

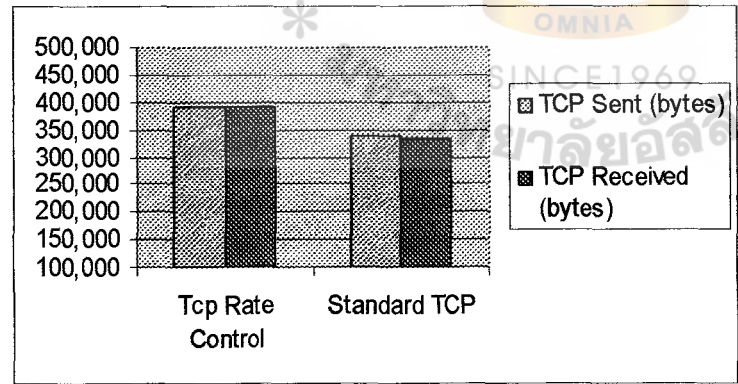


Figure 4-17 : Comparison of total sent and received packets for standard TCP and TCP Rate Control of case 11

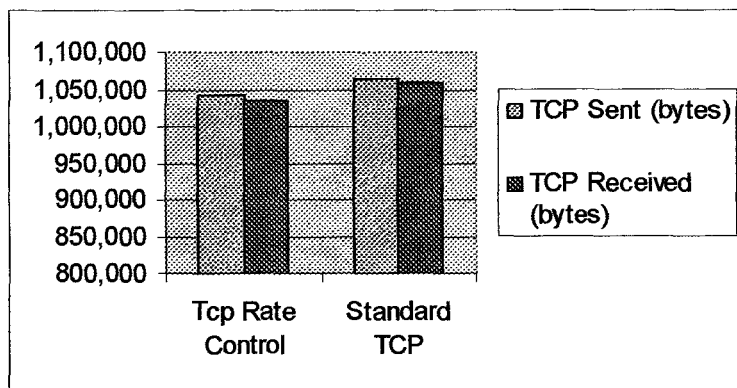


Figure 4-18: : Comparison of total sending and receiving packets for standard TCP and TCP Rate Control of case 12

From the result in Figure 4.17 in case 11, it shows that using TCP rate control algorithm can send more TCP packets than standard TCP. The different for both algorithm is about 14% for the total receiving TCP packets. However in Figure 4.18 in case 12, it is shown that standard TCP can send more TCP packets than TCP rate control. The different for both algorithm is about 2% for the total receiving TCP packets.

TCP rate control has higher performance than standard TCP in case 11. In case 11 the UDP packets are set to 80% of the total and in case 12 the UDP packets are set to 40% of the total bandwidth so the performance of TCP rate control is lower than standard TCP. TCP rate control has a higher performance when competing with a lot of UDP packets. Next section describes the evaluation of reducing the bandwidth to 1 Mbps to make the link more congested and compare the result using TCP rate control and standard TCP.

Reducing the Bandwidth to 1 Mbps

The adjustment of the bandwidth between switches to 1 Mbps to make the link more congested. Evaluation of reducing the bandwidth between switches to 1 Mbps is separated into 2 cases (case 13 - 14). The adjustment of the number of TCP and UDP sources are presented as follow:

Table 4-21: Number of TCP and UDP sources for Case 13-14

Case	Number of TCP Sources	Number of UDP Sources	% of TCP	% of UDP
13	5	10	7	93
14	5	5	10	90

Table 4-22: Percentage of TCP and UDP sources and comparison percentage of TCP and UDP received packet for Case 13-14

Case	% of TCP Sources	% of UDP Sources	Standard TCP		TCP Rate Control	
			% of TCP Received	% of UDP Received	% of TCP Received	% of UDP Received
13	33	67	6.93	93.07	10.13	89.87
14	50	50	8.51	91.49	12.23	87.77

From the above configuration of the number of TCP and UDP sources, the measurement values are the number of sent, received and dropped packets for TCP and UDP packets. The results are presented as follow:

Table 4-23: Result of standard TCP for case 13-14

Case	TCP sent (bytes)	TCP Received (bytes)	UDP sent (bytes)	UDP Received (bytes)	TCP Dropped (bytes)
13	26,000	26,000	353,000	349,000	30,000
14	33,000	32,000	346,000	344,000	28,000

Table 4-24: Result of TCP Rate Control for case 13-14

Case	TCP sent (bytes)	TCP Received (bytes)	UDP sent (bytes)	UDP Received (bytes)	TCP Dropped (bytes)
13	38,000	38,000	341,000	337,000	42,000
14	47,000	46,000	332,000	330,000	46,000

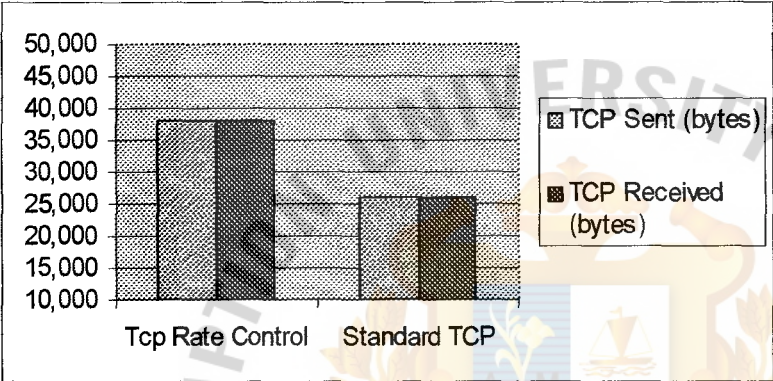


Figure 4-19: Comparison of total sent and received packets for standard TCP and TCP Rate Control of case 13

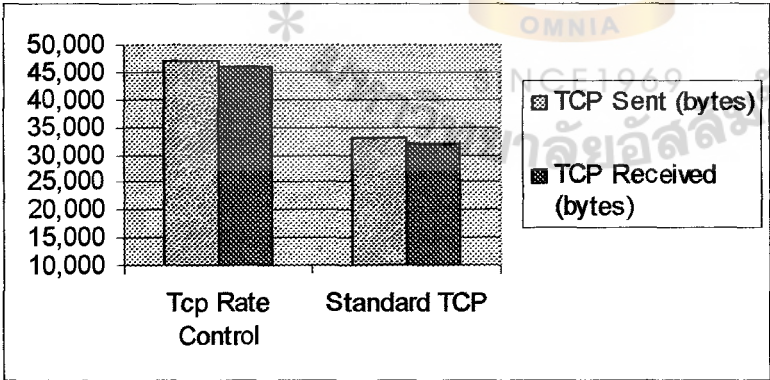


Figure 4-20: Comparison of total sent and received packets for standard TCP and TCP Rate Control of case 14

The above result shows that using TCP rate control can send more TCP packets than standard TCP for both cases. After reducing bandwidth to 1 Mbps to make the link very congested, TCP rate control algorithm has a very high performance than standard TCP. The difference between both algorithms is about 30% for the receiving TCP packets. When using TCP rate control algorithm, the received UDP packets is less than by using standard TCP.

4.5.2 Discussion of Effect of Reducing the Bandwidth for Multi Hop Topology

Section 4.5.1 presents the evaluation of reducing the bandwidth to 5 Mbps. From the results, TCP rate control sent more TCP packets than standard TCP in case 11 but sent less TCP packets in case 12. In case 11 the UDP packets is set to 80% of the total and in case 12 the UDP packets is set to 40% of the bandwidth. TCP rate control has a higher performance when competing with a lot of UDP packets. The difference for both algorithms is about 2% for the total receiving TCP packets. Section 4.5.2 represents the evaluation of reducing the bandwidth to 1 Mbps. After reducing the bandwidth to 1 Mbps make the link very congested, the results show TCP rate control sent more TCP packets than standard TCP in both cases. The difference between both algorithms is about 30% for the receiving TCP packets.

In summary, reducing the bandwidth to 5 Mbps as in section 4.5.1 the link is not very congested so the performance of using TCP rate control and standard TCP depend on the number of TCP and UDP packets. TCP rate control has a higher performance when competing with a lot of UDP packets. But after reducing more bandwidth to 1 Mbps to make the link very congestion, using TCP rate control has a higher performance than standard TCP. Thus, it can be concluded that TCP rate control has a higher performance than standard TCP for a very congested link environment in Multi hop topology. However, when using TCP rate control algorithm, the receiving

UDP packets is less than using standard TCP in a congested environment. This is the limitation of TCP rate control algorithm for multi hop topology.



4.6 Comparative Results of TCP Rate Control and Standard

This section will discuss about the comparison of using TCP rate control and standard TCP algorithms that refer to the simulated results in section 4.2-4.7.

4.6.1 Discussion of Single Hop Topology

Refer to section 4.2-4.4, it used the same topology with the different bandwidth between switch and server. It started from 100 Mbps to 10 Mbps and 1 Mbps to make congested link. In summary, Table 4.19 shows the comparison of TCP packet received between TCP rate control and standard TCP algorithm for case 1 – 8.

Table 4-25: TCP received packet comparison for case 1-8

Standard TCP		TCP Rate Control	
Case	TCP Received (bytes)	TCP Received (bytes)	% Coefficient Differentiation
1	7,151,000	7,105,000	-0.64743139
2	7,123,000	7,096,000	-0.38049605
3	2,862,000	2,786,000	-2.72792534
4	3,592,000	3,592,000	0
5	2,848,000	2,829,000	-0.67161541
6	1,691,000	1,683,000	-0.47534165
7	213,000	230,000	7.391304348
8	191,000	213,000	10.3286385

From the above comparison table, it is shown that case 7 -8 TCP rate control can receive more TCP packets than standard TCP. I conclude that TCP rate control can receive more TCP packets than standard TCP in a congested environment.

From Table 4.19, the value of % coefficient differentiation shows the percentage difference between TCP rate control and standard TCP. If the value is positive, it means TCP rate control received more TCP packet than standard TCP. And in the opposite way, if the value of % coefficient different is negative, TCP rate control sent less TCP packets than standard TCP.

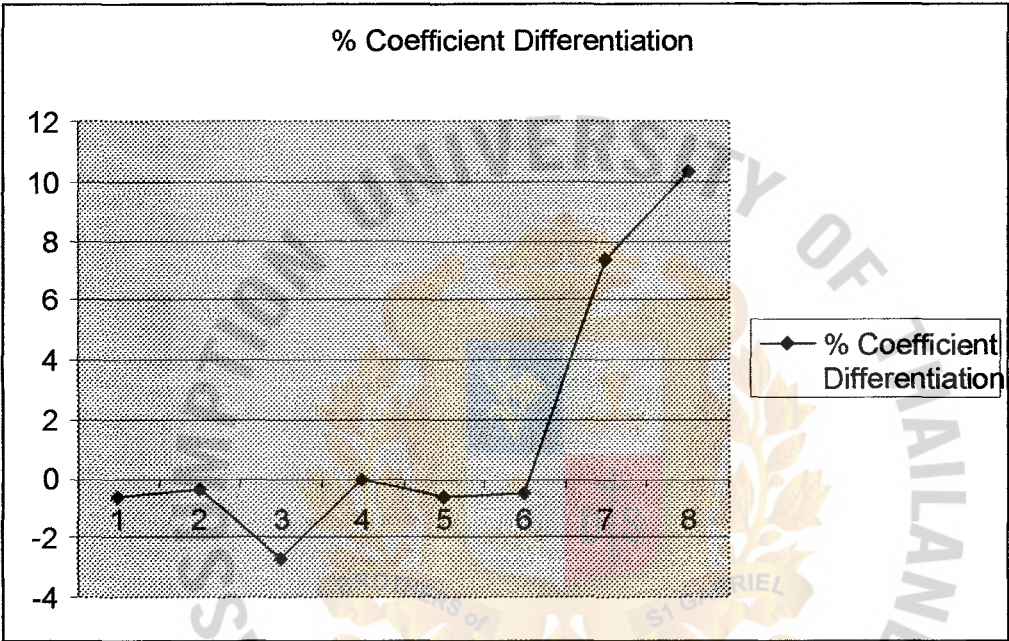


Figure 4-21: The percentage coefficient differentiation between TCP rate control and standard TCP for case study 9-14

The above figure presents the graph of the percentage of differentiation between TCP rate control and standard TCP. Refer to the graph TCP rate control working well in case 7 – 8, this shows that TCP rate control can work very good in congestion environment. From this result, the maximum different is 10.32% from this study.

4.6.2 Discussion of Multi Hop Topology

Refer to the section 4.4 – 4.7, I used two switches which one switch connected to TCP and UDP sources and another switch connected to server. I started the bandwidth of between two switches from 10 Mbps to 5 Mbps to 1 Mbps to make the link congested. The table below shown the summary result of TCP received packet of using TCP rate control and standard TCP and percentage of coefficient different.

Table 4-26: TCP received packet comparison for case 9 – 14

	Standard TCP	TCP Rate Control	% Coefficient Differentiation
Case	TCP Received	TCP Received	
9	2,067,000	1,998,000	-3.453453453
10	2,735,000	2,723,000	-0.440690415
11	334,000	392,000	14.79591837
12	1,059,000	1,035,000	-2.31884058
13	26,000	38,000	31.57894737
14	32,000	46,000	30.43478261

From the above comparison table, it is clear that case 13 - 14 TCP rate control can receive much more TCP packets than standard TCP. The result is similar to case 1-8, TCP rate control can send more TCP packet in congestion environment. From this result, the maximum difference is 31.57% from this study.

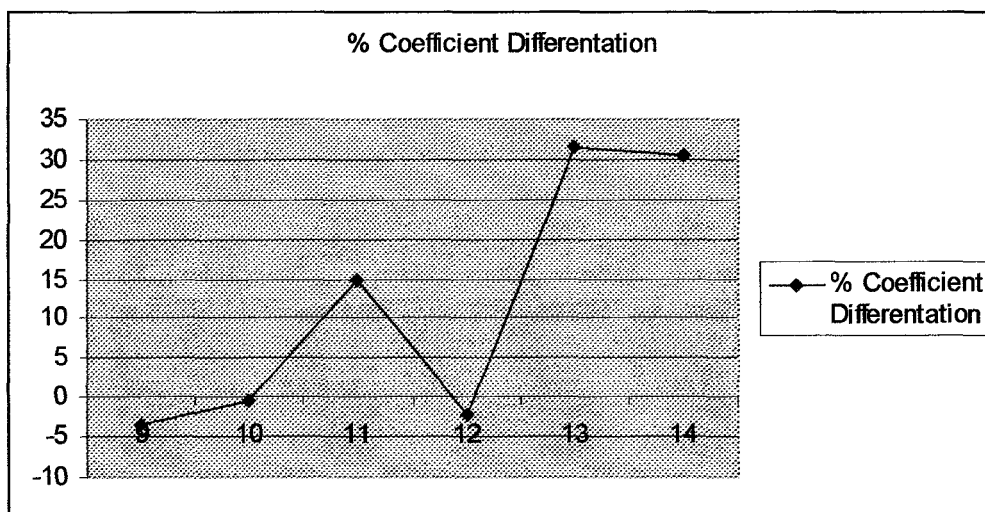


Figure 4-22: The percentage coefficient differentiation between TCP rate control and standard TCP for case study 9-14

The above figure represents the graph of the percentage of differentiation between TCP rate control and standard TCP. Referring to the graph TCP rate control, work very well in case 13 – 14 in which I reduced the bandwidth to 1 Mbps from 10 Mbps and 5 Mbps.

4.6.3 Summary Discussion of Results

Referring to the 2 categories of the simulation with 14 cases, each category I adjusted the bandwidth of the link to get the various results. And from the results, using TCP rate control technology works better than standard TCP in terms of being able to send more TCP packets with the congested environment. With non congested environment, TCP rate control and standard TCP gave the similar results. However, when using TCP rate control algorithm, the received UDP packets is less than by using standard TCP in a congested environment. This is the limitation of TCP rate control algorithm for both topologies.

5. CONCLUSION AND RECOMMENDATION

In this thesis, the simulation of TCP rate control and standard TCP are presented for both single hop and multi hop topology. The problem of standard TCP is that it uses slow-start algorithm for congestion control. With TCP slow-start, when a connection opens, only one packet is sent until an ACK is received. For each ACK receives ACK, the sender can double the transmission size. Note that this is exponential growth rate. But eventually packets are dropped. With normal transmission of TCP, the sending rate is not based on congestion environment and when network is under the problem of loss packets and retransmission can occur. Dropped and retransmission packet can increase latency. This is the significant role of this thesis to point out that TCP rate control based on congestion environment can reduce loss and retransmit packets and can decrease latency.

The proposed algorithm, TCP rate control, is a new technique for transparently augmenting end-to-end TCP performance by controlling the sending rate of a TCP source. The sending rate of TCP source is determined by its window size, the round trip time and the rate of acknowledgement. It controls the rate of TCP packets by controlling window size and the rate of acknowledgement based on congestion environment.

This thesis separated the simulation into 2 categories, single hop and multi hop topology with various link speeds. In the simulation of both topologies with 14 cases, I adjusted the bandwidth to make a congested environment. The result of both topologies show that using TCP rate control technology can give a higher performance than standard TCP in terms of sending more TCP packets in the

congested environment. For single hop topology the maximum difference between both algorithms is 10.32% for total received TCP packets. For multi hop topology it is 30.57%. However, with a non congested environment, TCP rate control and standard TCP gave the similar results. The difference between both algorithms is less than 5% for total receive TCP packets. However when using TCP rate control algorithm, the received UDP packets is less than using standard TCP in congested environment. This is the limitation of TCP rate control algorithm for both topologies.

The main contribution for this study is the evaluation of TCP rate control algorithm. This study shows that TCP rate control gives a better performance than standard TCP in congested environment for LAN switch network.

This TCP rate control is done only in LAN switch network. Usually the problem of TCP occurs in WAN network. Future work will study TCP rate control on the WAN network.

REFERENCES

- [1] "TCP Slow Start, Congestion Avoidance, Fast Retransmit, and Fast Recovery Algorithms", W. Stevens, NAAO, RFC2001, January 1997.
- [2] "TCP Selective Acknowledgment Options", M. Mathis, J. Mahdari PSC, S. Floyd LBNL, A. Romanow, Sun Microsystems, RFC2018 October 1996
- [3] "TCP Rate Control", Shrikrishna Karandikar, Shivkumar Kalyanaraman, Prasad Bagal, Bob Packer, Department of ECSE, Department of Computer Science, Rensselaer Polytechnic Institute, 2000
- [4] "Simulation-based Comparisons of Tahoe, Reno, and SACK TCP", Kevin Fall and Sally Floyd, Lawrence Berkeley National Laboratory, One Cyclotron Road, Berkeley, CA 94720, 1997
- [5] "Transport Control Protocol", Information Sciences Institute University of Southern California 4676 Admiralty Way Marina del Rey, California 90291, RFC 793, September 1981
- [6] "TCP Illustrated, Volume 1, W. Richard Stevens, Addison-Wesley Professional Computing Series, 1994
- [7] "Comparative study of RED, ECN and TCP Rate Control", Prasad Bagal, Shivkumar Kalyanaraman, Bob Packer, Department of ECSE, Rensselaer Polytechnic Institute, 1999

[8] “Computer Networks”, Andrew S. Tanenbaum, Prentice Hall Inc, 1996

[9] “An Algorithm for Rate Allocation in a Packet-Switching Network with Feedback”, Anna Charny, Massachusetts Institute of Technology, May 1994



APPENDIX

Programming Source Code

```
#####
# Case study 1 - Standard TCP
#####

#Create a simulator object
set ns [new Simulator]

set f [open out.tr w]
$ns trace-all $f

#set x [open x.trace w]
set t [open out.tcp w]
set a [open out.ack w]

set t_parrival [open out.pa w]

#Open the nam trace file
set nf [open out.nam w]
$ns namtrace-all $nf

proc finish {} {
    global ns nf x t a
    $ns flush-trace
    #Close the trace file
    close $f
    #
    close $nf
    #Execute nam on the trace file

    exec awk {
    {
        if($1 == "-" && ($5 == "tcp") && ($3 == "2"))
            print $2,$11,$5
    }
    } out.tr > out.tcp

    exec awk {
    {
        if($1 == "-" && ($5 == "cbr") && ($3 == "2"))
            print $2,$11,$5
    }
    } out.tr > out.cbr

    exec awk {
    {
        if($1 == "r") && ($5 == "tcp") && ($4 == "0"))
            print $2,$11,$5
    }
    } out.tr > out.tcpr
    exec awk {
    {
        if($1 == "r") && ($5 == "cbr") && ($4 == "0"))
            print $2,$11,$5
    }
    } out.tr > out.cbrr

    exec awk {
    {
        if($1 == "d") && ($5 == "cbr"))
            print $2,$11
    }
    } out.tr > out.dcbr
```

```

    exec awk {
        {
            if (($1 == "d") && ($5 == "tcp"))
                print $2,$11
        }
    } out.tr > out.dtcp

    exec nam out.nam &
    exit 0
}

```

```

#Create nodes
set d [$ns node]
set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]
set n4 [$ns node]
set n5 [$ns node]

```

```

$ns duplex-link $n1 $d 100mb 10ms DropTail
$ns duplex-link $n0 $n1 100mb 10ms DropTail
$ns duplex-link $n2 $n1 100mb 10ms DropTail
$ns duplex-link $n3 $n1 100mb 10ms DropTail
$ns duplex-link $n4 $n1 100mb 10ms DropTail
$ns duplex-link $n5 $n1 100mb 10ms DropTail

```

```

# Set up BSD Sack TCP connection in opposite directions.
#
set src1 [$ns create-connection TCP/Sack1 $n0 TCPSink/Sack1 $d 5]
set src2 [$ns create-connection TCP/Sack1 $n2 TCPSink/Sack1 $d 6]
set src3 [$ns create-connection TCP/Sack1 $n3 TCPSink/Sack1 $d 7]
set src4 [$ns create-connection TCP/Sack1 $n4 TCPSink/Sack1 $d 8]
set src5 [$ns create-connection TCP/Sack1 $n5 TCPSink/Sack1 $d 9]

```

```

#
# Create ftp sources at the each node
#
set ftp1 [$src1 attach-app FTP]
set ftp2 [$src2 attach-app FTP]
set ftp3 [$src3 attach-app FTP]
set ftp4 [$src4 attach-app FTP]
set ftp5 [$src5 attach-app FTP]

```

```

#
# Start up the first ftp at the time 0 and
# the second ftp staggered 1 second later
#
$ns at 0.0 "$ftp1 start"
$ns at 0.0 "$ftp2 start"
$ns at 0.0 "$ftp3 start"
$ns at 0.0 "$ftp4 start"
$ns at 0.0 "$ftp5 start"

```

```

$ns at 3.0 "finish"

```

```

#Run the simulation
$ns run

```

```

#####
# Case study 1 – TCP Rate Control
#####

```

```

#Create a simulator object
set ns [new Simulator]

```

```

set f [open out.tr w]
$ns trace-all $f

```

```

#set x [open x.trace w]
set t [open out.tcp w]
set a [open out.ack w]

```

```

set t_parrival [open out.pa w]

```

```

#Open the nam trace file
set nf [open out.nam w]
$ns namtrace-all $nf

#Record procedure
proc record1 {parr1 r aa} {
    global ns rtt ai src1 sink qmond qmond1 qmon01 t_parrival aa

    set time 0.0001
    set a [$qmon01 set parrivals_]
    set b [$qmond set pdepartures_]
    set c [$qmond1 set parrivals_]

    if {$parr1 <= $a} {
        set now [$ns now]

        #hungry flow, find Ai
        if {$r >= $aa} {
            #u is flow in the link and h is bw those flows are using
            set u [expr $b-$c]
            set h [expr {$u*1000}]
            set ai [expr $aa+$h]
        }

        #bottleneck flow, find Ai
        else {
            set ri [expr {$r/2}]
            set aaa [expr {$aa/2}]
            set ai [expr $ri+$aaa]
        }
        set rtt [expr [$src1 set rtt_]]
        set amss [expr {$ai*0.00001}]
        set window [expr [$src1 set rtt_] * $amss]
        set de [expr {$rtt/$window}]
        $ns at [expr $now "$sink set interval_ $de"]
        $ns at [expr $now "$src1 set cwnd_ $window"]
        set parr1 [expr {$parr1 + $window}]
        set r [expr 1000000/$rtt]
    }

    set now [$ns now]
    $ns at [expr $now+$time] "record1 $parr1 $r $ai"
}

proc record2 {parr2 r aa} {
    global ns rtt ai src2 sink qmond qmond1 qmon21 t_parrival aa

    set time 0.0001
    set a [$qmon21 set parrivals_]
    set b [$qmond set pdepartures_]
    set c [$qmond1 set parrivals_]

    if {$parr2 <= $a} {
        set now [$ns now]

        #hungry flow, find Ai
        if {$r >= $aa} {
            #u is flow in the link and h is bw those flows are using
            set u [expr $b-$c]
            set h [expr {$u*1000}]
            set ai [expr $aa+$h]
        }

        #bottleneck flow, find Ai
        else {
            set ri [expr {$r/2}]
            set aaa [expr {$aa/2}]
            set ai [expr $ri+$aaa]
        }
        set rtt [expr [$src2 set rtt_]]
        set amss [expr {$ai*0.00001}]
        set window [expr [$src2 set rtt_] * $amss]
        set de [expr {$rtt/$window}]
        $ns at [expr $now "$sink set interval_ $de"]
        $ns at [expr $now "$src2 set cwnd_ $window"]
    }
}

```



```

set parr2 [expr {$parr2 + $window}]
set r [expr 1000000/$rtt]
}

set now [$ns now]
$ns at [expr $now+$time] "record2 $parr2 $r $ai"
}

proc record3 {parr3 r aa} {
    global ns rtt ai src3 sink qmond qmond1 qmon31 t_parrival aa

    set time 0.0001
    set a [$qmon31 set parrivals_]
    set b [$qmond set pdepartures_]
    set c [$qmond1 set parrivals_]

    if {$parr3 <= $a} {
        set now [$ns now]

        #hungry flow, find Ai
        if {$r >= $aa} {
            #u is flow in the link and h is bw those flows are using
            set u [expr $b-$c]
            set h [expr {$u*1000}]
            set ai [expr $aa+$h]
        }

        #bottleneck flow, find Ai
        else {
            set ri [expr {$r/2}]
            set aaa [expr {$aa/2}]
            set ai [expr $ri+$aaa]
        }
        set rtt [expr [$src3 set rtt_]]
        set amss [expr {$ai*0.00001}]
        set window [expr [$src3 set rtt_] * $amss]
        set dc [expr {$rtt/$window}]
        $ns at [expr $now] "$sink set interval_ $dc"
        $ns at [expr $now] "$src3 set cwnd_ $window"
        set parr3 [expr {$parr3 + $window}]
        set r [expr 1000000/$rtt]
    }

    set now [$ns now]
    $ns at [expr $now+$time] "record3 $parr3 $r $ai"
}

proc record4 {parr4 r aa} {
    global ns rtt ai src4 sink qmond qmond1 qmon41 t_parrival aa

    set time 0.0001
    set a [$qmon41 set parrivals_]
    set b [$qmond set pdepartures_]
    set c [$qmond1 set parrivals_]

    if {$parr4 <= $a} {
        set now [$ns now]

        #hungry flow, find Ai
        if {$r >= $aa} {
            #u is flow in the link and h is bw those flows are using
            set u [expr $b-$c]
            set h [expr {$u*2000}]
            set ai [expr $aa+$h]
        }

        #bottleneck flow, find Ai
        else {
            set ri [expr {$r/2}]
            set aaa [expr {$aa/2}]
            set ai [expr $ri+$aaa]
        }
        set rtt [expr [$src4 set rtt_]]
        set amss [expr {$ai*0.00001}]
        set window [expr [$src4 set rtt_] * $amss]
    }
}

```

```

set de [expr {$rtt/$window}]
$ns at [expr $now] "$sink set interval_ $de]
$ns at [expr $now] "$src4 set cwnd_ $window"
set parr4 [expr {$parr4 + $window}]
set r [expr 1000000/$rtt]
}

set now [$ns now]
$ns at [expr $now+$time] "record4 $parr4 $r $ai"
}

proc record5 {parr5 r aa} {
    global ns rtt ai src5 sink qmond qmond1 qmon51 t_parrival aa

    set time 0.0001
    set a [$qmon51 set parrivals_]
    set b [$qmond set pdepartures_]
    set c [$qmond1 set parrivals_]

    if {$parr5 <= $a} {
        set now [$ns now]

        #hungry flow, find Ai
        if {$r >= $aa} {
            #u is flow in the link and h is bw those flows are using
            set u [expr $b-$c]
            set h [expr {$u*1000}]
            set ai [expr $aa+$h]
        }

        #bottleneck flow, find Ai
        else {
            set ri [expr {$r/2}]
            set aaa [expr {$aa/2}]
            set ai [expr $ri+$aaa]
        }
        set rtt [expr [$src5 set rtt_]]
        set amss [expr {$ai*0.00001}]
        set window [expr [$src5 set rtt_] * $amss]
        set de [expr {$rtt/$window}]
        $ns at [expr $now] "$sink set interval_ $de]
        $ns at [expr $now] "$src5 set cwnd_ $window"
        set parr5 [expr {$parr5 + $window}]
        set r [expr 1000000/$rtt]
    }

    set now [$ns now]
    $ns at [expr $now+$time] "record5 $parr5 $r $ai"
}

proc finish {} {
    global ns nf x t a
    $ns flush-trace
    #Close the trace file
    close $f
    #
    close $nf
    #Execute nam on the trace file

    exec awk {
    {
        if($1 == "-" && ($5 == "tcp") && ($3 == "2"))
            print $2,$11,$5

    }
    } out.tr > out.tcp

    exec awk {
    {
        if($1 == "-" && ($5 == "cbr") && ($3 == "2"))
            print $2,$11,$5

    }
    } out.tr > out.cbr
}

```

```

exec awk {
{
    if(($1 == "r") && ($5 == "tcp") && ($4 == "0"))
        print $2,$11,$5
    }
} out.tr > out.tcpr

exec awk {
{
    if(($1 == "r") && ($5 == "cbr") && ($4 == "0"))
        print $2,$11,$5
    }
} out.tr > out.cbrr

exec awk {
{
    if(($1 == "d") && ($5 == "cbr"))
        print $2,$11
    }
} out.tr > out.dabr

exec awk {
{
    if(($1 == "d") && ($5 == "tcp"))
        print $2,$11
    }
} out.tr > out.dtcp

exec nam out.nam &
exit 0
}

#Create nodes
set d [$ns node]
set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]
set n4 [$ns node]
set n5 [$ns node]

$ns duplex-link $n1 $d 100mb 10ms DropTail
$ns duplex-link $n0 $n1 100mb 10ms DropTail
$ns duplex-link $n2 $n1 100mb 10ms DropTail
$ns duplex-link $n3 $n1 100mb 10ms DropTail
$ns duplex-link $n4 $n1 100mb 10ms DropTail
$ns duplex-link $n5 $n1 100mb 10ms DropTail

set qmon0 [$ns monitor-queue $n0 $n1 1]
set qmon01 [$ns monitor-queue $n1 $n0 1]

set qmon2 [$ns monitor-queue $n2 $n1 1]
set qmon21 [$ns monitor-queue $n1 $n2 1]

set qmon3 [$ns monitor-queue $n3 $n1 1]
set qmon31 [$ns monitor-queue $n1 $n3 1]

set qmon4 [$ns monitor-queue $n4 $n1 1]
set qmon41 [$ns monitor-queue $n1 $n4 1]

set qmon5 [$ns monitor-queue $n5 $n1 1]
set qmon51 [$ns monitor-queue $n1 $n5 1]

set qmond [$ns monitor-queue $n1 $d 1]
set qmond1 [$ns monitor-queue $d $n1 1]

# Set up BSD Sack TCP connection in opposite directions.
#
set src1 [$ns create-connection TCP/Sack1 $n0 TCPSink/Sack1 $d 5]
set src2 [$ns create-connection TCP/Sack1 $n2 TCPSink/Sack1 $d 6]
set src3 [$ns create-connection TCP/Sack1 $n3 TCPSink/Sack1 $d 7]

```

```

set src4 [$Sns create-connection TCP/Sack1 $n4 TCPSink/Sack1 $d 8]
set src5 [$Sns create-connection TCP/Sack1 $n5 TCPSink/Sack1 $d 9]

```

```

#
# Create ftp sources at the each node
#

```

```

set ftp1 [$Src1 attach-app FTP]
set ftp2 [$Src2 attach-app FTP]
set ftp3 [$Src3 attach-app FTP]
set ftp4 [$Src4 attach-app FTP]
set ftp5 [$Src5 attach-app FTP]

```

```

#
# Start up the first ftp at the time 0 and
# the second ftp staggered 1 second later
#

```

```

$Sns at 0.0 "$ftp1 start"
$Sns at 0.0 "$ftp2 start"
$Sns at 0.0 "$ftp3 start"
$Sns at 0.0 "$ftp4 start"
$Sns at 0.0 "$ftp5 start"

```

```

$Sns at 0.0 "record1 1 12500000 12500000"
$Sns at 0.0 "record2 1 12500000 12500000"
$Sns at 0.0 "record3 1 12500000 12500000"
$Sns at 0.0 "record4 1 12500000 12500000"
$Sns at 0.0 "record5 1 12500000 12500000"

```

```

$Sns at 3.0 "finish"

```

```

#Run the simulation
$Sns run

```

```

#####
# Case study 2 - Standard TCP
# (Modified from case study 1 – Standard TCP)
#####

```

```

#Create nodes
set d [$Sns node]
set n0 [$Sns node]
set n1 [$Sns node]
set n2 [$Sns node]
set n3 [$Sns node]
set n4 [$Sns node]
set n5 [$Sns node]
set n6 [$Sns node]
set n7 [$Sns node]
set n8 [$Sns node]
set n9 [$Sns node]
set n10 [$Sns node]
set n11 [$Sns node]
set n12 [$Sns node]
set n13 [$Sns node]
set n14 [$Sns node]
set n15 [$Sns node]

```

```

$Sns duplex-link $n1 $d 100mb 10ms DropTail
$Sns duplex-link $n0 $n1 100mb 10ms DropTail
$Sns duplex-link $n2 $n1 100mb 10ms DropTail
$Sns duplex-link $n3 $n1 100mb 10ms DropTail
$Sns duplex-link $n4 $n1 100mb 10ms DropTail
$Sns duplex-link $n5 $n1 100mb 10ms DropTail
$Sns duplex-link $n6 $n1 100mb 10ms DropTail
$Sns duplex-link $n7 $n1 100mb 10ms DropTail
$Sns duplex-link $n8 $n1 100mb 10ms DropTail
$Sns duplex-link $n9 $n1 100mb 10ms DropTail
$Sns duplex-link $n10 $n1 100mb 10ms DropTail
$Sns duplex-link $n11 $n1 100mb 10ms DropTail
$Sns duplex-link $n12 $n1 100mb 10ms DropTail
$Sns duplex-link $n13 $n1 100mb 10ms DropTail
$Sns duplex-link $n14 $n1 100mb 10ms DropTail

```

```
$ns duplex-link $n15 $n1 100mb 10ms DropTail
```

```
# Set up BSD Sack TCP connection in opposite directions.
```

```
#
```

```
set src1 [$ns create-connection TCP/Sack1 $n0 TCPSink/Sack1 $d 5]  
set src2 [$ns create-connection TCP/Sack1 $n2 TCPSink/Sack1 $d 6]  
set src3 [$ns create-connection TCP/Sack1 $n3 TCPSink/Sack1 $d 7]  
set src4 [$ns create-connection TCP/Sack1 $n4 TCPSink/Sack1 $d 8]  
set src5 [$ns create-connection TCP/Sack1 $n5 TCPSink/Sack1 $d 9]
```

```
#
```

```
# Create ftp sources at the each node
```

```
#
```

```
set ftp1 [$src1 attach-app FTP]  
set ftp2 [$src2 attach-app FTP]  
set ftp3 [$src3 attach-app FTP]  
set ftp4 [$src4 attach-app FTP]  
set ftp5 [$src5 attach-app FTP]
```

```
set udp0 [new Agent/UDP]  
$ns attach-agent $n6 $udp0  
set cbr0 [new Application/Traffic/CBR]  
$cbr0 attach-agent $udp0  
$cbr0 set packetSize_ 1000
```

```
set udp1 [new Agent/UDP]  
$ns attach-agent $n7 $udp1  
set cbr1 [new Application/Traffic/CBR]  
$cbr1 attach-agent $udp1  
$cbr1 set packetSize_ 1000
```

```
set udp2 [new Agent/UDP]  
$ns attach-agent $n8 $udp2  
set cbr2 [new Application/Traffic/CBR]  
$cbr2 attach-agent $udp2  
$cbr2 set packetSize_ 1000
```

```
set udp3 [new Agent/UDP]  
$ns attach-agent $n9 $udp3  
set cbr3 [new Application/Traffic/CBR]  
$cbr3 attach-agent $udp3  
$cbr3 set packetSize_ 1000
```

```
set udp4 [new Agent/UDP]  
$ns attach-agent $n10 $udp4  
set cbr4 [new Application/Traffic/CBR]  
$cbr4 attach-agent $udp4  
$cbr4 set packetSize_ 1000
```

```
set udp5 [new Agent/UDP]  
$ns attach-agent $n11 $udp5  
set cbr5 [new Application/Traffic/CBR]  
$cbr5 attach-agent $udp5  
$cbr5 set packetSize_ 1000
```

```
set udp6 [new Agent/UDP]  
$ns attach-agent $n12 $udp6  
set cbr6 [new Application/Traffic/CBR]  
$cbr6 attach-agent $udp6  
$cbr6 set packetSize_ 1000
```

```
set udp7 [new Agent/UDP]  
$ns attach-agent $n13 $udp7  
set cbr7 [new Application/Traffic/CBR]  
$cbr7 attach-agent $udp7  
$cbr7 set packetSize_ 1000
```

```
set udp8 [new Agent/UDP]  
$ns attach-agent $n14 $udp8  
set cbr8 [new Application/Traffic/CBR]  
$cbr8 attach-agent $udp8  
$cbr8 set packetSize_ 1000
```

```
set udp9 [new Agent/UDP]  
$ns attach-agent $n15 $udp9
```

```

set cbr9 [new Application/Traffic/CBR]
$scr9 attach-agent $udp9
$scr9 set packetSize_ 1000

```

```

set null0 [new Agent/Null]
$ns attach-agent $d $null0

```

```

$ns connect $udp0 $null0
$ns connect $udp1 $null0
$ns connect $udp2 $null0
$ns connect $udp3 $null0
$ns connect $udp4 $null0
$ns connect $udp5 $null0
$ns connect $udp6 $null0
$ns connect $udp7 $null0
$ns connect $udp8 $null0
$ns connect $udp9 $null0

```

```

#
# Start up the first ftp at the time 0 and
# the second ftp staggered 1 second later
#

```

```

$ns at 0.0 "$ftp1 start"
$ns at 0.0 "$ftp2 start"
$ns at 0.0 "$ftp3 start"
$ns at 0.0 "$ftp4 start"
$ns at 0.0 "$ftp5 start"

```

```

$ns at 0.0 "$scr0 start"
$ns at 0.0 "$scr1 start"
$ns at 0.0 "$scr2 start"
$ns at 0.0 "$scr3 start"
$ns at 0.0 "$scr4 start"
$ns at 0.0 "$scr5 start"
$ns at 0.0 "$scr6 start"
$ns at 0.0 "$scr7 start"
$ns at 0.0 "$scr8 start"
$ns at 0.0 "$scr9 start"

```

```

$ns at 3.0 "finish"

```

```

#Run the simulation
$ns run

```

```

#####
# Case study 2 – TCP Rate Control
# (Modified from case study 1 –TCP Rate Control)
#####

```

```

set udp0 [new Agent/UDP]
$ns attach-agent $n6 $udp0
set cbr0 [new Application/Traffic/CBR]
$scr0 attach-agent $udp0
$scr0 set packetSize_ 1000

```

```

set udp1 [new Agent/UDP]
$ns attach-agent $n7 $udp1
set cbr1 [new Application/Traffic/CBR]
$scr1 attach-agent $udp1
$scr1 set packetSize_ 1000

```

```

set null0 [new Agent/Null]
$ns attach-agent $d $null0
$ns connect $udp0 $null0
$ns connect $udp1 $null0

```

```

$ns at 0.0 "$ftp1 start"
$ns at 0.0 "$ftp2 start"
$ns at 0.0 "$ftp3 start"
$ns at 0.0 "$ftp4 start"
$ns at 0.0 "$ftp5 start"

```



```

$ns at 0.0 "$scr0 start"
$ns at 0.0 "$scr1 start"

ns at 0.0 "record1 1 12500000 12500000"
$ns at 0.0 "record2 1 12500000 12500000"
$ns at 0.0 "record3 1 12500000 12500000"
$ns at 0.0 "record4 1 12500000 12500000"
$ns at 0.0 "record5 1 12500000 12500000"

$ns at 3.0 "finish"

#Run the simulation
$ns run

#####
# Case study 3 - Standard TCP
# (Modified from case study 1 – Standard TCP)
#####

.....

#Create nodes
set d [$ns node]
set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set n6 [$ns node]
set n7 [$ns node]
set n8 [$ns node]
set n9 [$ns node]
set n10 [$ns node]
set n11 [$ns node]
set n12 [$ns node]
set n13 [$ns node]
set n14 [$ns node]
set n15 [$ns node]
set n16 [$ns node]
set n17 [$ns node]
set n18 [$ns node]
set n19 [$ns node]
set n20 [$ns node]

$ns duplex-link $n1 $d 100mb 10ms DropTail
$ns duplex-link $n0 $n1 100mb 10ms DropTail
$ns duplex-link $n2 $n1 100mb 10ms DropTail
$ns duplex-link $n6 $n1 100mb 10ms DropTail
$ns duplex-link $n7 $n1 100mb 10ms DropTail
$ns duplex-link $n8 $n1 100mb 10ms DropTail
$ns duplex-link $n9 $n1 100mb 10ms DropTail
$ns duplex-link $n10 $n1 100mb 10ms DropTail
$ns duplex-link $n11 $n1 100mb 10ms DropTail
$ns duplex-link $n12 $n1 100mb 10ms DropTail
$ns duplex-link $n13 $n1 100mb 10ms DropTail
$ns duplex-link $n14 $n1 100mb 10ms DropTail
$ns duplex-link $n15 $n1 100mb 10ms DropTail
$ns duplex-link $n16 $n1 100mb 10ms DropTail
$ns duplex-link $n17 $n1 100mb 10ms DropTail
$ns duplex-link $n18 $n1 100mb 10ms DropTail
$ns duplex-link $n19 $n1 100mb 10ms DropTail
$ns duplex-link $n20 $n1 100mb 10ms DropTail

# Set up BSD Sack TCP connection in opposite directions.
#
set src1 [$ns create-connection TCP/Sack1 $n0 TCPSink/Sack1 $d 5]
set src2 [$ns create-connection TCP/Sack1 $n2 TCPSink/Sack1 $d 6]

#
# Create ftp sources at the each node
#
set ftp1 [$src1 attach-app FTP]
set ftp2 [$src2 attach-app FTP]

```

```

set udp0 [new Agent/UDP]
$ns attach-agent $n6 $udp0
set cbr0 [new Application/Traffic/CBR]
$scr0 attach-agent $udp0
$scr0 set packetSize_ 1000

set udp1 [new Agent/UDP]
$ns attach-agent $n7 $udp1
set cbr1 [new Application/Traffic/CBR]
$scr1 attach-agent $udp1
$scr1 set packetSize_ 1000

set udp2 [new Agent/UDP]
$ns attach-agent $n8 $udp2
set cbr2 [new Application/Traffic/CBR]
$scr2 attach-agent $udp2
$scr2 set packetSize_ 1000

set udp3 [new Agent/UDP]
$ns attach-agent $n9 $udp3
set cbr3 [new Application/Traffic/CBR]
$scr3 attach-agent $udp3
$scr3 set packetSize_ 1000

set udp4 [new Agent/UDP]
$ns attach-agent $n10 $udp4
set cbr4 [new Application/Traffic/CBR]
$scr4 attach-agent $udp4
$scr4 set packetSize_ 1000

set udp5 [new Agent/UDP]
$ns attach-agent $n11 $udp5
set cbr5 [new Application/Traffic/CBR]
$scr5 attach-agent $udp5
$scr5 set packetSize_ 1000

set udp6 [new Agent/UDP]
$ns attach-agent $n12 $udp6
set cbr6 [new Application/Traffic/CBR]
$scr6 attach-agent $udp6
$scr6 set packetSize_ 1000

set udp7 [new Agent/UDP]
$ns attach-agent $n13 $udp7
set cbr7 [new Application/Traffic/CBR]
$scr7 attach-agent $udp7
$scr7 set packetSize_ 1000

set udp8 [new Agent/UDP]
$ns attach-agent $n14 $udp8
set cbr8 [new Application/Traffic/CBR]
$scr8 attach-agent $udp8
$scr8 set packetSize_ 1000

set udp9 [new Agent/UDP]
$ns attach-agent $n15 $udp9
set cbr9 [new Application/Traffic/CBR]
$scr9 attach-agent $udp9
$scr9 set packetSize_ 1000

set udp10 [new Agent/UDP]
$ns attach-agent $n16 $udp10
set cbr10 [new Application/Traffic/CBR]
$scr10 set packetSize_ 1000
$scr10 attach-agent $udp10

set udp11 [new Agent/UDP]
$ns attach-agent $n17 $udp11
set cbr11 [new Application/Traffic/CBR]
$scr11 attach-agent $udp11
$scr11 set packetSize_ 1000

set udp12 [new Agent/UDP]
$ns attach-agent $n18 $udp12
set cbr12 [new Application/Traffic/CBR]

```

```
$cbr12 attach-agent $udp12
$cbr12 set packetSize_ 1000
```

```
set udp13 [new Agent/UDP]
$ns attach-agent $n19 $udp13
set cbr13 [new Application/Traffic/CBR]
$cbr13 attach-agent $udp13
$cbr13 set packetSize_ 1000
```

```
set udp14 [new Agent/UDP]
$ns attach-agent $n20 $udp14
set cbr14 [new Application/Traffic/CBR]
$cbr14 attach-agent $udp14
$cbr14 set packetSize_ 1000
```

```
set null0 [new Agent/Null]
$ns attach-agent $d $null0
$ns connect $udp0 $null0
$ns connect $udp1 $null0
$ns connect $udp2 $null0
$ns connect $udp3 $null0
$ns connect $udp4 $null0
$ns connect $udp5 $null0
$ns connect $udp6 $null0
$ns connect $udp7 $null0
$ns connect $udp8 $null0
$ns connect $udp9 $null0
$ns connect $udp10 $null0
$ns connect $udp11 $null0
$ns connect $udp12 $null0
$ns connect $udp13 $null0
$ns connect $udp14 $null0
```

```
#
# Start up the first ftp at the time 0 and
# the second ftp staggered 1 second later
#
$ns at 0.0 "$ftp1 start"
$ns at 0.0 "$ftp2 start"
```

```
$ns at 0.0 "$cbr0 start"
$ns at 0.0 "$cbr1 start"
$ns at 0.0 "$cbr2 start"
$ns at 0.0 "$cbr3 start"
$ns at 0.0 "$cbr4 start"
$ns at 0.0 "$cbr5 start"
$ns at 0.0 "$cbr6 start"
$ns at 0.0 "$cbr7 start"
$ns at 0.0 "$cbr8 start"
$ns at 0.0 "$cbr9 start"
$ns at 0.0 "$cbr10 start"
$ns at 0.0 "$cbr11 start"
$ns at 0.0 "$cbr12 start"
$ns at 0.0 "$cbr13 start"
$ns at 0.0 "$cbr14 start"
$ns at 3.0 "finish"
```

```
#Run the simulation
$ns run
```

```
#####
# Case study 3 – TCP Rate Control
# (Modified from case study 3 – Standard TCP)
#####
```

```
#Create a simulator object
set ns [new Simulator]
```

```
set f [open out.tr w]
$ns trace-all $f
```

```

#set x [open x.trace w]
set t [open out.tcp w]
set a [open out.ack w]

set t_parrival [open out.pa w]

#Open the nam trace file
set nf [open out.nam w]
$ns namtrace-all $nf

#Record procedure
proc record1 {parr1 r aa} {
    global ns rtt ai src1 sink qmond qmond1 qmon01 t_parrival aa

    set time 0.0001
    set a [$qmon01 set parrivals_]
    set b [$qmond set pdepartures_]
    set c [$qmond1 set parrivals_]

    if {$parr1 <= $a} {
        set now [$ns now]

        #hungry flow, find Ai
        if {$r >= $aa} {
            #u is flow in the link and h is bw those flows are using
            set u [expr $b-$c]
            set h [expr {$u*1000}]
            set ai [expr $aa+$h]
        }

        #bottleneck flow, find Ai
        else {
            set ri [expr {$r/2}]
            set aaa [expr {$aa/2}]
            set ai [expr $ri+$aaa]
        }
        set rtt [expr [$src1 set rtt_] ]
        set amss [expr {$ai*0.00001}]
        set window [expr [$src1 set rtt_] * $amss]
        set dc [expr {$rtt/$window}]
        $ns at [expr $now] "$sink set interval_ $dc"
        $ns at [expr $now] "$src1 set cwnd_ $window"
        set parr1 [expr {$parr1 + $window}]
        set r [expr 1000000/$rtt]
    }

    set now [$ns now]
    $ns at [expr $now+$time] "record1 $parr1 $r $ai"
}

proc record2 {parr2 r aa} {
    global ns rtt ai src2 sink qmond qmond1 qmon21 t_parrival aa

    set time 0.0001
    set a [$qmon21 set parrivals_]
    set b [$qmond set pdepartures_]
    set c [$qmond1 set parrivals_]

    if {$parr2 <= $a} {
        set now [$ns now]

        #hungry flow, find Ai
        if {$r >= $aa} {
            #u is flow in the link and h is bw those flows are using
            set u [expr $b-$c]
            set h [expr {$u*1000}]
            set ai [expr $aa+$h]
        }

        #bottleneck flow, find Ai
        else {
            set ri [expr {$r/2}]
            set aaa [expr {$aa/2}]
            set ai [expr $ri+$aaa]
        }
    }
}

```

```

set rtt [expr {$src2 set srtt_}]
set amss [expr {$sai*0.00001}]
set window [expr {$src2 set srtt_} * $amss]
set de [expr {$rtt/$window}]
$ns at [expr $now "$sink set interval_ $de]
$ns at [expr $now] "$src2 set cwnd_ $window"
set parr2 [expr {$parr2 + $window}]
set r [expr 1000000/$rtt]
}

set now [$ns now]
$ns at [expr $now+$time] "record2 $parr2 $r $sai"
}

```

```

set qmon0 [$ns monitor-queue $n0 $n1 1]
set qmon01 [$ns monitor-queue $n1 $n0 1]

set qmon2 [$ns monitor-queue $n2 $n1 1]
set qmon21 [$ns monitor-queue $n1 $n2 1]

set qmond [$ns monitor-queue $n1 $d 1]
set qmond1 [$ns monitor-queue $d $n1 1]

```

```

$ns at 0.00 "record1 1 125000 125000"
$ns at 0.00 "record2 1 125000 125000"

```

```

$ns at 3.0 "finish"

```

```

#Run the simulation
$ns run

```

```

#####
# Case study 4 - Standard TCP
# (Modified from case study 1 - Standard TCP)
#####

```

```

$ns duplex-link $n1 $d 10mb 10ms DropTail
$ns duplex-link $n0 $n1 100mb 10ms DropTail
$ns duplex-link $n2 $n1 100mb 10ms DropTail
$ns duplex-link $n3 $n1 100mb 10ms DropTail
$ns duplex-link $n4 $n1 100mb 10ms DropTail
$ns duplex-link $n5 $n1 100mb 10ms DropTail

```

```

#####
# Case study 4 - TCP Rate Control
# (Modified from case study 1 - TCP Rate Control)
#####

```

```

$ns duplex-link $n1 $d 10mb 10ms DropTail
$ns duplex-link $n0 $n1 100mb 10ms DropTail
$ns duplex-link $n2 $n1 100mb 10ms DropTail
$ns duplex-link $n3 $n1 100mb 10ms DropTail
$ns duplex-link $n4 $n1 100mb 10ms DropTail
$ns duplex-link $n5 $n1 100mb 10ms DropTail

```

```

$ns at 0.00 "record1 1 1250000 1250000"
$ns at 0.00 "record2 1 1250000 1250000"
$ns at 0.00 "record3 1 1250000 1250000"
$ns at 0.00 "record4 1 1250000 1250000"
$ns at 0.00 "record5 1 1250000 1250000"

```

```

#####
# Case study 5 - Standard TCP
# (Modified from case study 1 - Standard TCP)
#####

```

```

#Create nodes
set d [$ns node]
set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]
set n4 [$ns node]
set n5 [$ns node]
set n6 [$ns node]
set n7 [$ns node]
set n8 [$ns node]
set n9 [$ns node]
set n10 [$ns node]

```

```

$ns duplex-link $n1 $d 10mb 10ms DropTail
$ns duplex-link $n0 $n1 100mb 10ms DropTail
$ns duplex-link $n2 $n1 100mb 10ms DropTail
$ns duplex-link $n3 $n1 100mb 10ms DropTail
$ns duplex-link $n4 $n1 100mb 10ms DropTail
$ns duplex-link $n5 $n1 100mb 10ms DropTail
$ns duplex-link $n6 $n1 100mb 10ms DropTail
$ns duplex-link $n7 $n1 100mb 10ms DropTail
$ns duplex-link $n8 $n1 100mb 10ms DropTail
$ns duplex-link $n9 $n1 100mb 10ms DropTail
$ns duplex-link $n10 $n1 100mb 10ms DropTail

```

```

# Set up BSD Sack TCP connection in opposite directions.
#

```

```

set src1 [$ns create-connection TCP/Sack1 $n0 TCPSink/Sack1 $d 5]
set src2 [$ns create-connection TCP/Sack1 $n2 TCPSink/Sack1 $d 6]
set src3 [$ns create-connection TCP/Sack1 $n3 TCPSink/Sack1 $d 7]
set src4 [$ns create-connection TCP/Sack1 $n4 TCPSink/Sack1 $d 8]
set src5 [$ns create-connection TCP/Sack1 $n5 TCPSink/Sack1 $d 9]

```

```

#
# Create ftp sources at the each node
#

```

```

set ftp1 [$src1 attach-app FTP]
set ftp2 [$src2 attach-app FTP]
set ftp3 [$src3 attach-app FTP]
set ftp4 [$src4 attach-app FTP]
set ftp5 [$src5 attach-app FTP]

```

```

set udp0 [new Agent/UDP]
$ns attach-agent $n6 $udp0
set cbr0 [new Application/Traffic/CBR]
$scr0 attach-agent $udp0
$scr0 set packetSize_ 1000

```

```

set udp1 [new Agent/UDP]
$ns attach-agent $n7 $udp1
set cbr1 [new Application/Traffic/CBR]
$scr1 attach-agent $udp1
$scr1 set packetSize_ 1000

```

```

set udp2 [new Agent/UDP]

```



```

$ns attach-agent $n8 $udp2
set cbr2 [new Application/Traffic/CBR]
$scr2 attach-agent $udp2
$scr2 set packetSize_ 1000

set udp3 [new Agent/UDP]
$ns attach-agent $n9 $udp3
set cbr3 [new Application/Traffic/CBR]
$scr3 attach-agent $udp3
$scr3 set packetSize_ 1000

set udp4 [new Agent/UDP]
$ns attach-agent $n10 $udp4
set cbr4 [new Application/Traffic/CBR]
$scr4 attach-agent $udp4
$scr4 set packetSize_ 1000

set null0 [new Agent/Null]
$ns attach-agent $d $null0

$ns connect $udp0 $null0
$ns connect $udp1 $null0
$ns connect $udp2 $null0
$ns connect $udp3 $null0
$ns connect $udp4 $null0
#
# Start up the first ftp at the time 0 and
# the second ftp staggered 1 second later
#
$ns at 0.0 "$ftp1 start"
$ns at 0.0 "$ftp2 start"
$ns at 0.0 "$ftp3 start"
$ns at 0.0 "$ftp4 start"
$ns at 0.0 "$ftp5 start"

$ns at 0.0 "$scr0 start"
$ns at 0.0 "$scr1 start"
$ns at 0.0 "$scr2 start"
$ns at 0.0 "$scr3 start"
$ns at 0.0 "$scr4 start"

$ns at 3.0 "finish"

#Run the simulation
$ns run

#####
# Case study 5 – TCP Rate Control
# (Modified from case study 4 – TCP Rate Control)
#####
.....

#Create nodes
set d [$ns node]
set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]
set n4 [$ns node]
set n5 [$ns node]
set n6 [$ns node]
set n7 [$ns node]
set n8 [$ns node]
set n9 [$ns node]
set n10 [$ns node]

$ns duplex-link $n1 $d 10mb 10ms DropTail
$ns duplex-link $n0 $n1 100mb 10ms DropTail
$ns duplex-link $n2 $n1 100mb 10ms DropTail
$ns duplex-link $n3 $n1 100mb 10ms DropTail
$ns duplex-link $n4 $n1 100mb 10ms DropTail
$ns duplex-link $n5 $n1 100mb 10ms DropTail

```

```

$ns duplex-link $n6 $n1 100mb 10ms DropTail
$ns duplex-link $n7 $n1 100mb 10ms DropTail
$ns duplex-link $n8 $n1 100mb 10ms DropTail
$ns duplex-link $n9 $n1 100mb 10ms DropTail
$ns duplex-link $n10 $n1 100mb 10ms DropTail

```

```

set qmon0 [$ns monitor-queue $n0 $n1 1]
set qmon01 [$ns monitor-queue $n1 $n0 1]

```

```

set qmon2 [$ns monitor-queue $n2 $n1 1]
set qmon21 [$ns monitor-queue $n1 $n2 1]

```

```

set qmon3 [$ns monitor-queue $n3 $n1 1]
set qmon31 [$ns monitor-queue $n1 $n3 1]

```

```

set qmon4 [$ns monitor-queue $n4 $n1 1]
set qmon41 [$ns monitor-queue $n1 $n4 1]

```

```

set qmon5 [$ns monitor-queue $n5 $n1 1]
set qmon51 [$ns monitor-queue $n1 $n5 1]

```

```

set qmond [$ns monitor-queue $n1 $d 1]
set qmond1 [$ns monitor-queue $d $n1 1]

```

```

# Set up BSD Sack TCP connection in opposite directions.

```

```

#
set src1 [$ns create-connection TCP/Sack1 $n0 TCPSink/Sack1 $d 5]
set src2 [$ns create-connection TCP/Sack1 $n2 TCPSink/Sack1 $d 6]
set src3 [$ns create-connection TCP/Sack1 $n3 TCPSink/Sack1 $d 7]
set src4 [$ns create-connection TCP/Sack1 $n4 TCPSink/Sack1 $d 8]
set src5 [$ns create-connection TCP/Sack1 $n5 TCPSink/Sack1 $d 9]

```

```

#
# Create ftp sources at the each node

```

```

#
set ftp1 [$src1 attach-app FTP]
set ftp2 [$src2 attach-app FTP]
set ftp3 [$src3 attach-app FTP]
set ftp4 [$src4 attach-app FTP]
set ftp5 [$src5 attach-app FTP]

```

```

set udp0 [new Agent/UDP]
$ns attach-agent $n6 $udp0
set cbr0 [new Application/Traffic/CBR]
$scr0 attach-agent $udp0
$scr0 set packetSize_ 1000

```

```

set udp1 [new Agent/UDP]
$ns attach-agent $n7 $udp1
set cbr1 [new Application/Traffic/CBR]
$scr1 attach-agent $udp1
$scr1 set packetSize_ 1000

```

```

set udp2 [new Agent/UDP]
$ns attach-agent $n8 $udp2
set cbr2 [new Application/Traffic/CBR]
$scr2 attach-agent $udp2
$scr2 set packetSize_ 1000

```

```

set udp3 [new Agent/UDP]
$ns attach-agent $n9 $udp3
set cbr3 [new Application/Traffic/CBR]
$scr3 attach-agent $udp3
$scr3 set packetSize_ 1000

```

```

set udp4 [new Agent/UDP]
$ns attach-agent $n10 $udp4
set cbr4 [new Application/Traffic/CBR]
$scr4 attach-agent $udp4
$scr4 set packetSize_ 1000

```

```

set null0 [new Agent/Null]
$ns attach-agent $d $null0

```

```

$ns connect $udp0 $null0

```

```

$ns connect $udp1 $null0
$ns connect $udp2 $null0
$ns connect $udp3 $null0
$ns connect $udp4 $null0
#
# Start up the first ftp at the time 0 and
# the second ftp staggered 1 second later
#
$ns at 0.0 "$ftp1 start"
$ns at 0.0 "$ftp2 start"
$ns at 0.0 "$ftp3 start"
$ns at 0.0 "$ftp4 start"
$ns at 0.0 "$ftp5 start"

$ns at 0.0 "$cbr0 start"
$ns at 0.0 "$cbr1 start"
$ns at 0.0 "$cbr2 start"
$ns at 0.0 "$cbr3 start"
$ns at 0.0 "$cbr4 start"

$ns at 0.0 "record1 1 1250000 1250000"
$ns at 0.0 "record2 1 1250000 1250000"
$ns at 0.0 "record3 1 1250000 1250000"
$ns at 0.0 "record4 1 1250000 1250000"
$ns at 0.0 "record5 1 1250000 1250000"

$ns at 3.0 "finish"

#Run the simulation
$ns run

#####
# Case study 6 - Standard TCP
# (Modified from case study 1 – Standard TCP)
#####

.....

#Create nodes
set d [$ns node]
set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]
set n4 [$ns node]
set n5 [$ns node]
set n6 [$ns node]
set n7 [$ns node]
set n8 [$ns node]
set n9 [$ns node]
set n10 [$ns node]
set n11 [$ns node]
set n12 [$ns node]
set n13 [$ns node]
set n14 [$ns node]
set n15 [$ns node]
set n16 [$ns node]
set n17 [$ns node]
set n18 [$ns node]
set n19 [$ns node]
set n20 [$ns node]

$ns duplex-link $n1 $d 10mb 10ms DropTail
$ns duplex-link $n0 $n1 100mb 10ms DropTail
$ns duplex-link $n2 $n1 100mb 10ms DropTail
$ns duplex-link $n3 $n1 100mb 10ms DropTail
$ns duplex-link $n4 $n1 100mb 10ms DropTail
$ns duplex-link $n5 $n1 100mb 10ms DropTail
$ns duplex-link $n6 $n1 100mb 10ms DropTail
$ns duplex-link $n7 $n1 100mb 10ms DropTail
$ns duplex-link $n8 $n1 100mb 10ms DropTail
$ns duplex-link $n9 $n1 100mb 10ms DropTail
$ns duplex-link $n10 $n1 100mb 10ms DropTail

```

```
$ns duplex-link $n1 $n1 100mb 10ms DropTail
$ns duplex-link $n12 $n1 100mb 10ms DropTail
$ns duplex-link $n13 $n1 100mb 10ms DropTail
$ns duplex-link $n14 $n1 100mb 10ms DropTail
$ns duplex-link $n15 $n1 100mb 10ms DropTail
$ns duplex-link $n16 $n1 100mb 10ms DropTail
$ns duplex-link $n17 $n1 100mb 10ms DropTail
$ns duplex-link $n18 $n1 100mb 10ms DropTail
$ns duplex-link $n19 $n1 100mb 10ms DropTail
$ns duplex-link $n20 $n1 100mb 10ms DropTail
```

```
set qmon0 [$ns monitor-queue $n0 $n1 1]
set qmon01 [$ns monitor-queue $n1 $n0 1]
```

```
set qmon2 [$ns monitor-queue $n2 $n1 1]
set qmon21 [$ns monitor-queue $n1 $n2 1]
```

```
set qmon3 [$ns monitor-queue $n3 $n1 1]
set qmon31 [$ns monitor-queue $n1 $n3 1]
```

```
set qmon4 [$ns monitor-queue $n4 $n1 1]
set qmon41 [$ns monitor-queue $n1 $n4 1]
```

```
set qmon5 [$ns monitor-queue $n5 $n1 1]
set qmon51 [$ns monitor-queue $n1 $n5 1]
```

```
set qmond [$ns monitor-queue $n1 $d 1]
set qmond1 [$ns monitor-queue $d $n1 1]
```

Set up BSD Sack TCP connection in opposite directions.

```
#
set src1 [$ns create-connection TCP/Sack1 $n0 TCPSink/Sack1 $d 5]
set src2 [$ns create-connection TCP/Sack1 $n2 TCPSink/Sack1 $d 6]
set src3 [$ns create-connection TCP/Sack1 $n3 TCPSink/Sack1 $d 7]
set src4 [$ns create-connection TCP/Sack1 $n4 TCPSink/Sack1 $d 8]
set src5 [$ns create-connection TCP/Sack1 $n5 TCPSink/Sack1 $d 9]
```

```
#
# Create ftp sources at the each node
```

```
#
set ftp1 [$src1 attach-app FTP]
set ftp2 [$src2 attach-app FTP]
set ftp3 [$src3 attach-app FTP]
set ftp4 [$src4 attach-app FTP]
set ftp5 [$src5 attach-app FTP]
```

```
set udp0 [new Agent/UDP]
$ns attach-agent $n6 $udp0
set cbr0 [new Application/Traffic/CBR]
$scr0 attach-agent $udp0
$scr0 set packetSize_ 1000
```

```
set udp1 [new Agent/UDP]
$ns attach-agent $n7 $udp1
set cbr1 [new Application/Traffic/CBR]
$scr1 attach-agent $udp1
$scr1 set packetSize_ 1000
```

```
set udp2 [new Agent/UDP]
$ns attach-agent $n8 $udp2
set cbr2 [new Application/Traffic/CBR]
$scr2 attach-agent $udp2
$scr2 set packetSize_ 1000
```

```
set udp3 [new Agent/UDP]
$ns attach-agent $n9 $udp3
set cbr3 [new Application/Traffic/CBR]
$scr3 attach-agent $udp3
$scr3 set packetSize_ 1000
```

```
set udp4 [new Agent/UDP]
$ns attach-agent $n10 $udp4
set cbr4 [new Application/Traffic/CBR]
$scr4 attach-agent $udp4
$scr4 set packetSize_ 1000
```

```

set udp5 [new Agent/UDP]
$ns attach-agent $n11 $udp5
set cbr5 [new Application/Traffic/CBR]
$scr5 attach-agent $udp5
$scr5 set packetSize_ 1000

```

```

set udp6 [new Agent/UDP]
$ns attach-agent $n12 $udp6
set cbr6 [new Application/Traffic/CBR]
$scr6 attach-agent $udp6
$scr6 set packetSize_ 1000

```

```

set udp7 [new Agent/UDP]
$ns attach-agent $n13 $udp7
set cbr7 [new Application/Traffic/CBR]
$scr7 attach-agent $udp7
$scr7 set packetSize_ 1000

```

```

set udp8 [new Agent/UDP]
$ns attach-agent $n14 $udp8
set cbr8 [new Application/Traffic/CBR]
$scr8 attach-agent $udp8
$scr8 set packetSize_ 1000

```

```

set udp9 [new Agent/UDP]
$ns attach-agent $n15 $udp9
set cbr9 [new Application/Traffic/CBR]
$scr9 attach-agent $udp9
$scr9 set packetSize_ 1000

```

```

set udp10 [new Agent/UDP]
$ns attach-agent $n16 $udp10
set cbr10 [new Application/Traffic/CBR]
$scr10 set packetSize_ 1000
$scr10 attach-agent $udp10

```

```

set udp11 [new Agent/UDP]
$ns attach-agent $n17 $udp11
set cbr11 [new Application/Traffic/CBR]
$scr11 attach-agent $udp11
$scr11 set packetSize_ 1000

```

```

set udp12 [new Agent/UDP]
$ns attach-agent $n18 $udp12
set cbr12 [new Application/Traffic/CBR]
$scr12 attach-agent $udp12
$scr12 set packetSize_ 1000

```

```

set udp13 [new Agent/UDP]
$ns attach-agent $n19 $udp13
set cbr13 [new Application/Traffic/CBR]
$scr13 attach-agent $udp13
$scr13 set packetSize_ 1000

```

```

set udp14 [new Agent/UDP]
$ns attach-agent $n20 $udp14
set cbr14 [new Application/Traffic/CBR]
$scr14 attach-agent $udp14
$scr14 set packetSize_ 1000

```

```

set null0 [new Agent/Null]
$ns attach-agent $d $null0
$ns connect $udp0 $null0
$ns connect $udp1 $null0
$ns connect $udp2 $null0
$ns connect $udp3 $null0
$ns connect $udp4 $null0
$ns connect $udp5 $null0
$ns connect $udp6 $null0
$ns connect $udp7 $null0
$ns connect $udp8 $null0
$ns connect $udp9 $null0
$ns connect $udp10 $null0
$ns connect $udp11 $null0

```

```

$ns connect $udp12 $null0
$ns connect $udp13 $null0
$ns connect $udp14 $null0

```

```

#
# Start up the first ftp at the time 0 and
# the second ftp staggered 1 second later
#

```

```

$ns at 0.0 "$ftp1 start"
$ns at 0.0 "$ftp2 start"
$ns at 0.0 "$ftp3 start"
$ns at 0.0 "$ftp4 start"
$ns at 0.0 "$ftp5 start"

```

```

$ns at 0.0 "$cbr0 start"
$ns at 0.0 "$cbr1 start"
$ns at 0.0 "$cbr2 start"
$ns at 0.0 "$cbr3 start"
$ns at 0.0 "$cbr4 start"
$ns at 0.0 "$cbr5 start"
$ns at 0.0 "$cbr6 start"
$ns at 0.0 "$cbr7 start"
$ns at 0.0 "$cbr8 start"
$ns at 0.0 "$cbr9 start"
$ns at 0.0 "$cbr10 start"
$ns at 0.0 "$cbr11 start"
$ns at 0.0 "$cbr12 start"
$ns at 0.0 "$cbr13 start"
$ns at 0.0 "$cbr14 start"

```

```

$ns at 3.0 "finish"

```

```

#Run the simulation
$ns run

```

```

#####
# Case study 6 - Standard TCP
# (Modified from case study 4 – TCP Rate Control)
#####

```

```

#Create nodes

```

```

set d [$ns node]
set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]
set n4 [$ns node]
set n5 [$ns node]
set n6 [$ns node]
set n7 [$ns node]
set n8 [$ns node]
set n9 [$ns node]
set n10 [$ns node]
set n11 [$ns node]
set n12 [$ns node]
set n13 [$ns node]
set n14 [$ns node]
set n15 [$ns node]
set n16 [$ns node]
set n17 [$ns node]
set n18 [$ns node]
set n19 [$ns node]
set n20 [$ns node]

```

```

$ns duplex-link $n1 $d 10mb 10ms DropTail
$ns duplex-link $n0 $n1 100mb 10ms DropTail
$ns duplex-link $n2 $n1 100mb 10ms DropTail
$ns duplex-link $n3 $n1 100mb 10ms DropTail
$ns duplex-link $n4 $n1 100mb 10ms DropTail
$ns duplex-link $n5 $n1 100mb 10ms DropTail

```



```

$ns duplex-link $n6 $n1 100mb 10ms DropTail
$ns duplex-link $n7 $n1 100mb 10ms DropTail
$ns duplex-link $n8 $n1 100mb 10ms DropTail
$ns duplex-link $n9 $n1 100mb 10ms DropTail
$ns duplex-link $n10 $n1 100mb 10ms DropTail
$ns duplex-link $n11 $n1 100mb 10ms DropTail
$ns duplex-link $n12 $n1 100mb 10ms DropTail
$ns duplex-link $n13 $n1 100mb 10ms DropTail
$ns duplex-link $n14 $n1 100mb 10ms DropTail
$ns duplex-link $n15 $n1 100mb 10ms DropTail
$ns duplex-link $n16 $n1 100mb 10ms DropTail
$ns duplex-link $n17 $n1 100mb 10ms DropTail
$ns duplex-link $n18 $n1 100mb 10ms DropTail
$ns duplex-link $n19 $n1 100mb 10ms DropTail
$ns duplex-link $n20 $n1 100mb 10ms DropTail

```

```

# Set up BSD Sack TCP connection in opposite directions.
#

```

```

set src1 [$ns create-connection TCP/Sack1 $n0 TCPSink/Sack1 $d 5]
set src2 [$ns create-connection TCP/Sack1 $n2 TCPSink/Sack1 $d 6]
set src3 [$ns create-connection TCP/Sack1 $n3 TCPSink/Sack1 $d 7]
set src4 [$ns create-connection TCP/Sack1 $n4 TCPSink/Sack1 $d 8]
set src5 [$ns create-connection TCP/Sack1 $n5 TCPSink/Sack1 $d 9]

```

```

#
# Create ftp sources at the each node
#

```

```

set ftp1 [$src1 attach-app FTP]
set ftp2 [$src2 attach-app FTP]
set ftp3 [$src3 attach-app FTP]
set ftp4 [$src4 attach-app FTP]
set ftp5 [$src5 attach-app FTP]

```

```

set udp0 [new Agent/UDP]
$ns attach-agent $n6 $udp0
set cbr0 [new Application/Traffic/CBR]
$scr0 attach-agent $udp0
$scr0 set packetSize_ 1000

```

```

set udp1 [new Agent/UDP]
$ns attach-agent $n7 $udp1
set cbr1 [new Application/Traffic/CBR]
$scr1 attach-agent $udp1
$scr1 set packetSize_ 1000

```

```

set udp2 [new Agent/UDP]
$ns attach-agent $n8 $udp2
set cbr2 [new Application/Traffic/CBR]
$scr2 attach-agent $udp2
$scr2 set packetSize_ 1000

```

```

set udp3 [new Agent/UDP]
$ns attach-agent $n9 $udp3
set cbr3 [new Application/Traffic/CBR]
$scr3 attach-agent $udp3
$scr3 set packetSize_ 1000

```

```

set udp4 [new Agent/UDP]
$ns attach-agent $n10 $udp4
set cbr4 [new Application/Traffic/CBR]
$scr4 attach-agent $udp4
$scr4 set packetSize_ 1000

```

```

set udp5 [new Agent/UDP]
$ns attach-agent $n11 $udp5
set cbr5 [new Application/Traffic/CBR]
$scr5 attach-agent $udp5
$scr5 set packetSize_ 1000

```

```

set udp6 [new Agent/UDP]
$ns attach-agent $n12 $udp6
set cbr6 [new Application/Traffic/CBR]
$scr6 attach-agent $udp6

```

```

$scr6 set packetSize_ 1000

set udp7 [new Agent/UDP]
$ns attach-agent $n13 $udp7
set cbr7 [new Application/Traffic/CBR]
$scr7 attach-agent $udp7
$scr7 set packetSize_ 1000

set udp8 [new Agent/UDP]
$ns attach-agent $n14 $udp8
set cbr8 [new Application/Traffic/CBR]
$scr8 attach-agent $udp8
$scr8 set packetSize_ 1000

set udp9 [new Agent/UDP]
$ns attach-agent $n15 $udp9
set cbr9 [new Application/Traffic/CBR]
$scr9 attach-agent $udp9
$scr9 set packetSize_ 1000

set udp10 [new Agent/UDP]
$ns attach-agent $n16 $udp10
set cbr10 [new Application/Traffic/CBR]
$scr10 set packetSize_ 1000
$scr10 attach-agent $udp10

set udp11 [new Agent/UDP]
$ns attach-agent $n17 $udp11
set cbr11 [new Application/Traffic/CBR]
$scr11 attach-agent $udp11
$scr11 set packetSize_ 1000

set udp12 [new Agent/UDP]
$ns attach-agent $n18 $udp12
set cbr12 [new Application/Traffic/CBR]
$scr12 attach-agent $udp12
$scr12 set packetSize_ 1000

set udp13 [new Agent/UDP]
$ns attach-agent $n19 $udp13
set cbr13 [new Application/Traffic/CBR]
$scr13 attach-agent $udp13
$scr13 set packetSize_ 1000

set udp14 [new Agent/UDP]
$ns attach-agent $n20 $udp14
set cbr14 [new Application/Traffic/CBR]
$scr14 attach-agent $udp14
$scr14 set packetSize_ 1000

set null0 [new Agent/Null]
$ns attach-agent $d $null0
$ns connect $udp0 $null0
$ns connect $udp1 $null0
$ns connect $udp2 $null0
$ns connect $udp3 $null0
$ns connect $udp4 $null0
$ns connect $udp5 $null0
$ns connect $udp6 $null0
$ns connect $udp7 $null0
$ns connect $udp8 $null0
$ns connect $udp9 $null0
$ns connect $udp10 $null0
$ns connect $udp11 $null0
$ns connect $udp12 $null0
$ns connect $udp13 $null0
$ns connect $udp14 $null0

#
# Start up the first ftp at the time 0 and
# the second ftp staggered 1 second later
#
$ns at 0.0 "$ftp1 start"
$ns at 0.0 "$ftp2 start"
$ns at 0.0 "$ftp3 start"

```

```

$ns at 0.0 "$ftp4 start"
$ns at 0.0 "$ftp5 start"

$ns at 0.0 "$cbr0 start"
$ns at 0.0 "$cbr1 start"
$ns at 0.0 "$cbr2 start"
$ns at 0.0 "$cbr3 start"
$ns at 0.0 "$cbr4 start"
$ns at 0.0 "$cbr5 start"
$ns at 0.0 "$cbr6 start"
$ns at 0.0 "$cbr7 start"
$ns at 0.0 "$cbr8 start"
$ns at 0.0 "$cbr9 start"
$ns at 0.0 "$cbr10 start"
$ns at 0.0 "$cbr11 start"
$ns at 0.0 "$cbr12 start"
$ns at 0.0 "$cbr13 start"
$ns at 0.0 "$cbr14 start"

$ns at 0.0 "record1 1 1250000 1250000"
$ns at 0.0 "record2 1 1250000 1250000"
$ns at 0.0 "record3 1 1250000 1250000"
$ns at 0.0 "record4 1 1250000 1250000"
$ns at 0.0 "record5 1 1250000 1250000"

$ns at 3.0 "finish"

#Run the simulation
$ns run

#####
# Case study 7 - Standard TCP
# (Modified from case study 7 – Standard TCP)
#####

.....

#Create nodes
set d [$ns node]
set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]
set n4 [$ns node]
set n5 [$ns node]
set n6 [$ns node]
set n7 [$ns node]
set n21 [$ns node]
set n22 [$ns node]
set n23 [$ns node]
set n24 [$ns node]
set n25 [$ns node]

$ns duplex-link $n1 $d 1mb 10ms DropTail
$ns duplex-link $n0 $n1 100mb 10ms DropTail
$ns duplex-link $n2 $n1 100mb 10ms DropTail
$ns duplex-link $n3 $n1 100mb 10ms DropTail
$ns duplex-link $n4 $n1 100mb 10ms DropTail
$ns duplex-link $n5 $n1 100mb 10ms DropTail
$ns duplex-link $n6 $n1 100mb 10ms DropTail
$ns duplex-link $n7 $n1 100mb 10ms DropTail
$ns duplex-link $n21 $n1 100mb 10ms DropTail
$ns duplex-link $n22 $n1 100mb 10ms DropTail
$ns duplex-link $n23 $n1 100mb 10ms DropTail
$ns duplex-link $n24 $n1 100mb 10ms DropTail
$ns duplex-link $n25 $n1 100mb 10ms DropTail

# Set up BSD Sack TCP connection in opposite directions.
#
set src1 [$ns create-connection TCP/Sack1 $n0 TCPSink/Sack1 $d 5]
set src2 [$ns create-connection TCP/Sack1 $n2 TCPSink/Sack1 $d 6]
set src3 [$ns create-connection TCP/Sack1 $n3 TCPSink/Sack1 $d 7]
set src4 [$ns create-connection TCP/Sack1 $n4 TCPSink/Sack1 $d 8]

```

```

set src5 [$ns create-connection TCP/Sack1 $n5 TCPSink/Sack1 $d 9]

set src21 [$ns create-connection TCP/Sack1 $n21 TCPSink/Sack1 $d 10]
set src22 [$ns create-connection TCP/Sack1 $n22 TCPSink/Sack1 $d 11]
set src23 [$ns create-connection TCP/Sack1 $n23 TCPSink/Sack1 $d 12]
set src24 [$ns create-connection TCP/Sack1 $n24 TCPSink/Sack1 $d 13]
set src25 [$ns create-connection TCP/Sack1 $n25 TCPSink/Sack1 $d 14]

#
# Create ftp sources at the each node
#
set ftp1 [$src1 attach-app FTP]
set ftp2 [$src2 attach-app FTP]
set ftp3 [$src3 attach-app FTP]
set ftp4 [$src4 attach-app FTP]
set ftp5 [$src5 attach-app FTP]

set ftp21 [$src21 attach-app FTP]
set ftp22 [$src22 attach-app FTP]
set ftp23 [$src23 attach-app FTP]
set ftp24 [$src24 attach-app FTP]
set ftp25 [$src25 attach-app FTP]

set udp0 [new Agent/UDP]
$ns attach-agent $n6 $udp0
set cbr0 [new Application/Traffic/CBR]
$scr0 attach-agent $udp0
$scr0 set packetSize_ 1000

set udp1 [new Agent/UDP]
$ns attach-agent $n7 $udp1
set cbr1 [new Application/Traffic/CBR]
$scr1 attach-agent $udp1
$scr1 set packetSize_ 1000

set null0 [new Agent/Null]
$ns attach-agent $d $null0
$ns connect $udp0 $null0
$ns connect $udp1 $null0

#
# Start up the first ftp at the time 0 and
# the second ftp staggered 1 second later
#
$ns at 0.0 "$ftp1 start"
$ns at 0.0 "$ftp2 start"
$ns at 0.0 "$ftp3 start"
$ns at 0.0 "$ftp4 start"
$ns at 0.0 "$ftp5 start"

$ns at 0.0 "$ftp21 start"
$ns at 0.0 "$ftp22 start"
$ns at 0.0 "$ftp23 start"
$ns at 0.0 "$ftp24 start"
$ns at 0.0 "$ftp25 start"

$ns at 0.0 "$scr0 start"
$ns at 0.0 "$scr1 start"

$ns at 3.0 "finish"

#Run the simulation
$ns run

#####
# Case study 7 – TCP Rate Control
# (Modified from case study 7 – Standard TCP)
#####

#Create a simulator object
set ns [new Simulator]

set f [open out.tr w]
$ns trace-all $f

```

```

#set x [open x.trace w]
set t [open out.tcp w]
set a [open out.ack w]

set t_parrival [open out.pa w]

#Open the nam trace file
set nf [open out.nam w]
$ns namtrace-all $nf

#Record procedure
proc record1 {parr1 r aa} {
    global ns rtt ai src1 sink qmond qmond1 qmon01 t_parrival aa

    set time 0.0001
    set a [$qmon01 set parrivals_]
    set b [$qmond set pdepartures_]
    set c [$qmond1 set parrivals_]

    if {$parr1 <= $a} {
        set now [$ns now]

        #hungry flow, find Ai
        if {$r >= $aa} {
            #u is flow in the link and h is bw those flows are using
            set u [expr $b-$c]
            set h [expr {$u*1000}]
            set ai [expr $aa+$h]
        }

        #bottleneck flow, find Ai
        else {
            set ri [expr {$r/2}]
            set aaa [expr {$aa/2}]
            set ai [expr $ri+$aaa]
        }

        set rtt [expr [$src1 set rtt_] ]
        set amss [expr {$ai*0.00001}]
        set window [expr [$src1 set rtt_] * $amss]
        set de [expr {$rtt/$window}]
        $ns at [expr $now "$sink set interval_ $de]
        $ns at [expr $now] "$src1 set cwnd_ $window"
        set parr1 [expr {$parr1 + $window}]
        set r [expr 1000000/$rtt]
    }

    set now [$ns now]
    $ns at [expr $now+$time] "record1 $parr1 $r $ai"
}

proc record2 {parr2 r aa} {
    global ns rtt ai src2 sink qmond qmond1 qmon21 t_parrival aa

    set time 0.0001
    set a [$qmon21 set parrivals_]
    set b [$qmond set pdepartures_]
    set c [$qmond1 set parrivals_]

    if {$parr2 <= $a} {
        set now [$ns now]

        #hungry flow, find Ai
        if {$r >= $aa} {
            #u is flow in the link and h is bw those flows are using
            set u [expr $b-$c]
            set h [expr {$u*1000}]
            set ai [expr $aa+$h]
        }

        #bottleneck flow, find Ai
        else {
            set ri [expr {$r/2}]
            set aaa [expr {$aa/2}]
            set ai [expr $ri+$aaa]
        }
    }
}

```

```

set rtt [expr [$src2 set srtt_]]
set amss [expr {$ai*0.00001}]
set window [expr [$src2 set srtt_] * $amss]
set de [expr {$rtt/$window}]
$ns at [expr $now "$sink set interval_ $de]
$ns at [expr $now] "$src2 set cwnd_ $window"
set parr2 [expr {$parr2 + $window}]
set r [expr 1000000/$rtt]
}

set now [$ns now]
$ns at [expr $now+$time] "record2 $parr2 $r $ai"
}

proc record3 {parr3 r aa} {
    global ns rtt ai src3 sink qmond qmond1 qmon31 t_parrival aa

    set time 0.0001
    set a [$qmon31 set parrivals_]
    set b [$qmond set pdepartures_]
    set c [$qmond1 set parrivals_]

    if {$parr3 <= $a} {
        set now [$ns now]

        #hungry flow, find Ai
        if {$r >= $aa} {
            #u is flow in the link and h is bw those flows are using
            set u [expr $b-$c]
            set h [expr {$u*1000}]
            set ai [expr $aa+$h]
        }

        #bottleneck flow, find Ai
        else {
            set ri [expr {$r/2}]
            set aaa [expr {$aa/2}]
            set ai [expr $ri+$aaa]
        }

        set rtt [expr [$src3 set srtt_]]
        set amss [expr {$ai*0.00001}]
        set window [expr [$src3 set srtt_] * $amss]
        set de [expr {$rtt/$window}]
        $ns at [expr $now "$sink set interval_ $de]
        $ns at [expr $now] "$src3 set cwnd_ $window"
        set parr3 [expr {$parr3 + $window}]
        set r [expr 1000000/$rtt]
    }

    set now [$ns now]
    $ns at [expr $now+$time] "record3 $parr3 $r $ai"
}

proc record4 {parr4 r aa} {
    global ns rtt ai src4 sink qmond qmond1 qmon41 t_parrival aa

    set time 0.0001
    set a [$qmon41 set parrivals_]
    set b [$qmond set pdepartures_]
    set c [$qmond1 set parrivals_]

    if {$parr4 <= $a} {
        set now [$ns now]

        #hungry flow, find Ai
        if {$r >= $aa} {
            #u is flow in the link and h is bw those flows are using
            set u [expr $b-$c]
            set h [expr {$u*2000}]
            set ai [expr $aa+$h]
        }

        #bottleneck flow, find Ai
        else {
            set ri [expr {$r/2}]

```



```

set aaa [expr {$aa/2}]
set ai [expr $ri+$aaa]
}
set rtt [expr [$src4 set srtt_]]
set amss [expr {$ai*0.00001}]
set window [expr [$src4 set srtt_] * $amss]
set de [expr {$rtt/$window}]
$ns at [expr $now "$sink set interval_ $de]
$ns at [expr $now] "$src4 set cwnd_ $window"
set parr4 [expr {$parr4 + $window}]
set r [expr 1000000/$rtt]
}

set now [$ns now]
$ns at [expr $now+$time] "record4 $parr4 $r $ai"
}

proc record5 {parr5 r aa} {
    global ns rtt ai src5 sink qmond qmond1 qmon51 t_parrival aa

    set time 0.0001
    set a [$qmon51 set parrivals_]
    set b [$qmond set pdepartures_]
    set c [$qmond1 set parrivals_]

    if {$parr5 <= $a} {
        set now [$ns now]

        #hungry flow, find Ai
        if {$r >= $aa} {
            #u is flow in the link and h is bw those flows are using
            set u [expr $b-$c]
            set h [expr {$u*1000}]
            set ai [expr $aa+$h]
        }

        #bottleneck flow, find Ai
        else {
            set ri [expr {$r/2}]
            set aaa [expr {$aa/2}]
            set ai [expr $ri+$aaa]
        }
        set rtt [expr [$src5 set srtt_]]
        set amss [expr {$ai*0.00001}]
        set window [expr [$src5 set srtt_] * $amss]
        set de [expr {$rtt/$window}]
        $ns at [expr $now "$sink set interval_ $de]
        $ns at [expr $now] "$src5 set cwnd_ $window"
        set parr5 [expr {$parr5 + $window}]
        set r [expr 1000000/$rtt]
    }

    set now [$ns now]
    $ns at [expr $now+$time] "record5 $parr5 $r $ai"
}

proc record21 {parr21 r aa} {
    global ns rtt ai src21 sink qmond qmond1 qmon211 t_parrival aa

    set time 0.0001
    set a [$qmon211 set parrivals_]
    set b [$qmond set pdepartures_]
    set c [$qmond1 set parrivals_]

    if {$parr21 <= $a} {
        set now [$ns now]

        #hungry flow, find Ai
        if {$r >= $aa} {
            #u is flow in the link and h is bw those flows are using
            set u [expr $b-$c]
            set h [expr {$u*1000}]
            set ai [expr $aa+$h]
        }
    }
}

```

```

#bottleneck flow, find Ai
else {
set ri [expr {$r/2}]
set aaa [expr {$aa/2}]
set ai [expr $ri+$aaa]
}
set rtt [expr [$src21 set srtt_]]
set amss [expr {$ai*0.00001}]
set window [expr [$src21 set srtt_] * $amss]
set de [expr {$rtt/$window}]
$ns at [expr $now "$sink set interval_ $de]
$ns at [expr $now] "$src1 set cwnd_ $window"
set parr21 [expr {$parr21 + $window}]
set r [expr 1000000/$rtt]
}

set now [$ns now]
$ns at [expr $now+$time] "record21 $parr21 $r $ai"
}

proc record22 {parr22 r aa} {
global ns rtt ai src22 sink qmond qmond1 qmon221 t_parrival aa

set time 0.0001
set a [$qmon221 set parrivals_]
set b [$qmond set pdepartures_]
set c [$qmond1 set parrivals_]

if {$parr22 <= $a} {
set now [$ns now]

#hungry flow, find Ai
if {$r >= $aa} {
#u is flow in the link and h is bw those flows are using
set u [expr $b-$c]
set h [expr {$u*1000}]
set ai [expr $aa+$h]
}

#bottleneck flow, find Ai
else {
set ri [expr {$r/2}]
set aaa [expr {$aa/2}]
set ai [expr $ri+$aaa]
}
set rtt [expr [$src22 set srtt_]]
set amss [expr {$ai*0.00001}]
set window [expr [$src22 set srtt_] * $amss]
set de [expr {$rtt/$window}]
$ns at [expr $now "$sink set interval_ $de]
$ns at [expr $now] "$src22 set cwnd_ $window"
set parr22 [expr {$parr22 + $window}]
set r [expr 1000000/$rtt]
}

set now [$ns now]
$ns at [expr $now+$time] "record22 $parr22 $r $ai"
}

proc record23 {parr23 r aa} {
global ns rtt ai src23 sink qmond qmond1 qmon231 t_parrival aa

set time 0.0001
set a [$qmon231 set parrivals_]
set b [$qmond set pdepartures_]
set c [$qmond1 set parrivals_]

if {$parr23 <= $a} {
set now [$ns now]

#hungry flow, find Ai
if {$r >= $aa} {
#u is flow in the link and h is bw those flows are using
set u [expr $b-$c]
set h [expr {$u*1000}]

```

```

set ai [expr $aa+$h]
}

#bottleneck flow, find Ai
else {
set ri [expr {$r/2}]
set aaa [expr {$aa/2}]
set ai [expr $ri+$aaa]
}
set rtt [expr [$src23 set srtt_]]
set amss [expr {$ai*0.00001}]
set window [expr [$src23 set srtt_] * $amss]
set de [expr {$rtt/$window}]
$ns at [expr $now "$sink set interval_ $de]
$ns at [expr $now] "$src23 set cwnd_ $window"
set parr23 [expr {$parr23 + $window}]
set r [expr 1000000/$rtt]
}
set now [$ns now]
$ns at [expr $now+$time] "record23 $parr23 $r $ai"
}
proc record24 {parr24 r aa} {
global ns rtt ai src24 sink qmond qmond1 qmon241 t_parrival aa

set time 0.0001
set a [$qmon241 set parrivals_]
set b [$qmond set pdepartures_]
set c [$qmond1 set parrivals_]

if {$parr24 <= $a} {
set now [$ns now]

#hungry flow, find Ai
if {$r >= $aa} {
#u is flow in the link and h is bw those flows are using
set u [expr $b-$c]
set h [expr {$u*1000}]
set ai [expr $aa+$h]
}

#bottleneck flow, find Ai
else {
set ri [expr {$r/2}]
set aaa [expr {$aa/2}]
set ai [expr $ri+$aaa]
}
set rtt [expr [$src24 set srtt_]]
set amss [expr {$ai*0.00001}]
set window [expr [$src24 set srtt_] * $amss]
set de [expr {$rtt/$window}]
$ns at [expr $now "$sink set interval_ $de]
$ns at [expr $now] "$src24 set cwnd_ $window"
set parr24 [expr {$parr24 + $window}]
set r [expr 1000000/$rtt]
}
set now [$ns now]
$ns at [expr $now+$time] "record24 $parr24 $r $ai"
}
}
proc record25 {parr25 r aa} {
global ns rtt ai src25 sink qmond qmond1 qmon251 t_parrival aa

set time 0.0001
set a [$qmon251 set parrivals_]
set b [$qmond set pdepartures_]
set c [$qmond1 set parrivals_]

if {$parr25 <= $a} {
set now [$ns now]

#hungry flow, find Ai
if {$r >= $aa} {
#u is flow in the link and h is bw those flows are using
set u [expr $b-$c]
set h [expr {$u*1000}]

```

```

set ai [expr $aa+$h]
}

#bottleneck flow, find Ai
else {
set ri [expr {$r/2}]
set aaa [expr {$aa/2}]
set ai [expr $ri+$aaa]
}
set rtt [expr {$src25 set srtt_}]
set amss [expr {$ai*0.00001}]
set window [expr [$src25 set srtt_] * $amss]
set de [expr {$rtt/$window}]
$ns at [expr $now "$sink set interval_ $de]
$ns at [expr $now] "$src25 set cwnd_ $window"
set parr25 [expr {$parr25 + $window}]
set r [expr 1000000/$rtt]
}
set now [$ns now]
$ns at [expr $now+$time] "record25 $parr25 $r $ai"
}

```

```

set qmon0 [$ns monitor-queue $n0 $n1 1]
set qmon01 [$ns monitor-queue $n0 $n0 1]

set qmon2 [$ns monitor-queue $n2 $n1 1]
set qmon21 [$ns monitor-queue $n1 $n2 1]

set qmon3 [$ns monitor-queue $n3 $n1 1]
set qmon31 [$ns monitor-queue $n1 $n3 1]

set qmon4 [$ns monitor-queue $n4 $n1 1]
set qmon41 [$ns monitor-queue $n1 $n4 1]

set qmon5 [$ns monitor-queue $n5 $n1 1]
set qmon51 [$ns monitor-queue $n1 $n5 1]

set qmon21 [$ns monitor-queue $n21 $n1 1]
set qmon211 [$ns monitor-queue $n1 $n21 1]

set qmon22 [$ns monitor-queue $n22 $n1 1]
set qmon221 [$ns monitor-queue $n1 $n22 1]

set qmon23 [$ns monitor-queue $n23 $n1 1]
set qmon231 [$ns monitor-queue $n1 $n23 1]

set qmon24 [$ns monitor-queue $n24 $n1 1]
set qmon241 [$ns monitor-queue $n1 $n24 1]

set qmon25 [$ns monitor-queue $n25 $n1 1]
set qmon251 [$ns monitor-queue $n1 $n25 1]

set qmond [$ns monitor-queue $n1 $d 1]
set qmond1 [$ns monitor-queue $d $n1 1]

```

```

$ns at 0.00 "record1 1 125000 125000"
$ns at 0.00 "record2 1 125000 125000"
$ns at 0.00 "record3 1 125000 125000"
$ns at 0.00 "record4 1 125000 125000"
$ns at 0.00 "record5 1 125000 125000"

```

```

$ns at 0.00 "record21 1 125000 125000"
$ns at 0.00 "record22 1 125000 125000"
$ns at 0.00 "record23 1 125000 125000"

```

```
$ns at 0.00 "record24 1 125000 125000"
$ns at 0.00 "record25 1 125000 125000"
```

```
$ns at 3.0 "finish"
```

```
#Run the simulation
$ns run
```

```
#####
# Case study 8 - Standard TCP
# (Modified from case study 1 – Standard TCP)
#####
```

```
#Create nodes
set d [$ns node]
set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]
set n4 [$ns node]
set n5 [$ns node]
set n6 [$ns node]
set n7 [$ns node]
```

```
$ns duplex-link $n1 $d 1mb 10ms DropTail
$ns duplex-link $n0 $n1 100mb 10ms DropTail
$ns duplex-link $n2 $n1 100mb 10ms DropTail
$ns duplex-link $n3 $n1 100mb 10ms DropTail
$ns duplex-link $n4 $n1 100mb 10ms DropTail
$ns duplex-link $n5 $n1 100mb 10ms DropTail
$ns duplex-link $n6 $n1 100mb 10ms DropTail
$ns duplex-link $n7 $n1 100mb 10ms DropTail
```

```
# Set up BSD Sack TCP connection in opposite directions.
```

```
#
set src1 [$ns create-connection TCP/Sack1 $n0 TCPSink/Sack1 $d 5]
set src2 [$ns create-connection TCP/Sack1 $n2 TCPSink/Sack1 $d 6]
set src3 [$ns create-connection TCP/Sack1 $n3 TCPSink/Sack1 $d 7]
set src4 [$ns create-connection TCP/Sack1 $n4 TCPSink/Sack1 $d 8]
set src5 [$ns create-connection TCP/Sack1 $n5 TCPSink/Sack1 $d 9]
```

```
#
# Create ftp sources at the each node
```

```
#
set ftp1 [$src1 attach-app FTP]
set ftp2 [$src2 attach-app FTP]
set ftp3 [$src3 attach-app FTP]
set ftp4 [$src4 attach-app FTP]
set ftp5 [$src5 attach-app FTP]
```

```
set udp0 [new Agent/UDP]
$ns attach-agent $n6 $udp0
set cbr0 [new Application/Traffic/CBR]
$scr0 attach-agent $udp0
$scr0 set packetSize_ 1000
```

```
set udp1 [new Agent/UDP]
$ns attach-agent $n7 $udp1
set cbr1 [new Application/Traffic/CBR]
$scr1 attach-agent $udp1
$scr1 set packetSize_ 1000
```

```
set null0 [new Agent/Null]
$ns attach-agent $d $null0
```

```
$ns connect $udp0 $null0
$ns connect $udp1 $null0
```

```
#
# Start up the first ftp at the time 0 and
# the second ftp staggered 1 second later
```

```

#
$ns at 0.0 "$ftp1 start"
$ns at 0.0 "$ftp2 start"
$ns at 0.0 "$ftp3 start"
$ns at 0.0 "$ftp4 start"
$ns at 0.0 "$ftp5 start"

$ns at 0.0 "$cbr0 start"
$ns at 0.0 "$cbr1 start"

$ns at 3.0 "finish"

#Run the simulation
$ns run

#####
# Case study 8 – TCP Rate Control
# (Modified from case study 4 – TCP Rate Control)
#####

#Create nodes
set d [$ns node]
set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]
set n4 [$ns node]
set n5 [$ns node]
set n6 [$ns node]
set n7 [$ns node]

$ns duplex-link $n1 $d 1mb 10ms DropTail
$ns duplex-link $n0 $n1 100mb 10ms DropTail
$ns duplex-link $n2 $n1 100mb 10ms DropTail
$ns duplex-link $n3 $n1 100mb 10ms DropTail
$ns duplex-link $n4 $n1 100mb 10ms DropTail
$ns duplex-link $n5 $n1 100mb 10ms DropTail
$ns duplex-link $n6 $n1 100mb 10ms DropTail
$ns duplex-link $n7 $n1 100mb 10ms DropTail

set qmon0 [$ns monitor-queue $n0 $n1 1]
set qmon01 [$ns monitor-queue $n1 $n0 1]

set qmon2 [$ns monitor-queue $n2 $n1 1]
set qmon21 [$ns monitor-queue $n1 $n2 1]

set qmon3 [$ns monitor-queue $n3 $n1 1]
set qmon31 [$ns monitor-queue $n1 $n3 1]

set qmon4 [$ns monitor-queue $n4 $n1 1]
set qmon41 [$ns monitor-queue $n1 $n4 1]

set qmon5 [$ns monitor-queue $n5 $n1 1]
set qmon51 [$ns monitor-queue $n1 $n5 1]

set qmond [$ns monitor-queue $n1 $d 1]
set qmond1 [$ns monitor-queue $d $n1 1]

# Set up BSD Sack TCP connection in opposite directions.
#
set src1 [$ns create-connection TCP/Sack1 $n0 TCPSink/Sack1 $d 5]
set src2 [$ns create-connection TCP/Sack1 $n2 TCPSink/Sack1 $d 6]
set src3 [$ns create-connection TCP/Sack1 $n3 TCPSink/Sack1 $d 7]
set src4 [$ns create-connection TCP/Sack1 $n4 TCPSink/Sack1 $d 8]
set src5 [$ns create-connection TCP/Sack1 $n5 TCPSink/Sack1 $d 9]

#
# Create ftp sources at the each node
#
set ftp1 [$src1 attach-app FTP]
set ftp2 [$src2 attach-app FTP]
set ftp3 [$src3 attach-app FTP]

```



```
set ftp4 [$src4 attach-app FTP]
set ftp5 [$src5 attach-app FTP]
```

```
set udp0 [new Agent/UDP]
$ns attach-agent $n6 $udp0
set cbr0 [new Application/Traffic/CBR]
$scr0 attach-agent $udp0
$scr0 set packetSize_ 1000
```

```
set udp1 [new Agent/UDP]
$ns attach-agent $n7 $udp1
set cbr1 [new Application/Traffic/CBR]
$scr1 attach-agent $udp1
$scr1 set packetSize_ 1000
```

```
set null0 [new Agent/Null]
$ns attach-agent $d $null0
```

```
$ns connect $udp0 $null0
$ns connect $udp1 $null0
```

```
#
# Start up the first ftp at the time 0 and
# the second ftp staggered 1 second later
#
```

```
$ns at 0.0 "$ftp1 start"
$ns at 0.0 "$ftp2 start"
$ns at 0.0 "$ftp3 start"
$ns at 0.0 "$ftp4 start"
$ns at 0.0 "$ftp5 start"
```

```
$ns at 0.0 "$scr0 start"
$ns at 0.0 "$scr1 start"
```

```
$ns at 0.0 "record1 1 125000 125000"
$ns at 0.0 "record2 1 125000 125000"
$ns at 0.0 "record3 1 125000 125000"
$ns at 0.0 "record4 1 125000 125000"
$ns at 0.0 "record5 1 125000 125000"
```

```
$ns at 3.0 "finish"
```

```
#Run the simulation
$ns run
```

```
#####
# Case study 9 - Standard TCP
# (Modified from case study 1 – Standard TCP)
#####
```

```
#Create nodes
set d [$ns node]
set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]
set n4 [$ns node]
set n5 [$ns node]
set n6 [$ns node]
set n7 [$ns node]
set n8 [$ns node]
set n9 [$ns node]
set n10 [$ns node]
set n11 [$ns node]
set n12 [$ns node]
set n13 [$ns node]
set n14 [$ns node]
set n15 [$ns node]
set s1 [$ns node]
set s2 [$ns node]
```

```

$ns duplex-link $n1 $d 10mb 10ms DropTail
$ns duplex-link $n0 $n1 100mb 10ms DropTail
$ns duplex-link $n2 $n1 100mb 10ms DropTail
$ns duplex-link $n3 $n1 100mb 10ms DropTail
$ns duplex-link $n4 $n1 100mb 10ms DropTail
$ns duplex-link $n5 $n1 100mb 10ms DropTail
$ns duplex-link $n6 $n1 100mb 10ms DropTail
$ns duplex-link $n7 $n1 100mb 10ms DropTail
$ns duplex-link $n8 $n1 100mb 10ms DropTail
$ns duplex-link $n9 $n1 100mb 10ms DropTail
$ns duplex-link $n10 $n1 100mb 10ms DropTail
$ns duplex-link $n11 $n1 100mb 10ms DropTail
$ns duplex-link $n12 $n1 100mb 10ms DropTail
$ns duplex-link $n13 $n1 100mb 10ms DropTail
$ns duplex-link $n14 $n1 100mb 10ms DropTail
$ns duplex-link $n15 $n1 100mb 10ms DropTail
$ns duplex-link $s1 $d 100mb 10ms DropTail
$ns duplex-link $s2 $d 100mb 10ms DropTail

```

```

#
# Set up BSD Sack TCP connection in opposite directions.
#
set src1 [$ns create-connection TCP/Sack1 $n0 TCPSink/Sack1 $s2 5]
set src2 [$ns create-connection TCP/Sack1 $n2 TCPSink/Sack1 $s2 6]
set src3 [$ns create-connection TCP/Sack1 $n3 TCPSink/Sack1 $s2 7]
set src4 [$ns create-connection TCP/Sack1 $n4 TCPSink/Sack1 $s2 8]
set src5 [$ns create-connection TCP/Sack1 $n5 TCPSink/Sack1 $s2 9]

```

```

#
# Create ftp sources at the each node
#
set ftp1 [$src1 attach-app FTP]
set ftp2 [$src2 attach-app FTP]
set ftp3 [$src3 attach-app FTP]
set ftp4 [$src4 attach-app FTP]
set ftp5 [$src5 attach-app FTP]

```

```

set udp0 [new Agent/UDP]
$ns attach-agent $n6 $udp0
set cbr0 [new Application/Traffic/CBR]
$scr0 attach-agent $udp0
$scr0 set packetSize_ 1000

```

```

set udp1 [new Agent/UDP]
$ns attach-agent $n7 $udp1
set cbr1 [new Application/Traffic/CBR]
$scr1 attach-agent $udp1
$scr1 set packetSize_ 1000

```

```

set udp2 [new Agent/UDP]
$ns attach-agent $n8 $udp2
set cbr2 [new Application/Traffic/CBR]
$scr2 attach-agent $udp2
$scr2 set packetSize_ 1000

```

```

set udp3 [new Agent/UDP]
$ns attach-agent $n9 $udp3
set cbr3 [new Application/Traffic/CBR]
$scr3 attach-agent $udp3
$scr3 set packetSize_ 1000

```

```

set udp4 [new Agent/UDP]
$ns attach-agent $n10 $udp4
set cbr4 [new Application/Traffic/CBR]
$scr4 attach-agent $udp4
$scr4 set packetSize_ 1000

```

```

set udp5 [new Agent/UDP]
$ns attach-agent $n11 $udp5
set cbr5 [new Application/Traffic/CBR]
$scr5 attach-agent $udp5
$scr5 set packetSize_ 1000

```

```

set udp6 [new Agent/UDP]
$ns attach-agent $n12 $udp6

```

```
set cbr6 [new Application/Traffic/CBR]
$scr6 attach-agent $udp6
$scr6 set packetSize_ 1000
```

```
set udp7 [new Agent/UDP]
$ns attach-agent $n13 $udp7
set cbr7 [new Application/Traffic/CBR]
$scr7 attach-agent $udp7
$scr7 set packetSize_ 1000
```

```
set udp8 [new Agent/UDP]
$ns attach-agent $n14 $udp8
set cbr8 [new Application/Traffic/CBR]
$scr8 attach-agent $udp8
$scr8 set packetSize_ 1000
```

```
set udp9 [new Agent/UDP]
$ns attach-agent $n15 $udp9
set cbr9 [new Application/Traffic/CBR]
$scr9 attach-agent $udp9
$scr9 set packetSize_ 1000
```

```
set null0 [new Agent/Null]
$ns attach-agent $s1 $null0
```

```
$ns connect $udp0 $null0
$ns connect $udp1 $null0
$ns connect $udp2 $null0
$ns connect $udp3 $null0
$ns connect $udp4 $null0
$ns connect $udp5 $null0
$ns connect $udp6 $null0
$ns connect $udp7 $null0
$ns connect $udp8 $null0
$ns connect $udp9 $null0
```

```
#
# Start up the first ftp at the time 0 and
# the second ftp staggered 1 second later
#
```

```
$ns at 0.0 "$ftp1 start"
$ns at 0.0 "$ftp2 start"
$ns at 0.0 "$ftp3 start"
$ns at 0.0 "$ftp4 start"
$ns at 0.0 "$ftp5 start"
$ns at 0.0 "$cbr0 start"
$ns at 0.0 "$cbr1 start"
$ns at 0.0 "$cbr2 start"
$ns at 0.0 "$cbr3 start"
$ns at 0.0 "$cbr4 start"
$ns at 0.0 "$cbr5 start"
$ns at 0.0 "$cbr6 start"
$ns at 0.0 "$cbr7 start"
$ns at 0.0 "$cbr8 start"
$ns at 0.0 "$cbr9 start"
```

```
$ns at 3.0 "finish"
```

```
#Run the simulation
$ns run
```

```
#####
# Case study 9 – TCP Rate Control
# (Modified from case study 1 – TCP Rate Control)
#####
```

```
#Create nodes
set d [$ns node]
set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]
```

```

set n4 [$ns node]
set n5 [$ns node]
set n6 [$ns node]
set n7 [$ns node]
set n8 [$ns node]
set n9 [$ns node]
set n10 [$ns node]
set n11 [$ns node]
set n12 [$ns node]
set n13 [$ns node]
set n14 [$ns node]
set n15 [$ns node]
set s1 [$ns node]
set s2 [$ns node]

$ns duplex-link $n1 $d 10mb 10ms DropTail
$ns duplex-link $n0 $n1 100mb 10ms DropTail
$ns duplex-link $n2 $n1 100mb 10ms DropTail
$ns duplex-link $n3 $n1 100mb 10ms DropTail
$ns duplex-link $n4 $n1 100mb 10ms DropTail
$ns duplex-link $n5 $n1 100mb 10ms DropTail
$ns duplex-link $n6 $n1 100mb 10ms DropTail
$ns duplex-link $n7 $n1 100mb 10ms DropTail
$ns duplex-link $n8 $n1 100mb 10ms DropTail
$ns duplex-link $n9 $n1 100mb 10ms DropTail
$ns duplex-link $n10 $n1 100mb 10ms DropTail
$ns duplex-link $n11 $n1 100mb 10ms DropTail
$ns duplex-link $n12 $n1 100mb 10ms DropTail
$ns duplex-link $n13 $n1 100mb 10ms DropTail
$ns duplex-link $n14 $n1 100mb 10ms DropTail
$ns duplex-link $n15 $n1 100mb 10ms DropTail
$ns duplex-link $s1 $d 100mb 10ms DropTail
$ns duplex-link $s2 $d 100mb 10ms DropTail

set qmon0 [$ns monitor-queue $n0 $n1 1]
set qmon01 [$ns monitor-queue $n1 $n0 1]

set qmon2 [$ns monitor-queue $n2 $n1 1]
set qmon21 [$ns monitor-queue $n1 $n2 1]

set qmon3 [$ns monitor-queue $n3 $n1 1]
set qmon31 [$ns monitor-queue $n1 $n3 1]

set qmon4 [$ns monitor-queue $n4 $n1 1]
set qmon41 [$ns monitor-queue $n1 $n4 1]

set qmon5 [$ns monitor-queue $n5 $n1 1]
set qmon51 [$ns monitor-queue $n1 $n5 1]

set qmond [$ns monitor-queue $n1 $d 1]
set qmond1 [$ns monitor-queue $d $n1 1]

#
# Set up BSD Sack TCP connection in opposite directions.
#
set src1 [$ns create-connection TCP/Sack1 $n0 TCPSink/Sack1 $s2 5]
set src2 [$ns create-connection TCP/Sack1 $n2 TCPSink/Sack1 $s2 6]
set src3 [$ns create-connection TCP/Sack1 $n3 TCPSink/Sack1 $s2 7]
set src4 [$ns create-connection TCP/Sack1 $n4 TCPSink/Sack1 $s2 8]
set src5 [$ns create-connection TCP/Sack1 $n5 TCPSink/Sack1 $s2 9]

#
# Create ftp sources at the each node
#
set ftp1 [$src1 attach-app FTP]
set ftp2 [$src2 attach-app FTP]
set ftp3 [$src3 attach-app FTP]
set ftp4 [$src4 attach-app FTP]
set ftp5 [$src5 attach-app FTP]

set udp0 [new Agent/UDP]
$ns attach-agent $n6 $udp0
set cbr0 [new Application/Traffic/CBR]
$cbr0 attach-agent $udp0
$cbr0 set packetSize_ 1000

```

```

set udp1 [new Agent/UDP]
$ns attach-agent $n7 $udp1
set cbr1 [new Application/Traffic/CBR]
$scr1 attach-agent $udp1
$scr1 set packetSize_ 1000

set udp2 [new Agent/UDP]
$ns attach-agent $n8 $udp2
set cbr2 [new Application/Traffic/CBR]
$scr2 attach-agent $udp2
$scr2 set packetSize_ 1000

set udp3 [new Agent/UDP]
$ns attach-agent $n9 $udp3
set cbr3 [new Application/Traffic/CBR]
$scr3 attach-agent $udp3
$scr3 set packetSize_ 1000

set udp4 [new Agent/UDP]
$ns attach-agent $n10 $udp4
set cbr4 [new Application/Traffic/CBR]
$scr4 attach-agent $udp4
$scr4 set packetSize_ 1000

set udp5 [new Agent/UDP]
$ns attach-agent $n11 $udp5
set cbr5 [new Application/Traffic/CBR]
$scr5 attach-agent $udp5
$scr5 set packetSize_ 1000

set udp6 [new Agent/UDP]
$ns attach-agent $n12 $udp6
set cbr6 [new Application/Traffic/CBR]
$scr6 attach-agent $udp6
$scr6 set packetSize_ 1000

set udp7 [new Agent/UDP]
$ns attach-agent $n13 $udp7
set cbr7 [new Application/Traffic/CBR]
$scr7 attach-agent $udp7
$scr7 set packetSize_ 1000

set udp8 [new Agent/UDP]
$ns attach-agent $n14 $udp8
set cbr8 [new Application/Traffic/CBR]
$scr8 attach-agent $udp8
$scr8 set packetSize_ 1000

set udp9 [new Agent/UDP]
$ns attach-agent $n15 $udp9
set cbr9 [new Application/Traffic/CBR]
$scr9 attach-agent $udp9
$scr9 set packetSize_ 1000

set null0 [new Agent/Null]
$ns attach-agent $s1 $null0

$ns connect $udp0 $null0
$ns connect $udp1 $null0
$ns connect $udp2 $null0
$ns connect $udp3 $null0
$ns connect $udp4 $null0
$ns connect $udp5 $null0
$ns connect $udp6 $null0
$ns connect $udp7 $null0
$ns connect $udp8 $null0
$ns connect $udp9 $null0
#
# Start up the first ftp at the time 0 and
# the second ftp staggered 1 second later
#
$ns at 0.0 "$ftp1 start"
$ns at 0.0 "$ftp2 start"
$ns at 0.0 "$ftp3 start"

```

```

$ns at 0.0 "$ftp4 start"
$ns at 0.0 "$ftp5 start"
$ns at 0.0 "$cbr0 start"
$ns at 0.0 "$cbr1 start"
$ns at 0.0 "$cbr2 start"
$ns at 0.0 "$cbr3 start"
$ns at 0.0 "$cbr4 start"
$ns at 0.0 "$cbr5 start"
$ns at 0.0 "$cbr6 start"
$ns at 0.0 "$cbr7 start"
$ns at 0.0 "$cbr8 start"
$ns at 0.0 "$cbr9 start"

$ns at 0.0 "record1 1 1250000 1250000"
$ns at 0.0 "record2 1 1250000 1250000"
$ns at 0.0 "record3 1 1250000 1250000"
$ns at 0.0 "record4 1 1250000 1250000"
$ns at 0.0 "record5 1 1250000 1250000"

$ns at 3.0 "finish"

#Run the simulation
$ns run

#####
# Case study 10 - Standard TCP
# (Modified from case study 1 - Standard TCP)
#####

#Create nodes
set d [$ns node]
set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]
set n4 [$ns node]
set n5 [$ns node]
set n6 [$ns node]
set n7 [$ns node]
set n8 [$ns node]
set n9 [$ns node]
set n10 [$ns node]
set s1 [$ns node]
set s2 [$ns node]

$ns duplex-link $n1 $d 10mb 10ms DropTail
$ns duplex-link $n0 $n1 100mb 10ms DropTail
$ns duplex-link $n2 $n1 100mb 10ms DropTail
$ns duplex-link $n3 $n1 100mb 10ms DropTail
$ns duplex-link $n4 $n1 100mb 10ms DropTail
$ns duplex-link $n5 $n1 100mb 10ms DropTail
$ns duplex-link $n6 $n1 100mb 10ms DropTail
$ns duplex-link $n7 $n1 100mb 10ms DropTail
$ns duplex-link $n8 $n1 100mb 10ms DropTail
$ns duplex-link $n9 $n1 100mb 10ms DropTail
$ns duplex-link $n10 $n1 100mb 10ms DropTail
$ns duplex-link $s1 $d 100mb 10ms DropTail
$ns duplex-link $s2 $d 100mb 10ms DropTail

set qmon0 [$ns monitor-queue $n0 $n1 1]
set qmon01 [$ns monitor-queue $n1 $n0 1]

set qmon2 [$ns monitor-queue $n2 $n1 1]
set qmon21 [$ns monitor-queue $n1 $n2 1]

set qmon3 [$ns monitor-queue $n3 $n1 1]
set qmon31 [$ns monitor-queue $n1 $n3 1]

set qmon4 [$ns monitor-queue $n4 $n1 1]
set qmon41 [$ns monitor-queue $n1 $n4 1]

```



```

set qmon5 [$Sns monitor-queue $n5 $n1 1]
set qmon51 [$Sns monitor-queue $n1 $n5 1]

set qmond [$Sns monitor-queue $n1 $d 1]
set qmond1 [$Sns monitor-queue $d $n1 1]

#
# Set up BSD Sack TCP connection in opposite directions.
#
set src1 [$Sns create-connection TCP/Sack1 $n0 TCPSink/Sack1 $s2 5]
set src2 [$Sns create-connection TCP/Sack1 $n2 TCPSink/Sack1 $s2 6]
set src3 [$Sns create-connection TCP/Sack1 $n3 TCPSink/Sack1 $s2 7]
set src4 [$Sns create-connection TCP/Sack1 $n4 TCPSink/Sack1 $s2 8]
set src5 [$Sns create-connection TCP/Sack1 $n5 TCPSink/Sack1 $s2 9]

#
# Create ftp sources at the each node
#
set ftp1 [$src1 attach-app FTP]
set ftp2 [$src2 attach-app FTP]
set ftp3 [$src3 attach-app FTP]
set ftp4 [$src4 attach-app FTP]
set ftp5 [$src5 attach-app FTP]

set udp0 [new Agent/UDP]
$Sns attach-agent $n6 $udp0
set cbr0 [new Application/Traffic/CBR]
$Scbr0 attach-agent $udp0
$Scbr0 set packetSize_ 1000

set udp1 [new Agent/UDP]
$Sns attach-agent $n7 $udp1
set cbr1 [new Application/Traffic/CBR]
$Scbr1 attach-agent $udp1
$Scbr1 set packetSize_ 1000

set udp2 [new Agent/UDP]
$Sns attach-agent $n8 $udp2
set cbr2 [new Application/Traffic/CBR]
$Scbr2 attach-agent $udp2
$Scbr2 set packetSize_ 1000

set udp3 [new Agent/UDP]
$Sns attach-agent $n9 $udp3
set cbr3 [new Application/Traffic/CBR]
$Scbr3 attach-agent $udp3
$Scbr3 set packetSize_ 1000

set udp4 [new Agent/UDP]
$Sns attach-agent $n10 $udp4
set cbr4 [new Application/Traffic/CBR]
$Scbr4 attach-agent $udp4
$Scbr4 set packetSize_ 1000

set null0 [new Agent/Null]
$Sns attach-agent $s1 $null0

$Sns connect $udp0 $null0
$Sns connect $udp1 $null0
$Sns connect $udp2 $null0
$Sns connect $udp3 $null0
$Sns connect $udp4 $null0
#
# Start up the first ftp at the time 0 and
# the second ftp staggered 1 second later
#
$Sns at 0.0 "$ftp1 start"
$Sns at 0.0 "$ftp2 start"
$Sns at 0.0 "$ftp3 start"
$Sns at 0.0 "$ftp4 start"
$Sns at 0.0 "$ftp5 start"
$Sns at 0.0 "$cbr0 start"
$Sns at 0.0 "$cbr1 start"
$Sns at 0.0 "$cbr2 start"
$Sns at 0.0 "$cbr3 start"

```

```
$ns at 0.0 "$cbr4 start"

$ns at 0.00 "record1 1 125000 125000"
$ns at 0.00 "record2 1 125000 125000"
$ns at 0.00 "record3 1 125000 125000"
$ns at 0.00 "record4 1 125000 125000"
$ns at 0.00 "record5 1 125000 125000"

$ns at 3.0 "finish"

#Run the simulation
$ns run

#####
# Case study 11 - Standard TCP
# (Modified from case study 9 – Standard TCP)
#####

$ns duplex-link $n1 $d 5mb 10ms DropTail

#####
# Case study 11 – TCP Rate Control
# (Modified from case study 11 – TCP Rate Control)
#####

$ns duplex-link $n1 $d 5mb 10ms DropTail

$ns at 0.00 "record1 1 625000 625000"
$ns at 0.00 "record2 1 625000 625000"
$ns at 0.00 "record3 1 625000 625000"
$ns at 0.00 "record4 1 625000 625000"
$ns at 0.00 "record5 1 625000 625000"

$ns at 3.0 "finish"

#Run the simulation
$ns run

#####
# Case study 12 - Standard TCP
# (Modified from case study 10 – Standard TCP)
#####

$ns duplex-link $n1 $d 5mb 10ms DropTail
```

```
#####
# Case study 12 – TCP Rate Control
# (Modified from case study 10 – TCP Rate Control)
#####
```

```
$ns duplex-link $n1 $d 5mb 10ms DropTail
```

```
$ns at 0.00 "record1 1 625000 625000"
$ns at 0.00 "record2 1 625000 625000"
$ns at 0.00 "record3 1 625000 625000"
$ns at 0.00 "record4 1 625000 625000"
$ns at 0.00 "record5 1 625000 625000"
```

```
$ns at 3.0 "finish"
```

```
#Run the simulation
$ns run
```

```
#####
# Case study 13 - Standard TCP
# (Modified from case study 9 – Standard TCP)
#####
```

```
$ns duplex-link $n1 $d 1mb 10ms DropTail
```

```
#####
# Case study 13 – TCP Rate Control
# (Modified from case study 9 – TCP Rate Control)
#####
```

```
$ns duplex-link $n1 $d 1mb 10ms DropTail
```

```
$ns at 0.00 "record1 1 125000 125000"
$ns at 0.00 "record2 1 125000 125000"
$ns at 0.00 "record3 1 125000 125000"
$ns at 0.00 "record4 1 125000 125000"
$ns at 0.00 "record5 1 125000 125000"
```

```
$ns at 3.0 "finish"
```

```
#Run the simulation
$ns run
```

```
#####
# Case study 14 - Standard TCP
# (Modified from case study 10 – Standard TCP)
#####
```

```
$ns duplex-link $n1 $d 1mb 10ms DropTail
```

```
#####
# Case study 14 – TCP Rate Control
# (Modified from case study 10 – TCP Rate Control)
#####
```

```
$ns duplex-link $n1 $d 1mb 10ms DropTail
```

```
$ns at 0.00 "record1 1 125000 125000"
$ns at 0.00 "record2 1 125000 125000"
$ns at 0.00 "record3 1 125000 125000"
$ns at 0.00 "record4 1 125000 125000"
$ns at 0.00 "record5 1 125000 125000"
```

```
$ns at 3.0 "finish"
```

```
#Run the simulation
$ns run
```



