INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

Bell & Howell Information and Learning
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA
800-521-0600

UMI[®]

Performance Evaluation of Explicit Congestion Notification Methods in IP Networks ECN and BECN

by

Uvaiz Ahmed

A thesis submitted to the Faculty of Graduate Studies and Research in partial
fulfillment of the requirements for the degree of

Masters of Engineering

Ottawa-Carleton Institute for Electrical and Computer Engineering

Department of Systems and Computer Engineering

Carleton University

Ottawa, Ontario, Canada.

December 15th, 1999

© Uvaiz Ahmed, 1999

National Library
of Canada

Acquisitions and
Bibliographic Services

395 Wellington Street
Ottawa ON K1A 0N4
Canada

Bibliothèque nationale
du Canada

Acquisitions et
services bibliographiques

395, rue Wellington
Ottawa ON K1A 0N4
Canada

Your file Votre référence

Our file Notre référence

The author has granted a non-exclusive licence allowing the National Library of Canada to reproduce, loan, distribute or sell copies of this thesis in microform, paper or electronic formats.

The author retains ownership of the copyright in this thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without the author's permission.

L'auteur a accordé une licence non exclusive permettant à la Bibliothèque nationale du Canada de reproduire, prêter, distribuer ou vendre des copies de cette thèse sous la forme de microfiche/film, de reproduction sur papier ou sur format électronique.

L'auteur conserve la propriété du droit d'auteur qui protège cette thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

0-612-48467-X

Canada

The undersigned recommend to the faculty of Graduate Studies and
Research acceptance of the thesis

**“Performance Evaluation of Explicit Congestion
Notification Methods in IP Networks
ECN and BECN”**

Submitted by

Uvaiz Ahmed
B.Eng.

in partial fulfillment of the requirements for the degree of Masters of
Engineering

Thesis Supervisor

Thesis Supervisor

Chair,
Department of Systems and Computer Engineering
Faculty of Engineering

Carleton University
Dec 15, 1999

Most surely in the creation of the heavens and the earth and the alternation of the night and the day, and the ships that run in the sea with that which profits men, and the water that Allah sends down from the cloud, then gives life with it to the earth after its death and spreads in it all (kinds of) animals, and the changing of the winds and the clouds made subservient between the heaven and the earth, there are signs for a people who understand. (Quran 2.164)

Abstract

Congestion is a major problem in today's Internet. Most applications on the Internet use TCP as their transport protocol. It is up to TCP at the end systems to dynamically adjust to network congestion. This mechanism has served the Internet well over the years but the need for the role of the routers (the point of attachment where congestion actually occurs) has now become obvious. There has been a considerable amount of research in recent years to look at how the router queues can be managed so as to alleviate congestion. In this thesis we look at some of the mechanisms that have been proposed. Explicit Congestion Notification (ECN), a recent proposal from Floyd et al. and Backward Explicit Congestion Notification (BECN), a proposal from Hadi et al. are the two main mechanism evaluated in this thesis. An ECN/BECN router detects incipient congestion by using the Random Early Detect (RED) queue management algorithm. Rather than dropping a packet to signal congestion as in RED the two mechanisms propose to signal congestion by marking the packet (ECN) or by sending an ICMP source quench message to the sender (BECN). Our experiments show that both ECN and BECN improve TCP throughput compared to the TCP RENO stacks. We compare the relative advantage and propose new directions for research.

Acknowledgements

Praise be to Allah the almighty for having given me the courage and the dedication for pursuing this masters and completing the thesis.

I would at the outset like to express my gratitude to my parents and my family for their support during my studies and the preparation of this thesis.

I would like to thank my supervisors Prof. I. Lambadaris and Prof. T.Kunz for their guidance and encouragement during the course of this research. It has been a great opportunity to work under them and to benefit from their extensive knowledge.

This thesis would not be possible without the help of my colleagues at Nortel especially Jamal Hadi Salim who was my mentor and my supervisor at Nortel. He along with B. Nandy and N. Seddigh were the motivating factor for this thesis. I would like to sincerely tell them "Thank you" and hope to enjoy working with them.

Finally I am grateful to Carleton University, Nortel Networks and my thesis supervisors for the financial support that has made my studies possible.

TABLE OF CONTENTS

Abstract.....	iv
Acknowledgment.....	v
Table Of Contents.....	vi
List of Figures.....	viii
List of Acronyms used.....	xii
 Chapter 1 Introduction.....	 1
1.1 Background and Motivation.....	1
1.2. Objective of the Thesis.....	3
1.3. Thesis Outline.....	5
1.4. Thesis Contribution.....	5
 Chapter 2 Congestion Control and Queue management.....	 6
2.1 Congestion Control in TCP.....	6
2.1.1. Slow Start.....	6
2.1.2 Congestion Avoidance.....	8
2.1.3. Fast Retransmit.....	10
2.1.4. Fast Recovery.....	11
2.2. Queue Management in Routers.....	12
2.2.1. Tail drop.....	12
2.2.2 Drawbacks of tail drop.....	13
2.3. Active Queue Management.....	14
2.4. Random Early Detection.....	15
2.5. Explicit Congestion Notification.....	18
2.5.1 Changes at the router.....	19
2.5.2. Changes at the TCP Host side.....	21
2.6. Backward Explicit Congestion Notification (BECN).....	24
2.6.2. Role of the Router.....	29
2.6.3. Role of the Source End System.....	30

Chapter 3 Terminology used, Testbed setup and Validation.....	32
3.1. Testbed setup.....	32
3.2. Terminology used.....	37
3.3. Validating the Implementation.....	39
3.3.1. ECN validation.....	39
3.3.2. BECN validation.....	41
3.4. RED parameter selection.....	42
CHAPTER 4 Experiments, Results and Analysis.....	46
4.1. The Experiments.....	46
4.2. Fairness.....	47
4.2.1 Fairness index.....	47
4.2.2. Experiment details.....	48
4.3. Bulk transfers.....	50
4.3.1. Experiment Details.....	51
4.3.2. ECN results for Bulk transfer.....	52
4.3.3. Observation and Analysis (ECN).....	55
4.3.4. Analysis of ECN using Time Sequence Graphs (TSG).....	57
4.3.5. BECN results for bulk transfer.....	61
4.3.6. Observation and Analysis (for BECN).....	65
4.3.7 Analysis of BECN using Time Sequence Graphs (TSG)	67
4.4. Transactional transfers.....	69
4.4.1. Experiment Details.....	70
4.4.2. Transactional Transfer results for ECN.....	72
4.4.3. Observation and Analysis (ECN).....	78
4.4.4. Transactional Transfer results for BECN.....	79
4.4.5. Observation and Analysis (BECN).....	83
4.5. Comparative Analysis of ECN and BECN.....	84
4.6. Conclusion.....	90
4.6.1. ECN and BECN.....	90
4.6.2. BECN.....	91
4.7. Future Work.....	92
References.....	93

List Of Figures

Fig 1.1:	Test Bed Setup	4
Fig 2.1:	Visualization of slow start and congestion avoidance	8
Fig 2.2:	Relationship between the average queue size and the packet drop probability in a RED router	16
Fig 2.3:	Visual Representation of ECN	23
Fig 2.4:	Pictorial representation of ECN and BECN	30
Fig 3.1:	Testbed setup	33
Fig 3.2a:	Queue size under low congestion	35
Fig 3.2b:	Queue size under moderate congestion	35
Fig 3.2c:	Queue size under high congestion	36
Fig 3.3:	Tcptrace graph showing ECN in action (server point of view	40
Fig 3.4a:	Average Queue size and physical queue size in a RED router (Macroscopic view)	43
Fig 3.4b:	Average Queue size and physical queue size in a RED router (Microscopic view	43
Fig 4.1:	Test bed for fairness	48
Fig 4.2:	Test bed depicting bulk transfer.	51
Fig 4.3a:	Plot of average throughput versus RED drop probability for ECN and NON ECN in case of low congestion	53
Fig 4.3b:	Plot of average throughput versus RED drop probability for ECN and NON ECN in case of Moderate Congestion.	54
Fig 4.3c:	Plot of average throughput versus RED drop probability for ECN and NON ECN in case of High Congestion.	54
Fig 4.4:	Percentage Improvement of ECN v/s NON ECN	55
Fig 4.4a:	A typical TSG graph showing various points of interest	57

Fig 4.5 a: Time sequence Graph for RENO for High congestion and 0.1 RED drop probability.....	59
Fig 4.5 b: Time sequence Graph for RENO for High congestion and 0.1 RED drop probability (Magnified	59
Fig 4.6a: Time sequence Graph for ECN for High congestion and 0.1 RED drop probability	59
Fig 4.6b: Time sequence Graph for ECN for High congestion and 0.1 RED drop probability(Magnified	59
Fig 4.7a: Average throughput for BECN and NON ECN v/s RED drop probabilities for Low congestion.....	61
Fig 4.7b: Average throughput for BECN and NON ECN v/s RED drop probabilities for Moderate Congestion.....	63
Fig 4.7c: Average throughput for BECN and NON ECN v/s RED drop probabilities for High congestion.....	63
Fig 4.8: Percentage Improvement of BECN v/s NON ECN.....	64
Fig 4.9: Time sequence graph for NON ECN for 0.1 RED drop probability and low congestion	68
Fig 4.10: Time sequence graph for ECN for 0.1 RED drop probability	68
Fig 4.11: Time sequence graph for BECN for 0.1 RED drop probability and low congestion	68
Fig: 4.12: Testbed for transactional transfers	71
Fig 4.13a: Average Transaction rate for ECN and NON ECN v/s RED drop probabilities for a request of 1KB and response of 5KB for Moderate Congestion.....	72
Fig 4.13b: Average Transaction rate for ECN and NON ECN v/s RED drop probabilities for a request of 1KB and response of 10KB for Moderate Congestion.....	73

Fig 4.13c: Average Transaction rate for ECN and NON ECN v/s RED drop probabilities for a request of 1KB and response of 20KB for Moderate Congestion.....	74
Fig 4.14: Percentage Improvement of ECN v/s NON ECN for transactional flows for Moderate congestion	75
Fig 4.15a: Average Transaction rate for ECN and NON ECN v/s RED drop probabilities for a request of 1KB and response of 5KB for High Congestion.....	75
Fig 4.15b: Average Transaction rate for ECN and NON ECN v/s RED drop probabilities for a request of 1KB and response of 10KB for High Congestion.....	75
Fig 4.15c: Average Transaction rate for ECN and NON ECN v/s RED drop probabilities for a request of 1KB and response of 20KB for High Congestion.....	77
Fig 4.16: Percentage Improvement of ECN v/s NON ECN for transactional flows for High congestion	77
Fig 4.17a: Average Transaction rate for BECN and NON ECN v/s RED drop probabilities for a request of 1KB and response of 5KB for Moderate Congestion.	80
Fig 4.17b: Average Transaction rate for BECN and NON ECN v/s RED drop probabilities for a request of 1KB and response of 10KB for Moderate Congestion.	80
Fig 4.17c: Average Transaction rate for BECN and NON ECN v/s RED drop probabilities for a request of 1KB and response of 10KB for Moderate Congestion.	81
Fig 4.18: Percentage Improvement of BECN v/s NON ECN for transactional flows for Moderate congestion	81
Fig 4.19a: Average Transaction rate for BECN and NON ECN v/s RED drop probabilities for a request of 1KB and response of 5KB for High Congestion.....	82
Fig 4.19b: Average Transaction rate for BECN and NON ECN v/s RED drop probabilities for a request of 1KB and response of 10KB for High Congestion.	82

Fig 4.19c: Average Transaction rate for BECN and NON ECN v/s RED drop probabilities for a request of 1KB and response of 20KB for High Congestion.	83
Fig 4.20: Percentage Improvement of BECN v/s NON ECN for transactional flows for High congestion.....	83
Fig 4.21a: Comparative analysis of RENO, ECN and BECN based on transactional rates for 5KB under moderate congestion	86
Fig 4.21b: Comparative analysis of RENO, ECN and BECN based on transactional rates for 10KB under moderate congestion	87
Fig 4.21c: Comparative analysis of RENO, ECN and BECN based on transactional rates for 20KB under moderate congestion	87
Fig 4.21d: Comparative analysis of RENO, ECN and BECN based on transactional rates for 5KB under High congestion	88
Fig 4.21e: Comparative analysis of RENO, ECN and BECN based on transactional rates for 10KB under High congestion	89
Fig 4.21f: Comparative analysis of RENO, ECN and BECN based on transactional rates for 20KB under High congestion	89

List of Acronyms Used

ACK:	Acknowledgements.
BECN:	Backward Explicit Congestion Notification.
CBQ:	Class Based Queuing.
CE	Congestion Experienced bit in IP header.
CWND	Congestion window.
CWR	Congestion Window Reduced flag in TCP Header
DUPACK:	Duplicate Acknowledgement.
ECE	ECN Echo flag in the TCP Header
ECN:	Explicit Congestion Notification.
ECT	ECN capable transport bit in IP header.
FIFO:	First In First Out discipline in a Queue.
FTP:	File Transfer Protocol
ICMP:	Internet Control Messaging Protocol.
IP:	Internet Protocol
ISQ:	ICMP source quench
LAN:	Local Area Networks
MECN:	Multilevel Explicit Congestion Notification.
NON ECN:	Flows without either ECN or BECN support (TCP RENO in this document).
RED:	Random Early Detection.
SSTHRESH	Slow start threshold
SACK:	Selective Acknowledgment (an improved algorithm for TCP)
SYN:	Synchronizing packet in TCP connection.
SYN-ACK:	Synchronization Acknowledgment for a SYN packet in TCP connection.
TCP:	Transport Control Protocol
TSG:	Time Sequence Graphs.
UDP:	User Datagram Protocol
WAN:	Wide Area Network.

Chapter 1

Introduction

1.1 Background and Motivation

This thesis deals with congestion control in the Internet and is mainly motivated by the advent of various congestion control mechanisms that have evolved in the last couple of years.

Congestion can occur when data arrives on a big pipe (a fast LAN) and gets sent out over a smaller pipe (a slower WAN). Congestion can also occur when multiple input streams arrive at a router whose output capacity is less than the sum of the inputs [1]. The traditional way that a router reacts to congestion is by dropping the incoming packets in the absence of buffer space. This is referred to as *tail drop* (explained in Section 2.2.1). This dropping of packets is an indication to the transport layer that there is congestion in the network.

TCP is the most dominant transport protocol today [3]. When congestion occurs anywhere in the network an intermediate router (which experiences congestion) starts to drop packets. TCP infers that there is congestion in the network by detecting these packet drops [2]. TCP in response to this indication of congestion invokes its end-to-end congestion control algorithms [4][5][6] to alleviate congestion. These congestion control algorithms make TCP sensitive even to a single packet loss. With the advent of web applications which use TCP as their transport protocol we have observed an increase in network traffic which leads to increased network congestion and increased packet drops and thus effects the performance of TCP drastically.

Although the end-to-end congestion avoidance schemes of TCP have served the Internet very well and have been a critical component in the robustness of the Internet, with the increase in the amount of traffic in the Internet and the increase in congestion in the Internet, it is no longer practical to rely only on these end-to-end congestion schemes to provide good service at all times. Such end-to-end congestion schemes treat the network as a black box. There is a limit to how much control can be accomplished from the edges of the network. It has been realized that there is a need for the network to complement the endpoint congestion avoidance mechanisms and complement the already existing congestion control algorithm at the transport/application layer.

A lot of research has gone into how the network can involve itself to complement these congestion control mechanisms. This is where router algorithms related to congestion come into play.

It is important to distinguish between two classes of router algorithms and to understand the differences between them. The two classes of algorithms can be classified into “queue management” algorithms and “queue scheduling” algorithms. *Queue management* deals with managing the lengths of packet queues by dropping packets when necessary or appropriate (e.g. in case of congestion) whereas *scheduling algorithms* determine which packet to send next and are used primarily for managing the allocation of bandwidth among different flows based on priority etc.

For the purpose of this thesis we will focus on queue management although the two classes of algorithm are closely related and sometimes interact with each other for providing solutions to different problems.

Random Early Detection (RED) is one of the queue management techniques that has gained popularity in recent years. It works on the principle of active queue management, the philosophy of which is to involve the routers for alleviating congestion *before* it actually occurs in the network. This is done by dropping packets probabilistically before the actual queue fills up in the router. This in turn will cause the sender TCP to backoff and hence alleviate congestion before it actually occurs.

ECN [7] and BECN [8] work in conjunction with RED and propose to provide congestion feedback to the sender without actually dropping the packets. This reduces the number of packet drops, which is expected to benefit TCP during times of congestion. A more detailed explanation of all these techniques is presented in the next chapter.

1.2. Objective of the Thesis

The objective of the thesis is to compare normal TCP RENO with ECN and BECN capable RENO stacks in an actual testbed. We have chosen to do actual implementation rather than simulation by setting up a testbed in our lab at Carleton University. All code is written for Linux based systems and we use real traffic for our experiments. We focus on bulk transfers and transactional transfers for judging the relative advantage/disadvantage of the various stacks. We use throughput, goodput, and transactional transfers (**the various terms used in the document are explained in Section 3.2**) as a metric for our results.

In the course of our experiments we vary the following parameters to analyze the effects of the different algorithms (i.e. ECN and BECN):

1. Congestion
2. RED Drop probability

3. Packet sizes

Fig 1.1 shows the testbed used to carry out our experiments. Here we present an overview of the testbed to familiarize the reader with our setup. We will revisit the testbed in Chapter 3 for a detailed analysis.

Fig 1.1: Testbed setup

Data flows from the server to the clients through the router where congestion is induced. The router has an incoming link of 10MB and an outgoing link of 1.5MB that results in congestion. We vary the congestion in the router by increasing the number of background flows in the router.

We also vary the RED drop probability (RED is introduced in Chapter 2) in the router so as to study the effects of varying this drop probability on ECN flows. For transactional transfer we vary the packet sizes requested from the server.

By varying the above parameters for different types of traffic (bulk and transactional traffic) we present our results and make conclusions that help us to decide

where the two algorithms may result in better performance. Furthermore we identify areas for further research related to our ideas and experiments.

1.3. Thesis Outline

The thesis is organized as follows:

Chapter 2 introduces the congestion control algorithm that is used in TCP and gives a brief overview of Queue Management. It deals with the drawbacks of traditional queue management and introduces the concept of Active Queue Management with special emphasis to RED, ECN and BECN. Chapter 3 deals with the specifics of the testbed that we have developed for our experiments and introduces the terminologies used in this thesis report. It also explains our implementation of the various algorithms (i.e. ECN and BECN) and the validation of our implementation. Chapter 4 outlines the experiments, presents the results and gives a detailed analysis of the results. We present our conclusions at the end for both ECN and BECN and give a few pointers to future work. A list of reference is included at the end.

1.3. Thesis Contribution

The following are the main contribution of this Thesis:

1. Detailed experimental analysis of ECN for IP networks. It was shown that TCP/IP with ECN support performs better than NON ECN TCP stack.
2. Implemented BECN in Linux and it was shown that TCP/IP with BECN support performs better than NON ECN TCP stack.

Chapter 2

2. Congestion Control and Queue management

2.1 Congestion Control in TCP

The majority of the current implementations of TCP use four intertwined algorithms called “slow start”, “congestion avoidance”, “fast recovery” and “fast retransmit” [1]. These algorithms dictate how TCP initiates a connection and responds to congestion, receipt of duplicate acknowledgements (ACK's) and timeout. In the following sections we give a brief overview of these algorithms and how they react to various states of TCP.

2.1.1. Slow Start

When TCP initiates a new connection it starts by going into slow start. The principle behind slow start is to adjust the rate at which packets are injected into the network by observing the rate at which acknowledgements are returned by the other end. To facilitate this it adds a variable to TCP called congestion window, “*cwnd*”. When TCP requires a new connection it initializes the *cwnd* to one segment¹. Upon negotiating the connection both the sender and the receiver exchange information like the segment size to be used and the receiver's advertised window that the receiver will accept etc. Each time an ACK is received from the other end the congestion window is increased by one segment.

¹ (The term segment is used to define the way TCP divides the data as a sequence of bytes for transmission and this segment usually travels across the Internet in a single IP datagram).

At any given time the sender can transmit up to the minimum between congestion window and the advertised window by the receiver. In effect, congestion window is flow control imposed by the sender, while the advertised window is flow control imposed by the receiver. The congestion window "*cwnd*" is based on the sender's assessment of perceived network congestion, and the receiver's advertised window is proportional to the amount of available buffer space at the receiver for this connection. This mechanism provides an exponential growth for the *cwnd* (approximately). Fig 2.1 illustrates the growth of the *cwnd*. Initially *cwnd* is initialized to one segment. When the acknowledgement for this segment is received the *cwnd* is increased to 2. Hence now TCP sends two packets and on receipt of the acknowledgement of these two packets *cwnd* is increased to 4. On receipt of the acknowledgement of these four packets the *cwnd* is increased to 8. This exponential growth of *cwnd* continues till there is congestion in the network. At some point congestion is experienced and an intermediate router will start to discard packets. This is an indication to the sender that the *cwnd* is too large and that there is congestion in the network. Note that TCP assumes that packet loss is due to congestion (although this is not true in the case of wireless networks but for the purpose of this thesis we are concerned with wireline networks). TCP therefore invokes its congestion avoidance algorithms.

Fig 2.1: Visualization of slow start and congestion avoidance (from Stevens, TCP/IP Illustrated Vol. 1)

2.1.2 Congestion Avoidance

Congestion avoidance is a way to deal with congestion in the network, which is indicated, to TCP, by a packet loss. Note that TCP works on a cumulative acknowledgement scheme and when it gets an out of order segment it sends a duplicate ack (DUPACK) which implies that a packet has been lost. For the sender TCP there are two indications of packet loss: a timeout and the receipt of duplicate ACK. Once the

sender realizes that a packet has been lost then it invokes the congestion avoidance algorithm, which we will describe shortly.

Although congestion avoidance and slow start are independent algorithms with different objectives, when congestion occurs TCP must slow down its transmission rate of packets into the network, and then invoke slow start to alleviate congestion. In practice both algorithms are implemented together.

Congestion avoidance and slow start require that two variables be maintained for each connection: a congestion window, *cwnd*, and a slow start threshold size, *ssthresh*.

The following algorithm shows how congestion avoidance and slow start collaborate with each other in normal TCP implementations [1].

1. When a TCP connection starts up set *cwnd* to one segment (the size of the segment is decided in the initialization phase) and *ssthresh* to a high value (usually 65535 bytes)
2. While sending data never send more than the minimum of *cwnd* and the receiver's advertised window.
3. When congestion occurs (indicated by a timeout or the reception of duplicate ACK), save one-half of the current window size (the minimum of *cwnd* and the receiver's advertised window, but at least two segments) in *ssthresh*. Additionally, if the congestion is indicated by a timeout, *cwnd* is set to one segment (i.e., slow start).
4. When the other end acknowledges new data, increase *cwnd*, but the way it increases depends on whether TCP is performing slow start or congestion avoidance.

If *cwnd* is less than or equal to *ssthresh*, TCP is in slow start; otherwise TCP is performing congestion avoidance. Slow start continues until TCP is halfway to where it was when congestion occurred (since it recorded half of the window size that caused the

problem in step 2), and then congestion avoidance takes over. Figure 2.1 illustrates the above two algorithms.

In figure 2.1 it is assumed that congestion occurred when *cwnd* had a value of 32 segments, *ssthresh* is then set to 16 segments and *cwnd* is set to 1 segment. Now slow start is performed. One segment is sent at time 0 and once an ACK for that segment is returned *cwnd* is incremented to 2. Now 2 segments are send, once the ACK for these segments are returned *cwnd* is incremented to 4 and so on till *cwnd* reaches a value of 16 at which time it is equal to *ssthresh*. Now “congestion avoidance” takes over and the *cwnd* is now incremented linearly with a maximum increase of one segment per round-trip time. In this way we can see that congestion avoidance avoids congesting the network.

2.1.3. Fast Retransmit

Fast retransmit is a modification that was proposed to the congestion avoidance algorithms, and makes use of the fact that TCP may generate an immediate acknowledgement (a duplicate ACK) when an out-of-order segment is received. Usually the sender does not know whether a duplicate ACK is caused by a lost segment or just a reordering of segments (which occurs at the receiver due to an arrival of an out of order segment).

In case that the duplicate ACK is due to a reordering of segments it is reasonable to expect that there will be one or more duplicate ACKs before the reordered segment is processed which in turn will produce a new ACK and then TCP will go on to its normal operation. In the event that there are three or more Duplicate ACKs received, it is a strong indication to TCP that a segment has been lost. Fast retransmit makes use of this

indication and performs the retransmission of the missing segment without waiting for the retransmission timer to expire. There is a performance improvement by doing fast retransmission since TCP does not have to wait for the extra time for its retransmission timer to expire and can retransmit the missing segment sooner.

2.1.4. Fast Recovery

Fast recovery is done in conjunction with fast retransmit. Once TCP performs fast retransmit of the missing segment, congestion avoidance and not slow start is performed: this action is called fast recovery.

The rationale for not performing slow start in this case is that the receipt of the duplicate ACK implies that although a previous packet has been lost, another segment has been received and thus data is still flowing between the two ends. Hence it does not make sense for TCP to go into slow start.

The fast retransmit and fast recovery algorithms are usually implemented together as follows.

1. When the third duplicate ACK in a row is received, set *ssthresh* to one-half the current congestion window, *cwnd*, but no less than two segments. Retransmit the missing segment. Set *cwnd* to *ssthresh* plus 3 times the segment size. This inflates the congestion window by the number of segments that have left the network and which the other end has cached.
2. Each time another duplicate ACK arrives; increment *cwnd* by the segment size. This inflates the congestion window for the additional segment that has left the network. Transmit a packet, if allowed by the new value of *cwnd*.

3. When the next ACK arrives that acknowledges new data, set *cwnd* to *ssthresh* (the value set in step 1). This ACK should be the acknowledgment of the retransmission from step 1, one round-trip time after the retransmission. Additionally, this ACK should acknowledge all the intermediate segments sent between the lost packet and the receipt of the first duplicate ACK. This step is congestion avoidance, since TCP is down to one-half the rate it was at when the packet was lost.

TCP congestion control and avoidance mechanisms as described above are necessary and powerful but due to the ever increasing flows in the Internet they by themselves are not sufficient to provide the best services at all times. The limitations of the amount of control that can be accomplished from the edges of the network has prompted interest among the research community to complement some sort of control from the routers. Managing the queue lengths in the router as described in the next section does this.

2.2. Queue Management in Routers

Queue management algorithms manage the length of packet queues by dropping packets when necessary or appropriate. In the following section we present the traditional queue management technique called tail drop and then introduce the notion of active queue management as a primer for RED, ECN and BECN.

2.2.1. Tail drop

Queue management algorithms traditionally manage the length of packet queues in the router by dropping packets when congestion occurs. A maximum length for each queue is configured. The router will accept packets till this maximum size is exceeded; at

which point it will drop incoming packets. New packets are accepted when buffer space allows. This technique is known as *Tail Drop*. This method has served the Internet well for years, but has several drawbacks.

2.2.2 Drawbacks of tail drop [9]

Following are some of the drawbacks of tail drop queue management

a) Undesirable Global Synchronization:

Since all arriving packets from all flows are dropped after the maximum threshold (i.e. when the buffer overflows), this interacts badly with the congestion control mechanism of TCP. A cycle is formed with a burst of drops after the maximum queue size is exceeded, followed by a period of underutilization at the router as end systems backoff. End systems then increase their windows simultaneously upto a point where a burst of drops happens again. This phenomenon is called *Global Synchronization*. It leads to poor link utilization and lower overall throughput.

b) Lock-Out:

Another problem with Tail Drop is that a single connection or a few flows could monopolize the queue space in some circumstances. For example if there is a flow that is sending a lot of data it will always fill up the queue. If at the same time another flow wants to send data and it does not have a lot of data to send then it will always find the queue full and its packets will get dropped (even though it is not contributing to the actual congestion). This results in the first flow monopolizing the queue space and this is an undesirable characteristic in a best effort network. This results in a lock out phenomenon.

c) Full Queues:

Finally, one of the major drawbacks of Tail Drop is that it allows queues to maintain a full (or, almost full) status for long periods of time, since tail drop signals congestion (via a packet drop) only when the queue has become full. It is important to reduce the steady-state queue size, and this is perhaps queue management's most important goal.

Besides tail drop there are other queue management techniques "random drop on full" and "drop front on full". In the "random drop on full" discipline a router drops a randomly selected packet from the queue when the queue is full and a new packet arrives. Under "drop front on full" [10], the router drops the packet at the front of the queue when the queue is full and a new packet arrives. Both of these solve the lockout problem, but do not solve the full-queues problem.

The solution to the full queues problem is for the routers to drop packets before a queue becomes full, so that the end nodes can respond to congestion before the buffers overflow. This proactive approach is termed active queue management and is described in detail in the following sections.

2.3. Active Queue Management

Active queue management mechanisms detect congestion *before* the queue overflows, and provide an indication of this congestion to the end nodes [7]. With active queue management the transport protocols (like TCP) do not have to rely on buffer overflow as the only indication of congestion because they get notified before serious congestion happens. By dropping packets before buffers overflow, active queue management allows routers to control when and how many packets to drop.

The next section introduces RED, an active queue management mechanism that addresses some of the problems listed above.

2.4. Random Early Detection

Random Early Detection (RED) [11] is a congestion avoidance mechanism implemented in routers which works on the basis of active queue management. RED addresses the shortcomings of Tail Drop.

A RED router signals incipient congestion to TCP by dropping packets probabilistically before the queue runs out of buffer space. This drop probability is dependent on a running average queue size to avoid any bias against bursty traffic. A RED router randomly drops arriving packets, such that the probability of dropping a packet belonging to a particular flow is approximately proportional to the flow's share of bandwidth. It achieves this by keeping a few statistics (explained later in this section) and calculating a probability on a per packet basis. Although this may present a computational problem to the router, the effect is minimal since IP packets are large (compared to, say, ATM packets) and CPU processing speed is not very expensive. Thus, if the sender is using relatively more bandwidth it gets penalized by having more of its packets dropped. RED operates by maintaining two levels of thresholds: minimum (min_{th}) and maximum (max_{th}). It drops a packet probabilistically if and only if the average queue size lies between the min_{th} and max_{th} thresholds. If the average queue size is above the maximum threshold, the arriving packet is always dropped. When the average queue size is between the minimum and the maximum threshold, each arriving packet is dropped with probability p_a , where p_a is a function of the average queue size. As

the average queue length varies between min_{th} and max_{th} , p_a increases linearly towards a configured maximum drop probability max_p . Beyond max_{th} , the drop probability is 100%. Figure 2.2 shows this relationship.

Fig 2.2: Relationship between the average queue size and the packet drop probability in a RED router

Dropping packets in this way assures that when some subset of the source TCP packets get dropped they invoke congestion avoidance algorithms that will ease the congestion at the router. Since the dropping is distributed across flows, the problem of global synchronization is avoided.

The general algorithm for RED routers is given as follows:

For each packet arrival

calculate the average queue size avg

if $min_{th} \leq avg < max_{th}$

calculate probability p_a

with probability p_a :

drop the arriving packet

else if $max_{th} \leq avg$

drop the arriving packet

The router's calculations of the average queue size take into account the period when the queue is empty (the idle period) by estimating the number m of packets that could have been transmitted by the router during the idle period. After the idle period the router computes the average queue size as if m packets had arrived to an empty queue during that period.

As avg varies from min_{th} to max_{th} , the packet-marking probability p_b varies linearly from 0 to max_p :

$$p_b = max_p (avg - min_{th}) / (max_{th} - min_{th}):$$

The final packet-marking probability p_a increases slowly as the count increases since the last marked packet:

$$p_a = p_b / (1 - count * p_b)$$

The router marks each packet (all marked packets are later dropped) that arrives at the router when the average queue size avg exceeds max_{th} .

avg : average queue size
 $count$: packets since last marked packet
 min_{th} : minimum threshold for queue
 max_{th} : maximum threshold for queue
 max_p : maximum value for p_b
 p_a : current packet-marking probability

As can be seen from the above algorithm RED tries to make sure that the queue size in the router stays between min_{th} and max_{th} . In the next section we outline ECN which is an extension to RED.

2.5. Explicit Congestion Notification

Explicit Congestion notification is an extension proposed to RED which marks a packet instead of dropping it [7]. Since ECN marks packets (rather than dropping it) before congestion actually occurs, this is very useful for flows like TCP that are sensitive to even a single packet loss. Once a packet is marked it is send to the recipient who will in turn inform the sender (in the subsequent ACK) about incipient congestion. This will in turn trigger congestion avoidance algorithm (by the sender) to alleviate congestion. The advantage of using ECN TCP over normal TCP is, not only does the flows' packet not get dropped but it also avoids congestion in the router. In this thesis we examine the benefit (if any) of adding ECN support to TCP. We aim to carry out experiments under different traffic types and study the effect of adding ECN to TCP. The experiments will be carried out under different levels of congestion using different data sizes and various RED drop probabilities.

ECN requires support from both the router as well as the end hosts, i.e. the end hosts TCP stack needs to be modified. Packet from flows that are not ECN capable will continue to be dropped by RED (as was the case before ECN).

2.5.1 Changes at the router

Router side support for ECN can be added by modifying current RED implementations. For ECN capable packets the router marks the packets rather than dropping them (if the average queue size is between min_{th} and max_{th}).

It is necessary that the router identifies that a packet is ECN capable and should only mark packets that are ECN capable. This is achieved by using two bits in the IP header.

1. The ECN Capable Transport (ECT) bit: Set by the sender end system if both end systems are ECN capable (for a unicast transport protocol). In TCP this is confirmed in the pre-negotiation during the connection setup phase (explained in Section 2.5.2).
2. Congestion Experienced (CE) bit: Packets encountering congestion are marked by the router using the Congestion Experienced (CE) bit (if the average queue size is between min_{th} and max_{th}) on their way to the receiver end system (from the sender end system), with a probability proportional to the average queue size following the procedure used in RED [9] routers.

The general algorithm for ECN capable routers is the following:

for each packet arrival

calculate the average queue size avg

if $min_{th} \leq avg < max_{th}$
calculate probability p_a

with probability p_a :

If ECT bit is set
mark the arriving packet
else
drop the arriving packet

else if $max_{th} \leq avg$

drop the arriving packet

Bits 10 and 11 in the IPV6 header are proposed respectively for the ECT and CE bits. Bits 6 and 7 of the IPV4 header DSCP field are proposed as the ECT and CE placeholders respectively. These bits are currently unused.

2.5.2. Changes at the TCP Host side

The proposal to add ECN to TCP specifies two new flags in the reserved field of the TCP header. Bit 9 in the reserved field of the TCP header is designated as the ECN-Echo (ECE) flag and Bit 8 is designated as the Congestion Window Reduced (CWR) flag.

These two bits are used both for the initializing phase in which the sender and the receiver negotiate the capability and the desire to use ECN, as well as the subsequent actions to be taken in case there is congestion experienced in the network during the established state.

The following is a brief explanation of the changes that need to be made at the host TCP stacks. Figure 2.3 describes ECN once the connection is established and both sides are ECN capable.

- a) In the connection setup phase, the source and the destination TCP's have to exchange information about their desire and/or capability to use ECN. This is done by setting both the ECN-Echo flag and the CWR flag in the SYN² packet of the initial connection phase by the sender; on receipt of this SYN packet, the receiver will set the ECN-Echo flag in the SYN-ACK response (indicating its desire to participate in ECN). Once this agreement has been reached, the sender will thereon set the ECT bit in the IP header of data packets for that flow, to indicate to the network that it is capable and willing to participate in ECN.

² (SYNchronizing segment): This is the first segment sent by the TCP protocol and is used to synchronize the two ends of a connection in preparation for opening a connection.

- b) When a router has decided from its active queue management mechanism to drop (for NON ECN flow) or mark a packet (for an ECN flow), it checks the IP-ECT bit in the packet header. It sets the CE bit in the IP header if the IP-ECT bit is set (which implies that the flow is ECN capable). When such a packet reaches the receiver, the receiver responds by setting the ECN-Echo flag (in TCP header) in the next outgoing ACK for the flow. The receiver will continue to do this in subsequent ACKs till it receives from the sender an indication that it (the sender) has responded to the congestion notification (this is explained below).
- c) Upon receipt of an ACK that has an ECN-Echo flag set, the sender triggers its congestion avoidance algorithm by halving its congestion window *cwnd* and updating its congestion window threshold value *ssthresh*. Once it has taken these appropriate steps, the sender sets the CWR bit on the next outgoing packet to tell the receiver that it has reacted to the (receiver's) notification of congestion. The receiver reacts to the CWR by halting its sending of the congestion notifications (ECE) to the sender if there is no new congestion in the network.

Note that the sender's reaction to the indication of congestion in the network (when it receives an ACK packet that has the ECN-Echo flag set) is equivalent to the Fast Retransmit/Recovery algorithm (when there is a congestion loss) in NON ECN-capable TCP i.e. the sender halves the congestion window *cwnd* and reduces the slow start threshold *ssthresh*. The Fast Retransmit/Recovery is still available for ECN capable stacks.

Fig 2.3: Visual Representation

2.6. Backward Explicit Congestion Notification (BECN).

Backward explicit congestion notification (BECN) is an alternative approach to the current ECN mechanism involving the use of an existing IP signaling mechanism, the Internet Control Messaging Protocol (ICMP) [12] Source Quench message. It is suggested by Hadi et al in their draft [8].

The use of ICMP Source Quench (ISQ) allows a basic ECN mechanism for IP, which does not require any negotiation between end systems. Congestion notification is kept at the network (IP) level. The congestion state can be reflected up to the transport layer (e.g. TCP or UDP) for appropriate action. The authors believe that the ISQ based approach reduces the reaction time to congestion in the network.

In addition, they have proposed a mechanism in which the ISQ message can include information on the severity of the congestion, allowing the end host to react accordingly so as to make maximal use of the resources and thereby maximizing network utilization. This is termed as MECN (Multilevel Explicit Congestion Notification). In MECN the ISQ message sent back to the sender will include information of the severity of congestion, and the sender, based on this information will adjust its *cwnd*. For example, if the congestion is not very severe then instead of cutting the *cwnd* by half it may cut down the window by 2/3 or 3/4. The interested reader should look into [8] for further details. In this thesis we limit ourselves to BECN and its implementation and the affects that it has on existing TCP implementations.

IP currently does not have any adhered to mechanism to notify its transport protocols of network congestion problems. ISQs have been in the past used for

congestion notification; TCP implements its own congestion control algorithm and makes inferences about network congestion as explained previously.

UDP applications are usually unresponsive and the protocols running over UDP (e.g., RTP) use their own congestion control methods if they do at all.

In the previous section the suggestion for introducing a methodology for adding explicit congestion notification to IP has been introduced. Following are some of the drawbacks that are intuitive in case of ECN.

- a) ECN is coupled to the transport layer (TCP) via the use of header information (ECE bit). To extend this proposal to other transport protocols will require changes to each of their respective headers. BECN proposes to decouple the transport level protocols and use the source quench mechanisms to notify the transport layer of congestion problems. This provides the value that all IP transport protocols (including any new ones that might be added in the future) are notified in the same manner about network congestion.

For the purpose of this thesis we shall consider the effects of BECN for TCP only, and in particular TCP mechanisms which use packet drops as indicators of congestion such as TCP-Reno and its variants.

- b) ECN requires the congestion notification to incur a round trip time (RTT) before the sender can react (since the sender has to wait for the acknowledgment of a packet before it gets notified of incipient congestion). In a path with high delay-bandwidth product this would be problematic for two reasons: i) in the scenario where the delay-bandwidth product is dominated mostly by the high bandwidth (as in high-speed networks), a large amount of traffic will pass through the

intermediate routers causing an increase in congestion level before the sender is notified. ii) in the scenario where the delay-bandwidth product is dominated mostly by the high latency/RTT (as in satellite networks), the reaction will take too long to address the congestion issue. In both cases, the efficient use of the available bandwidth is affected. BECN avoids this by having the router send the congestion notification as soon as congestion occurs in the router to the sender

- c) Because of the binary nature of the feedback (of ECN), the reaction is limited to halving the window size even if the congestion level is very low. Network resources could be more effectively utilized if the feedback was indicative of the congestion level at the overloaded point in the network. This is explained in the draft [8] under MECN.

Before explaining the actual BECN algorithms we digress here to provide a brief history of why ISQ's were traditionally not used and the arguments put forth by the proponents of BECN for the use of ISQ's.

ISQ messages, generated by a router to an end system, in the past have been considered inefficient due to the following reasons:

- Router CPU abuse while processing these extra messages
- Bandwidth consumption on the reverse path. It is suggested in [13] that the routers, if implementing ISQs, should rate limit their generation because they consume too much bandwidth in the reverse path. We have to note that the reason that ISQs consumed too much bandwidth in the reverse path was because the queues were essentially tail drop. Once the maximum queue capacity of the queue is reached all

incoming packets would be dropped and trigger an ISQ and this will increase the bandwidth consumption in the reverse path.

The authors argue that CPU time is no longer a constrained resource today and that the benefits provided by ECN outweigh the small performance hit added. Moreover, it has been shown [11] that when using RED (with cooperating end systems) less packet drops happen at the router in comparison to the traditional drop-tail algorithms used in disapproving ISQ. This implies the amount of processing needed at the router is reduced. It has been quantitatively shown in simulations that only about 1-5% of the packets are marked or dropped in a RED router under incipient congestion. Hence a faster reaction to the problem as provided by ISQ would alleviate the problem faster, resulting in even further reductions.

Using RED for BECN provides the following advantages:

- A connection is notified (by an ISQ in this case) of congestion at a rate proportional to the connection's share of the bandwidth at the congested router.
- Generation of ISQ messages will be limited to the period between when incipient congestion is detected until the source end system adjusts.

Given this scheme, which addresses congested routers sequentially on a downstream path, we can argue that the back-path, even if it is the same as the forward path, is probably not really congested since it covers the path only to the first point of congestion along that path.

In essence RED addresses both the backward path congestion problem, if the back path is the same one as the forward path, as well as the router processing concerns.

It is assumed that the participating routers are capable of RED. We would like to remind the reader that in such a router, a packet has a probability of being dropped where this probability is dependent on average queue size. Both the CE and ECT bits defined (for ECN) in [7] are maintained. During the de-multiplexing of the IP message, the values of both CE and ECT are passed to the transport layer. For packets with the ECT bit set in the IP header (i.e. when these packets are ECN capable), instead of the packet being dropped it would have the CE bit in the header set before being forwarded. These packets are marked with a given probability if the average queue size lies between the minimum and maximum thresholds as described in [7]

ICMP Source Quench messages (ISQ) are generated by the intermediate congested RED router and sent back to the source as an indication of incipient congestion whenever that router decides to mark the CE bit.

ISQs are usually not generated for a packet that has already been marked previously by another router regardless of whether that packet is contributing to some congestion; however, when the router queue level mandates that the packet be dropped then an ISQ is sent back to the source regardless of whether the packet was marked previously or not.

The source reacts at the transport protocol level by lowering its data throughput into the network. In TCP, upon identifying the flow causing the congestion, the sender reacts by halving both the congestion window and the slow start threshold value for that flow. The sender does not react to an ISQ message more than once per window. This is similar to the algorithm defined for ECN.

2.6.2. Role of the Router

A BECN capable router is assumed to have an active queue management algorithm implemented in it. If we take RED as the scheme as in our case the following is the modification that will have to be done in the router's operating software.

1. If the incoming packet causes the average queue size to go above the maximum threshold, then the packet is dropped. If the ECT bit is set (specifying that the end is BECN capable) in the IP header, send an ISQ back to the source.
2. If the incoming packet causes the average queue to go between the minimum and maximum thresholds, and if RED probability chooses this packet (to be marked), and if the ECT bit (i.e. it is BECN capable) is set and if the packet is not already marked (i.e. the CE bit is not set) then mark the packet (i.e. set the CE bit) and send back an ISQ: If the ECT bit is not set and RED chooses this packet then drop the packet

The general algorithm for BECN capable routers is as follows:

for each packet arrival

calculate the average queue size avg

if $min_{th} \leq avg < max_{th}$
 calculate probability p_n

with probability p_n :

If ECT bit is set and If the packet is not already been marked
 mark the arriving packet and send an ISQ to the sender.

else

drop the arriving packet

else if $max_{th} \leq avg$
 drop the arriving packet
 send an ISQ to the source

2.6.3. Role of the Source End System

If an ISQ message is received then the sender knows that there is network congestion. The flow causing the congestion is identified from the ICMP data. The TCP source reacts by halving both the congestion window and the slow start threshold value for that flow.

The sender does not react to ISQ's more than once per window. Upon receipt of an ISQ packet at time t , it notes the packets that are outstanding at that time (sent but not yet acked) and waits until a time when they have all been acknowledged before reacting to a new ISQ message.

Fig 2.4: Pictorial representation of ECN and BECN

Fig 2.4 is a pictorial representation of the way BECN works. Following is the sequence of events in case the router was ECN or BECN capable

1. The packet leaves the source to the destination and is first intercepted by the router R1.

2. If the router is congested than depending whether the router is ECN or BECN the following will happen

If it is ECN mark the packet

If it is BECN mark the packet and send an ICMP source quench to the sender - (2)

which in turn will react by slowing own its packet injection rate.

3. The packet is sent to the intermediate router R2- (3)

4. It reaches the end system (4) and the end system will send an ACK for the packet- (5).

If the packet was marked in case of ECN the ACK will carry the information that the flow is causing congestion. This will go back to the sender (6 and 7) and the sender, if it is ECN capable and if the ECN-echo bit is set, will back down.

In both cases (ECN and BECN) the packet is marked but in case of BECN the round trip delay (3 to 7) for the source to react to congestion is avoided.

Chapter 3

3. Terminology used, Testbed setup and Validation

This chapter focuses on the specifics of the testbed and the configuration of the components. We explain our test bed and then introduce the terminology used. We then go on to explain how we validated our implementation of both ECN and BECN in Linux. We have chosen Linux as the platform to implement the support for both congestion control algorithms (for TCP) because of the free availability of source code. The fact that the TCP/IP code has been thoroughly used and tested in Linux, by modifying this stack we were sure that the results obtained would reflect the true working of TCP.

3.1. Testbed setup

Fig. 3.1 shows the set up of the testbed. All the workstations are Pentium based and have exactly the same configuration except for the laptop. The router and the delay box are also Pentium based system, which have Linux as their operating system. The router runs Linux 2.128 and has a traffic conditioner that is based on class based queuing [14]. It uses RED as active queue management for managing its buffer space.

For the end systems we have modified the Linux 2.0.32 kernel TCP/IP stack to incorporate support of ECN and BECN. Linux 2.0.32 uses TCP RENO as its TCP/IP stack and hence we have made changes to RENO TCP. The code is available at [15]. The workstations used were Pentium II systems running at 266MHz with 128MB RAM and identical 3COM network cards. The server and the client workstations were identical DELL machines with the above configuration. The machine labeled congestion was a laptop and the others were all desktops.

The advantage of ECN/BECN becomes apparent when there is congestion in the network; hence we have used Class Based Queuing (CBQ) [14] to constrict packets from the data server from the 10 MB link to a 1.5Mbps pipe. Class based queuing is a method of link sharing in packet networks. It defines a set of algorithms for hierarchical link sharing. Hierarchical link-sharing allows multiple agencies, protocol families, or traffic types to share the bandwidth on a link in a controlled fashion. Sally Floyd and Van Jacobson proposed CBQ and it is explained in detail in [14]. CBQ has been implemented in Linux and we use the same for constricting the bandwidth at the outgoing link from the router to create congestion in the network. Data is always retrieved from the server towards the clients. Since the incoming pipe into the router (from the server) is 10Mbps and the outgoing pipe from the router is 1.5MB, this creates congestion at the exit from the router towards the clients for all flows passing through the router.

The workstations labeled ECN/BECN and NON ECN are running the same version of Linux and have exactly the same hardware configuration. As seen from the above figure, 10.0.0.3 is ECN or BECN capable, 10.0.0.5 is neither ECN nor BECN capable and the server is always ECN or BECN capable depending upon the experiments that are carried out. The server is rebooted into the appropriate kernel based on our experiments. We boot to a kernel with support for ECN for experiments involving ECN and we boot to a BECN kernel for experiments involving BECN. The server can also handle NON ECN/BECN flows using TCP RENO.

The machine labeled congestion is used to create congestion in the network. Congestion is created by starting a number of congestion flows (using FTP) in the background. Starting FTP sessions from the server to the “congestion” laptop creates congestion. The data flows from the server to the laptop via the router. Since the outgoing link from the router is constricted to 1.5 MB (compared to 10MB on the incoming link) packets that arrive experience congestion and are queued in the router. We monitor the average queue size of the router and depending on this we are able to create different levels of congestion.

Based on the average queue length we define three levels of congestion. If the average queue size is in the region of min_{th} (of the RED router) then the router is said to be experiencing low congestion. If the average queue size is in the middle of min_{th} and max_{th} then the router is said to be experiencing medium congestion and if the average queue size is near the max_{th} we say that the router is experiencing high congestion. From our initial experiments we noticed that 2 FTP flows cause low congestion, 5 FTP flows

cause medium congestion and 10 FTP flows cause High congestion. Fig 3.2 shows the average queue size and gives a pictorial understanding of the three levels of congestion.

Fig 3.2a): Queue size under Low congestion: Notice that the queue size is near min_{th} hence depicting Low Congestion

Fig 3.2a) represents the instantaneous queue size and the average queue size when there are two FTP flows in the background. As can be seen the average queue size is in the vicinity of min_{th} . Based on this observation we say that the congestion level is low.

Fig 3.2b): Queue size under Medium congestion: Notice that the queue size is in the middle of min_{th} and max_{th} hence depicting Moderate Congestion

Fig 3.2 b) shows the average queue size with 5 background flows. We notice that the average queue size is between min_{th} and max_{th} . Hence we say that the level of congestion in the router is medium.

Fig 3.2c): Queue size under High congestion: Notice that the queue size is near max_{th} hence depicting High Congestion

Fig 3.2 c) shows the state of the queue in the router when there are 10 FTP flows in the background. We observe that the average queue size is in the vicinity of max_{th} and based on this observation we say that there is high congestion in the network.

The direction of the congesting flows is the same as the direction of the competing flows and hence we made sure that the congestion flows are constricted in the same manner as the competing flows. Using a delay emulator, we introduce controlled latency in the network. The router has RED implemented in it and has the capability to support ECN/BECN flows.

The scenario that we have tried to emulate is that of a typical WAN. A LAN network, which typically runs at 10MB using Ethernet, and is connected to the Internet

via a 1.5MB T1 link. The typical kind of delay experienced in this kind of a network is 20 msec. We introduce delay using the delay emulator device which is a PC running the Nistnet [16] package on a Linux platform. Nistnet is a software package that delays all the packets that are destined from the server to the client by a fixed amount of time. To emulate the delay in our “typical” network we set the latency of the link for the experiments to be 20 msec.

3.2. Terminology used

This section presents background terminology used in the next few sections.

- **Congesting flows:** These are TCP flows that are started in the background so as to create congestion in the network. We use the laptop labeled *congestion* (shown in the testbed Figure 3.1) to introduce congesting flows. We use FTP for creating congestion. The direction of all congesting flows is such that the path that the data packets take is from the server to the client (i.e. the machine labeled *congestion*).
- **Low, Moderate and High congestion:** For the case of low congestion we start two congesting flows in the background, for moderate congestion we start five congesting flows and for the case of high congestion we start ten congesting flows in the background. Details of the various levels of congestion have been outlined in Section 3.1.
- **Competing flows:** These are the flows that we are interested in. They are ECN/BECN TCP flows originating from the server (10.1.0.6) to 10.0.0.3 or NON ECN TCP flows from the server to 10.0.0.5. These are started at the same time. They are termed competing flows because they compete for the available bandwidth in the

presence of congestion in the background. We are interested in the way that the competing flows react to congestion.

- **RED Region:** When the router average queue size is between the minimum threshold of RED (min_{th}) and the maximum threshold (max_{th}) we say that we are operating in the RED region. It is important that the queue operates in the RED region since otherwise it degenerates to FIFO and the advantages of ECN or BECN will not be experienced.
- **Drop probability/rate:** This is the RED parameter that sets the maximum probability of a packet being marked at the router. This corresponds to max_p as explained in Section 2.4. As we increase this drop probability both the number of packets that are dropped at the router (for the case of NON ECN flows) and the number of packets that are marked (for ECN/BECN flows) will increase. This drop probability is varied for different levels of congestion and our tests are repeated for varying levels of congestion with varying drop rates. The results are presented in the subsequent sections.
- **Low, Medium and High drop probability:** We use the term low probability to mean a drop probability max_p of 0.02, medium probability for 0.2 and high probability for 0.5. We also experimented with drop probabilities of 0.05, 0.1, and 0.3.
- **Goodput:** We define goodput as the effective data rate as observed by the user i.e. if we transmitted 4 data packets, with two of them being retransmitted packets, the efficiency is 50% and the resulting goodput is $2 \times \text{packet size} / \text{time taken to transmit}$. This metric is used to denote the useful data that is seen from the other end.

3.3. Validating the Implementation

Both ECN and BECN are new proposals, which involve changes at the TCP stack and the router code. We have taken an already tried and tested RENO TCP stack and router code and made the appropriate code changes for implementing ECN and BECN. Since the implementation was done in Linux we had to make sure that the implementation was in accordance to the proposed standards of ECN and BECN. In this section we present a few of the validation tests that we did. The validation done was extensive in case of ECN. Much of the validation for ECN holds true for BECN and hence there was not much needed to be done for validating BECN once we were sure that ECN was working properly.

3.3.1. ECN validation

We spent time validating that the implementation of ECN was conformant to the specification (of ECN) as described in RFC 2481. To do this, the popular tcpdump packet analyzer [17] was modified to show the packets being marked. Tcpdump is used to observe all the packets that pass through the subnet and is usually installed in one of the workstations on the subnet. For example if we are interested in the packets of the 10.0.0.X network we have tcpdump running on one of the machines in the network (say 10.0.0.1). Tcpdump can be used to store the observed packets into a dump file. Subsequently a variety of packet analyzers can be used on that file to analyze the flows that make up the dump file.

In the first phase of our validation we made sure that our implementation of ECN was conformant to the RFC of ECN. Before a connection can use ECN it is required that the two end TCP flows negotiate the desire (and capability) to use ECN. To confirm that

this was being handled in the correct manner we ran tcpdump, collected the results, and verified to see that the following were being handled properly:

- a) In the negotiation of the connection the sender sets its CWR and ECT bit in the initial SYN packet.
- b) The receiver on the other hand accepts its participation in ECN by setting its ECT bit in the SYN ACK packet.
- c) Once this has been negotiated, all the data packets from the ECN server have their ECT bit set.

In the second phase of our verification we modified tcptrace [18] which is a popular packet analyzer program in order to plot the marked packets for visualization and analysis.

Figure 3.3 shows a sample of one of the traces as seen from the server.

Fig 3.3: Tcptrace graph showing ECN in action (server point of view)

It shows a time sequence graph (TSG is covered again in Section 4.3.4) that is a plot of sequence numbers with respect to time. The “E” indicates arrival of an ECN-ECHO from the client, the “C” indicates the CWR reaction to ECN-ECHO.

Observe that the “E” stops after a “C” is issued by the server. Also observe the decrease in the slope of the sequence number plot every time a CWR is issued (indicating that the sender has slowed down).

The TCP receiver sends “E”s until it receives a “C” packet from the sender. Thus, the receiver will generally send “E”s for several successive acknowledgements (depending on *cwnd* and the number of in-flight packets).

3.3.2. BECN validation.

Validating BECN was comparatively straightforward since there is no complicated negotiation mechanism that has to be done. For all the flows that are BECN capable the ECT bit is set and the router will mark these packets in case of congestion and will send an ICMP source quench to the sender. The way in which we validated BECN was to have the router print out log messages on the screen that ICMP quenches were being generated when the packets arriving in the router finds the router in the RED region. The end system TCP stack would react by reducing its rate of injecting packets into the network and would display a log message confirming that congestion reaction had taken place. We even displayed the changes in the slow start threshold and the congestion window to make sure that the reaction that was taking place was conformant to the BECN draft.

3.4. RED parameter selection

In our initial testing we noticed that as we increase the number of congesting flows the RED queue degenerates into a simple Tail Drop queue i.e. the average queue exceeds the maximum threshold most of the times. Note that this phenomena has also been observed by [19] who proposes a dynamic solution to alleviate it by adjusting the packet dropping probability “ max_p ” based on the past history of the average queue size.

It was necessary that in the course of our experiments, the router operate in the RED region, i.e., we have to make sure that most of the time the average queue is maintained between min_{th} and max_{th} . If this is not maintained, then the queue acts like a Tail Drop queue and the advantages of ECN/BECN diminishes.

To ensure that we were operating in the RED region we monitored the average queue size and the actual queue size in times of low, moderate and high congestion and fine-tuned the RED parameters.

We made changes to the router code to output the instantaneous queue size and the average queue size at regular intervals. These values were then plotted with respect to time. We ran the experiments for varying ranges of congesting flows and different drop probabilities.

We had expected that the worst case would be when the drop probability is the lowest (i.e. 0.02) and the congestion is at the highest level (i.e. 10 congesting flows). As expected we found that this is the (worst) case. The reason behind this is because there are a lot of packets that are being forced into the queue by the congesting and the competing flows and the probability of dropping the packets is very low. Hence the queue size builds up and is maximum for our range of experiments. We noticed that the

queue size is around 75-80KB and hence we decided that if we were to maintain our max_{th} around this value we would effectively make sure that the queue does not degenerate to FIFO.

We selected the following values suitable for our range of experiments: min_{th} : 15KB, max_{th} : 75KB (assuming a buffer space for about 50 packets) , physical queue size: 100KB.

In the following figure we present our results of the instantaneous queue size and the average queue size at the router. As can be seen since in all the cases we are operating within the RED region we are sure that our results are not biased due to the router operating in the wrong region.

Fig 3.4a). Average Queue size and physical queue size in a RED router (Macroscopic View)

Fig 3.4b): Average Queue size and physical queue size in a RED router (Microscopic View)

Figure 3.4a) shows the average queue and the instantaneous queue sizes over the period of our experiments. Figure 3.4b) shows a blowup of the zoomed in part of Fig 3.4a) the area of which is depicted by the rectangle in Figure 3.4a).

The above figure shows the extreme cases in which the queue sizes vary. As can be seen from the above figure both the actual queue size and the instantaneous queue size

rarely go above the max_{th} . This ensures that our experiments are not affected by the queue size operating in the wrong region and degenerating into FIFO.

When the RED drop probability is low (0.02) and there is low congestion (2 FTP flows in the background) then not many packets are dropped and hence we are operating near the min_{th} region. This can be seen from Fig 3.4a and 3.4b (the upper two graphs). In the case that the RED drop probability is low (0.02) and there is high congestion (10 FTP flows in the background), the network experiences severe congestion as there are not a lot of packets being dropped and hence this is the scenario where the router queue may degenerate into a FIFO queue (i.e. worst case). However as can be seen from the above figures (the bottom two) by choosing the values that we have chosen for min_{th} and max_{th} we have pretty much ensured that the router operates in the RED region.

Practically in some cases the router queues under severe congestion do not operate in the RED region. There is a lot of research work that is being done to make sure that the router queues operate within the RED region. These include techniques to dynamically update the min_{th} and max_{th} etc. but these are outside the scope of this thesis. For this thesis we are interested in finding the benefits of ECN and BECN and for this we have to make sure that we are operating in the RED region. From the above results we are confident that our results will truly reflect the case when the router is operating in the RED region

CHAPTER 4:

4. Experiments, Results and Analysis

In this chapter we explain the experiments conducted for comparing the two explicit congestion notification i.e. ECN and BECN. We compare them with the normal TCP RENO stack (we refer to the TCP RENO stack without support for congestion notification as NON ECN). We then analyze the results obtained and present our conclusion.

4.1. The Experiments

We performed three sets of experiments as part of this research. The first experiment outlined in Section 4.2 (Fairness) is carried out to make sure that the type of congesting flows in the background does not bias the result for the other two sets of experiment. Section 4.2.1 introduces the fairness index [20] used for testing fairness. Section 4.2.2 explains the experiment conducted and presents the results obtained. We show that the type of background flow does not bias our result and based on this we continue with the other sets of experiments.

We used two types of traffic for comparing ECN and BECN: Bulk and transactional traffic. Section 4.3 deals with Bulk transfers and Section 4.4 deals with Transactional transfer. We carry out these experiments for comparing ECN with NON ECN and study the advantages (or disadvantages) of ECN with respect to NON ECN. We use throughput in case of bulk transfers and transactional rate in case of transactional transfer as our performance metric. The same sets of tests are then repeated to compare

BECN with NON ECN. Finally, based on the above experiments we compare ECN with BECN.

4.2. Fairness

In the course of the experiments we wanted to make sure that the choice of the type of background flow does not bias the results that we collect. Since we were introducing congestion in the background by starting TCP flows we had to make sure that whether we used ECN or NON ECN TCP flows as the congesting (background) flows, this would not effect the result obtained.

Hence we carried out some tests initially with both ECN and NON ECN flows as the background flows. We repeated the experiments for different drop probabilities and calculated a fairness index [20] that we will introduce shortly

We noticed (when there was an equal number of ECN/BECN and NON ECN flows) that the number of packets dropped for the NON ECN flows was equal to the number of packets marked for the ECN flows, showing thereby that the RED algorithm was fair to both kind of flows.

4.2.1 Fairness index

The fairness index is a performance metric described in [20] by Jain. He postulates that the network is a multi-user system, and derives a metric to see how fairly each user is treated. He defines fairness as a function of variability of throughput across users. For a given set of user throughputs $(x_1, x_2 \dots x_n)$, the fairness index to the set is defined as follows:

$$f(x_1, x_2, \dots, x_n) = \frac{(\sum_{i=1}^n x_i)^2}{n \sum_{i=1}^n x_i^2}$$

The fairness index always lies between 0 and 1. A value of 1 indicates that all flows got exactly the same throughput.

Each of the tests was carried out 10 times to gain confidence in our results. To compute the fairness index we used TCP flows by starting FTP from the server to the client to generate traffic.

4.2.2. Experiment details

The setup for this experiment is shown in Fig: 4.1. It shows the direction of flows and the different flows that are used in the experiment.

Fig 4.1. Test bed for fairness

At initial time (denoted by $t = 0$) we start 2 NON ECN FTP sessions in the background to create congestion (indicated by the arrow with the diamond tail in the figure). At time $t=20$ seconds we start two competing flows, one from the ECN workstation and the other from the NON ECN workstation (the other two flows in the figure). We note the throughput of all the flows in the network and calculate the fairness index. The experiment is carried out for various RED drop probabilities. The same procedure is repeated with the background flows as ECN/BECN. We found that the fairness index was fairly consistent in both the cases indicating that there was no bias when the background flows were either ECN or NON ECN.

Table 4.1 shows the fairness index with the background flow as ECN and NON ECN. The first column indicates the various RED drop probabilities that was varied for the experiments. The corresponding throughput for the individual flows was recorded and the fairness index was computed. The second column presents the fairness index that was calculated with the background flow as ECN and the third column shows the fairness index computed with NON ECN as the background flow.

Max RED drop Probability	Fairness with background flows ECN	Fairness with background flows NON ECN
0.02	0.996888	0.991946
0.05	0.995987	0.988286
0.1	0.985403	0.989726
0.2	0.979368	0.983342

Table 4.1: Fairness index with background flows ECN and NON ECN for various drop probabilities

As can be seen from the above table, the fairness index is consistent among different drop probabilities whether we use ECN or NON ECN flows.

With the observation that the nature of background flows does not alter the results, we proceed by using the background flows as NON ECN for the rest of the experiments.

4.3. Bulk transfers

One of the traffic types that we have chosen to test ECN/BECN is bulk transfer. Typically these transfers take place over a period of time and involve transfers of a file from one location to another. These can include downloading a file from a remote location via FTP or HTTP. In these types of transfers it is important that the file that has been downloaded is error free. The metric we chose for bulk transfer is throughput. We have used FTP for analyzing bulk transfers since it uses TCP for file transfer.

4.3.1. Experiment Details:

All TCP flows used are RENO TCP. For the case of low congestion we start 2 FTP flows (from the server to the congestion laptop) in the background at time 0. Then after 20 seconds we start the competing flows, one to the ECN machine and the second to the NON ECN machine. Figure 4.2 shows the direction of the flows.

Fig 4.2.: Test bed depicting bulk transfer.

All flows proceed from the server towards the client network. The flow to the congestion machine (indicated by the diamond tail) serves only to create and vary congestion. The competing flows are the flows that compete for the remaining available bandwidth. These are the flows that we are comparing. As mentioned the metric of comparison is throughput. The size of the file transferred is 20MB. For the case of moderate congestion we start 5 FTP flows in the background and for the case of high congestion we start 10 FTP flows in the background. The experiments are carried out till both the competing flows complete their data transfer. At completion FTP displays the throughput that the flow received. This is the data we are interested in. We record the throughput for the various levels of congestion and within each level of congestion we vary the RED drop probabilities. We plot the results obtained. It is important to note that these flows are started simultaneously and the results are collected once both the flows have completed their transfers.

Figure 4.3 and 4.4 present the plot for bulk transfers in the case of ECN and, Fig 4.7 and 4.8 present the results obtained for BECN:

4.3.2. ECN results for Bulk transfer:

In this section we present the results obtained for the ECN bulk transfer. Fig 4.3a) shows the graph plotted for average throughput as we change the RED drop probabilities. The flow that is compared is ECN and NON ECN for the case of low congestion. It can be noted from the above graph that for the same RED drop probability ECN flow gets better throughput than NON ECN. For example (from Fig. 4.3a) when the RED drop probability is 0.1 ECN achieves 448 bps (bits per second) and NON ECN achieves 344

bps. We notice that as the RED drop probability is increased the throughput of the ECN flow is better than the throughput the NON ECN flow gets.

Fig 4.3 a) Plot of average throughput versus RED probability for ECN and NON ECN in case of low congestion. For a drop probability of 0.1 ECN gets better throughput (484 bps) than NON ECN (344bps).

The reason is that as the RED probability is increased more of the NON ECN flow's packets get dropped, hence its throughput decreases. For the case of ECN flow, as the RED drop probability is increased more of its packet gets marked (compared to being dropped, as is the case with NON ECN flows). Although the ECN flow reduces its transmission rate in the network, it makes use of the fact that the NON ECN flow has gone to a timeout (due to dropped packets) and it recovers sooner and hence there is a throughput increase. Fig 4.3b) and 4.3c) shows the graph plotted for throughput versus RED drop probability for ECN and NON ECN flows for the case of medium and high congestion. As in the case of low congestion we notice that ECN achieves higher throughput compared to NON ECN.

Fig 4.3b) Plot of average throughput versus RED drop probability for ECN and NON ECN in case of Moderate Congestion.

Fig 4.3c) Plot of average throughput versus RED drop probability for ECN and NON ECN in case of High Congestion.

Fig 4.4.: Percentage Improvement of ECN v/s NON ECN

Figure 4.4 shows the percentage improvement of ECN compared to NON ECN across all ranges of congestion. It can be seen that in all cases of congestion ECN performs better than NON ECN. The relative advantage of ECN increases as the congestion increases. The next section presents a detailed analysis of ECN.

4.3.3. Observation and Analysis (ECN):

We make three key observations from the bulk transfer experiments about the performance of ECN versus NON ECN:

- a) As the congestion level increases, the relative advantage for ECN increases but the absolute advantage decreases (this is expected, since there are more flows competing for the same link resource). ECN still does better under high congestion. From Fig. 4.4 for the RED drop probability of 0.1 the relative advantage of ECN increases from 30% to 50% as the congestion level increases from low to high.

- b) Maintaining congestion levels and varying the RED drop probability (RDP) reveals that the relative advantage of ECN increases with increasing RDP. As an example from Fig. 4.4, for the case of high congestion, as we vary the drop probability from 0.02 to 0.5, the relative advantage increases from 13% to 67%.
- c) There were hardly any retransmissions for ECN flows (except the occasional packet drop in a minority of the tests for the case of high congestion and low maximum drop probability).

We analyzed tcpdump traces for NON ECN with the help of *tcptrace* and observed that there were hardly any retransmits due to timeouts¹. This means that over a long period of time (as is the case of long bulk transfers), the data driven loss recovery mechanism of the Fast Retransmit/Recovery algorithm is very effective.

The algorithm for ECN on congestion notification from ECE is the same as that for a Fast Retransmit for NON ECN. Since both are operating in the RED region, ECN barely gets any advantage over NON ECN from the signaling (packet drop Vs marking). It is clear, however, from the results that ECN flows benefit in bulk transfers.

We believe that the main advantage of ECN for bulk transfers is that less time is spent recovering (whereas NON-ECN spends time retransmitting), and timeouts are avoided altogether.

[21] has shown that even with RED deployed, TCP RENO could suffer from multiple packet drops within the same window of data, likely to lead to multiple congestion reactions or timeouts. TCP with ECN reacts only once per in-flight packets thus reducing its vulnerability to this problem.

¹ Retransmit due to timeouts are inferred by counting the number of 3 DUPACKS and subtracting them from the total recorded number of retransmits

4.3.4. Analysis of ECN using Time Sequence Graphs (TSG).

Time sequence graphs are a plot of sequence numbers versus time. It shows the activity on the connection that is being studied. It is a useful graph for doing a “postmortem” of the flow. We digress here to provide some explanation about TSG’s and then proceed to study the results of our experiment using this explanation. The Y-axis shows the sequence numbers of the connection and the X-axis shows the time that those segments were active (i.e. either send or acked). TSG can be plotted from the data collected from tcpdump by using the packet analyzer program TCPTRACE. The resulting TSG will display important information like where the retransmission occur, where the sender had zero window (cwnd), where three duplicate acks were received etc. We can also make deductions from the TSG. For example if we see that for a brief period of time the TSG graph is flat, this implies that a timeout has occurred.

Fig 4.4a): A typical TSG graph showing various points of interest

Fig 4.4a) is a snapshot of the TSG for a NON ECN flow and will help us in understanding the various terminologies used and with this background we will apply this information for analyzing our ECN and NON ECN results. The “3” in the graph indicate the places where 3 DUPACKS were received, the “R” indicate retransmissions and the “0” indicate places where the congestion window was zero and hence the sender has to wait till it gets an ACK to send the next segment (this is also intuitively indicated by a flat but short space in the TSG). The places that are flat and long indicate that a timeout has occurred.

With a brief understanding of the TSG we now proceed to analyze the TSG for the ECN flow.

We collected the results for both the ECN flow and the NON ECN (remember that these are the competing flows and they are collected simultaneously in the same experiment) flow from the bulk transfer experiments and to gain more insight to those results we plot their TSG using TCPTRACE.

Figure 4.5 is the TSG for the NON ECN flow for 0.1 RED drop probability under high congestion. Fig 4.5a) shows the entire transfer and 4.5b) shows a magnified portion of fig 4.5a). Fig 4.6 shows the TSG for ECN in the bulk transfer case for a RED drop probability of 0.1 and High congestion. Fig 4.6a) is the TSG for the entire transfer and 4.6b) is a magnified view of a portion of Fig 4.6a). (Please note that the thick line above the actual sequence number line is the information of the various events on the graphs and has been cluttered when the graph has been reduced) From the graphs we can make the following observations:

Fig 4.5 a): Time sequence Graph for RENO for High congestion and 0.1 RED drop probability

Fig 4.5 b): Time sequence Graph for RENO for High congestion and 0.1 RED drop probability (Magnified)

Fig 4.6. a): Time sequence Graph for ECN for High congestion and 0.1 RED drop probability

Fig 4.6 b): Time sequence Graph for ECN for High congestion and 0.1 RED drop probability(Magnified)

The comparative slope for ECN is higher than RENO (indicating better performance) and this is because the time taken for ECN to transfer the file is shorter than for RENO (1400 sec for RENO compared to 1000 sec for ECN). ECN TCP sends at a more uniform and higher rate than RENO. Also we can see from Fig 4.5 b) that there are a lot of timeouts that are denoted by the places in the TSG of RENO that have flat time periods, i.e. the sequence number stays constant. From Fig 4.6 b) for ECN TCP we hardly see any timeouts. Also there are a lot of retransmissions for the case of RENO (denoted by 3 and R) on the graphs. All of the above (less retransmissions and nearly no timeouts) are contributing factors for ECN performing better than NON ECN.

4.3.5. BECN results for bulk transfer:

This section presents the results obtained for BECN under bulk traffic. Figures 4.7 a), b) and c) are a plot of throughput versus RED drop probabilities for the three levels of congestion.

Fig 4.7a): Average throughput for BECN and NON ECN v/s drop probabilities for Low congestion

From the graph we can deduce that there is an improvement in the throughput for the BECN flow as compared to the NON (B)ECN flow.

For example, we notice from figure 4.7a) that for the case of low congestion at RED drop probability of 0.1 the NON ECN flow gets an average throughput of 368 bps whereas the BECN flow gets a throughput of 480 bps.

Fig 4.7b): Average throughput for BECN and NON ECN v/s RED drop probabilities for Moderate congestion. Notice in all cases BECN performs better

Fig 4.7c): Average throughput for BECN and NON ECN v/s RED drop probabilities for High congestion. Notice in all cases BECN performs better

Figures 4.7 b) and 4.7c) show the results obtained for the throughput for BECN and NON ECN for the case of Moderate and High congestion. We can see that at any given point in the graph the throughput obtained for BECN is better than that of NON ECN. This implies that in all cases BECN performs better than NON ECN.

Figure 4.8 gives us the percentage improvement of BECN versus NON ECN

Fig 4.8.: Percentage Improvement of BECN vs NON ECN

From Figure 4.8 we notice that as the congestion level increases the percentage improvement of BECN increases. There is a twofold reason for this improvement. One is due to the fact that the BECN flow does not have its packets dropped as in the case of NON ECN. The other factor which makes BECN perform better than NON ECN (and even ECN) is that as the congestion level increases the latency increases. BECN has an ISQ mechanism that helps it recover faster and hence there is a substantial improvement for the BECN flow (this is elaborated in detail in Section 4.3.6).

4.3.6. Observation and Analysis (for BECN):

The analysis for ECN presented in Section 4.3.3 is also a factor that contributes to the improvement of BECN over RENO. The same reasons that benefit ECN benefit BECN. In comparison with RENO, not many packets of BECN capable TCP flows are dropped and hence there is an immediate benefit as explained in Section 4.3.3. Furthermore BECN spends less time retransmitting and hence the overall goodput for BECN capable flow is higher than RENO.

Another important reason that contributes to BECN performing better than ECN is that the feedback loop for the sender to react to congestion is shorter for BECN. In effect a BECN capable flow will react much faster to congestion than an ECN capable flow. Consider for example a case in which we have a network in which the path that a packet takes from the sender to the receiver consists of traversing several routers. If the congestion occurs at the first intermediate router then for the case of BECN the sender will be informed about the congestion immediately and hence will take congestion avoidance steps and this will have a snowball effect and the overall network congestion reduces. Whereas in the case of ECN the sender will get informed of the congestion after the receiver receives the packet and sends an acknowledgement to the sender. Even if we consider a moderate value of 500 msec as the time taken for a packet to traverse across North America, if the packet encounters congestion at the initial router then in effect BECN will be notified of the congestion about 800 – 1000 msec earlier and this will cause BECN to perform better.

Hence, as the latency of the link increases the feedback loop for NON BECN flows (experiencing congestion) increases and as a result we believe that BECN will perform better in cases where the latency increases. This is to be experimented with and is cited in future work.

The above argument of BECN performing better than NON BECN flows as latency increases can also be seen in our experiments. Comparing Fig 4.4 with Fig 4.8 we observe that as the level of congestion increases the percentage of improvement of BECN increases. This can be explained as follows: As the level of congestion in a router increases the queue size of the router increases, which results in packets encountering more latency. As explained above if a packet is marked by the router due to RED then the BECN flow will get notified sooner (because an ISQ is immediately generated for the flow) and hence will back down. This is expected to increase overall network utilization. Also it will improve the overall throughput that BECN receives in comparison to NON BECN flows since increased latency for the NON BECN packet results in NON BECN flows getting the notification much later (since increases in latency increase the round trip time). The subsequent ACK in which an ECN flow gets informed of congestion is delayed.

4.3.7: Analysis of BECN using Time Sequence Graphs (TSG):

The following Figures 4.9, 4.10 and 4.11 represent the Time Sequence Graphs for ECN, RENO and BECN. The time taken by the various TCP variants to complete the transfer is as follows:

RENO: 410 sec

ECN: 360 sec

BECN: 320 sec.

We see that the slope of BECN is the highest, followed by ECN, and then by RENO. The higher the slope the better the performance since it takes less time for that particular flow to transfer all data. All the advantage that ECN has apply to BECN i.e. lesser retransmissions and nearly no timeouts. BECN has an additional advantage of a shorter feedback loop and hence performs better than all other TCP variants.

Fig 4.9.. Time sequence graph for NON ECN for 0.1 RED drop probability and low congestion

Fig 4.10.. Time sequence graph for ECN for 0.1 RED drop probability and low congestion

Fig 4.11.. Time sequence graph for BECN for 0.1 RED drop probability and low congestion

4.4. Transactional transfers

We model transactional transfers by sending a small request and getting a response from a server before sending the next request. To generate transactional transfer traffic we use Netperf [22] with the CRR (Connect Request Response) option. Netperf is a benchmarking tool that can be used to measure various aspects of networking performance. It can be used to evaluate the performance of data transfer and request/response performance using either TCP or UDP, and the Berkeley Sockets interface. We can specify the type of traffic that we are interested in (UDP or TCP) and the time over which such request and responses should be sent. It will try to send as many of these request and responses within the time specified and will give the average number of these request and responses that it was able to do per second. Note that the request and responses happen back to back i.e. after one “transaction” is complete the next one takes place.

As an example, let us assume that we are retrieving a small file of say 5 – 20 KB, then in effect we send a small request to the server and the server responds by sending us the file. The transaction is complete when we receive the complete file. To gain confidence in our results we run our experiment for one hour. For each test there are a few thousand of these requests and responses taking place.

Although not exactly modeling HTTP 1.0 traffic, where several concurrent sessions are opened, Netperf - CRR is nevertheless a close approximation. Since Netperf-CRR waits for one connection to complete before opening the next one (0 think time),

that single connection could be viewed as the slowest response in the set of opened concurrent sessions (in HTTP).

The transactional data sizes were selected based on [23] which indicates that the average web transaction is around 8 – 10 KB; The smaller (5KB) size was selected to guesstimate the size of transactional processing that will become prevalent with policy management schemes in the diffserv [24] context.

Using Netperf we are able to initiate these kind of transactional transfers for a variable length of time. The main metric of interest in this case is the transaction rate, which is recorded by Netperf.

Transaction rate: the number of requests and responses for a particular requested size that we are able to execute per second. For example if our request is of 1KB and the response is 5KB then we define the transaction rate as the number of such complete transactions that we can accomplish per second.

4.4.1. Experimental Details

Similar to the case of bulk transfers we start the background FTP flows to introduce congestion in the network at time 0. For the case of transaction transfers we have taken only two levels of congestion i.e. Moderate (which is starting 5 FTP flows in the background) and High (i.e. starting 10 FTP sessions in the background). The reason that we limit ourselves to only two levels of congestion and do not carry out the experiments for low congestion is that, although ECN (or BECN) do show improvement at all levels, the effect is more profound when the congestion is high. Since we are interested in analyzing the data to achieve some conclusive results we ignore low congestion for this set of experiment. About 20 seconds later we start the transactional

transfers (using NON ECN in the first set of experiments and then in the second and the third test we use ECN and BECN) and run each test for three minutes. Fig 4.12 shows the setup of the test bed along with the direction of flows.

Fig: 4.12: Testbed for transactional transfers

We record the transactions per second that are completed for the various TCP implementations (i.e. NON ECN, ECN and BECN). We repeat the test for an hour and plot the various transactions per second, averaged out over the runs. The experiment is repeated for various maximum drop probabilities, file sizes and various levels of congestion. Figures 4.13 through 4.16 present the results obtained for ECN transactional transfers (compared to NON ECN) and Figures 4.17 through 4.20 present the results that were obtained for BECN transactional transfers (compared to NON ECN).

4.4.2. Transactional Transfer results for ECN

Figure 4.13a) shows the plot of the transaction rate for the request of file size 5KB under moderate congestion. We notice that ECN flows achieves a better transaction rate than NON ECN flows. From Figure 4.13a) the transaction rate that ECN gets for a RED drop probability of 0.1 is 1.93 as compared to 1.34 for NON ECN. We also notice that as the RED drop probability increases the transaction rate for ECN increases. The transaction rate of NON ECN is fairly constant.

Fig 4.13a).: Average Transaction rate for ECN and NON ECN v/s RED drop probabilities for a request of 1KB and response of 5KB for Moderate Congestion.

Fig 4.13b).: Average Transaction rate for ECN and NON ECN v/s RED drop probabilities for a request of 1KB and response of 10KB for Moderate Congestion.

Fig 4.13c).: Average Transaction rate for ECN and NON ECN v/s RED drop probabilities for a request of 1KB and response of 20KB for Moderate Congestion.

Figures 4.13b) and 4.13c) show the results obtained for file sizes of 10KB and 20KB respectively under moderate congestion. We can see that in all cases ECN does better than NON ECN.

Figure 4.14 shows the percentage improvement of ECN over NON ECN for the various file sizes (for moderate congestion) as we change the RED drop probabilities.

Fig 4.14.: Percentage Improvement of ECN v/s NON ECN for transactional flows for Moderate congestion

From the Figure 4.14 we notice that the smaller the file size, the higher is the percentage increase in transaction rate. For example for the RED drop probability of 0.2 the percentage increase in transaction rate for a 5KB flow is 62% whereas for the 20KB flow it is 29%. This is expected due to the following reason: if the size of the files is small then NON ECN will go into timeout and cannot benefit from the fast retransmit mechanism of TCP. For bigger file sizes this is not the case (this is explained in detail in section 4.4.3).

The following figures i.e. 4.15 and 4.16 represent the results obtained for transactional transfers under High congestion for ECN.

Fig 4.15a).: Average Transaction rate for ECN and NON ECN v/s RED drop probabilities for a request of 1KB and response of 5KB for High Congestion.

Fig 4.15b).: Average Transaction rate for ECN and NON ECN v/s RED drop probabilities for a request of 1KB and response of 10KB for High Congestion.

Fig 4.15c): Average Transaction rate for ECN and NON ECN v/s RED drop

Fig 4.16: Percentage Improvement of ECN v/s NON ECN for transactional flows for High congestion

4.4.3. Observation and Analysis (ECN)

We can make three key observations from the transactional transfer experiments for ECN with respect to its performance as compared to NON ECN:

- a) As congestion increases (with fixed drop probability) the relative advantage for ECN increases (again the absolute advantage does not increase since more flows are sharing the same bandwidth). For example, from Figures 4.14 and 4.16, if we consider the 5KB transactional flow, as we increase the congestion from medium (5 congesting flows) to high congestion (10 congesting flows) for a maximum drop probability of 0.1 the relative gain for ECN increases from 42% to 62%.
- b) Maintaining the congestion level while adjusting the maximum drop probability indicates that the relative advantage for ECN flow increases. From Figures 4.15a) in the case of high congestion for the 5KB flow, we observe that the number of transactions per second increases from 0.844 to 2.134 which corresponds to an increase in relative gain for ECN of 20% to 135% (from Figure 4.16).
- c) As the transactional data size increases, ECN's advantage diminishes because the probability of recovering from a packet loss via a Fast Retransmit increases for NON ECN. ECN, therefore, has a huge advantage as the transactional data size (i.e. file size) gets smaller, as is observed in the results.

This can be explained by looking at the TCP recovery mechanisms:

NON ECN flows depends, for recovery, on congestion signaling via receiving 3 duplicate ACKs, or (worse) by a retransmit timer expiry, whereas ECN depends mostly on TCP-ECE. This is by design in our experimental setup. [25] shows that most of the TCP loss recovery in fact happens via timeouts for short flows.

The effectiveness of the Fast Retransmit/Recovery algorithm is limited by the fact that there might not be enough data in the pipe to elicit 3 duplicate ACKs.

TCP RENO needs at least 4 packets to recover from losses without going into a timeout. For 5KB (4 packets for MTU of 1500 Bytes) the probability of NON ECN flow going into a timeout from a packet loss is at least 75% (only if the first packet alone is lost would there be a possibility of recovery via Fast Retransmit).

4.4.4. Transactional Transfer results for BECN

In this section we present the results obtained for BECN flows under transactional traffic. The experiments are the same as those carried out for ECN i.e. we use Netperf CRR to issue a succession of requests and responses and carry out the test for an interval of time. We carry out these experiments by varying the file sizes and the level of congestion. Figures 4.17 a), b), and c) present the results for transactional transfer for BECN under moderate congestion. Figure 4.18 presents the percentage increase in performance for BECN. Figures 4.19 and 4.20 are the same set of results but under high congestion.

Figure 4.17a) is a plot of the transaction rate for a request of size 5KB under moderate congestion. We notice that (for example) at the drop rate of 0.1 BECN gets a transaction rate of 2.074 and NON ECN achieves a rate of 1.354. We also notice that as the RED Drop probability increases (maintaining the same level of congestion) the transaction rate of BECN increases.

Fig 4.17a).: Average Transaction rate for BECN and NON ECN v/s RED drop probabilities for a request of 1KB and response of 5KB for Moderate Congestion.

Fig 4.17b).: Average Transaction rate for BECN and NON ECN v/s RED drop probabilities for a request of 1KB and response of 10KB for Moderate Congestion.

Fig 4.17c): Average Transaction rate for BECN and NON ECN v/s RED drop probabilities for a request of 1KB and response of 20KB for Moderate Congestion.

Fig 4.18: Percentage Improvement of BECN v/s NON ECN for transactional flows for Moderate congestion

Fig 4.19a).: Average Transaction rate for BECN and NON ECN v/s RED drop probabilities for a request of 1KB and response of 5KB for High Congestion.

Fig 4.19b).: Average Transaction rate for BECN and NON ECN v/s RED drop probabilities for a request of 1KB and response of 10KB for High Congestion.

Fig 4.19c).: Average Transaction rate for BECN and NON ECN v/s RED drop probabilities for a request of 1KB and response of 20KB for Moderate Congestion.

Fig 4.20: Percentage Improvement of BECN v/s NON ECN for transactional flows for High congestion

4.4.5. Observation and Analysis (BECN)

All the above advantages of ECN apply to BECN, along with the added advantage for BECN that the feedback loop is cut short and hence the sender gets informed of incipient congestion faster than ECN. Also, as the congestion increases, the latency that is experienced at the router for individual packets increases and this will benefit BECN in comparison to NON BECN since it will be informed sooner of congestion as explained in Section 4.3.6.

4.5. Comparative Analysis of ECN and BECN

In this section we aim to present a comparative analysis of NON ECN, ECN and BECN for transactional transfers. From the figure of the transactional testbed (Figure 4.12) note that we first start the congesting flows in the background and then start the transactional flow to collect our results. One set of experiments is done with NON ECN and we note the transaction rate obtained for various file sizes and under various levels of congestion. In the next set of experiments we run the ECN capable flow and take down the result and in the last set we collect the results for BECN. Observing that the flows in the background are the same for all the experiments we can compare these three flows from the results obtained. The following graphs will help in the analysis of ECN on a comparative scale with BECN.

Figure 4.21 a), b), and c) are a plot of the transaction rates obtained for all the three stacks of TCP (i.e. RENO, ECN and NON ECN) for different file sizes under moderate congestion. They are all collected individually under the same level of congestion (i.e. Moderate congestion). We notice the following:

- a) In all cases BECN and ECN perform better than RENO. For example from Figure 4.21a) for RED drop probability of 0.1 the transaction rates for RENO : 1.354 , ECN: 1.93 and BECN: 2.074
- b) Increasing the RED drop probability increases the performance of both ECN and BECN. From Figure 4.21a) as we increase the RED drop probability from 0.02 to 0.5 the transfer rate for ECN increases from 1.29 to 2.30 and for BECN it increases from 1.29 to 2.87.
- c) In all cases (except the one in case of low Drop probability of 0.02) we notice that BECN performs better than ECN.

a) can be explained due to the fact that for both ECN and BECN flows the packets are not getting dropped, as compared to RENO whose packets are getting dropped. This leads RENO to go into timeouts for small data sizes or retransmissions for bigger data sizes (if fast recovery/retransmit algorithm sets in) which affects overall goodput. ECN and BECN on the other hand do not have their packets marked and hence perform better.

b) can be explained due to the fact that (keeping the congestion level the same) as the RED drop probability increases the congesting flows (which are essentially NON ECN) packets get dropped and they backoff. This gives the ECN or BECN flow an advantage since they have not gone into timeout or retransmissions and hence have a better chance of increasing their sender window and utilizing the bandwidth better.

Similarly c) can be explained as follows: compared to ECN, BECN has an additional advantage that it benefits from faster notification with its ICMP source quench mechanism. The anomaly for the case of 0.02 is because the number of packets marked for both ECN and BECN is the same (and it is quite low) but there is a computational overhead for BECN. For each packet that comes into the router (in the case of BECN) an extra computational overhead is introduced for every marked packet i.e. an ICMP source quench needs to be generated. This may be the reason that we see nearly equal performance of ECN and BECN in the case of low Drop Probability.

Fig 4.21a): Comparative analysis of RENO, ECN and BECN based on transactional rates for 5KB under moderate congestion

Fig 4.2(b): Comparative analysis of RENO, ECN and BECN based on transactional rates for 10KB under moderate congestion

Fig 4.2(c): Comparative analysis of RENO, ECN and BECN based on transactional rates for 20KB under moderate congestion

Figure 4.21 d), e), and f) are a plot of the transaction rates obtained for all the three stacks of TCP (i.e. RENO, ECN and NON ECN) for different file sizes under High congestion. We notice that the anomaly for RED drop probability of 0.02 is not present here since there is high congestion and the computational overhead (of generating ISQ) is overshadowed by the advantages of ISQ for BECN under high congestion.

Fig 4.21d): Comparative analysis of RENO, ECN and BECN based on transactional rates for 5KB under High congestion

Fig 4.21e): Comparative analysis of RENO, ECN and BECN based on transactional rates for 10KB under High congestion

Fig 4.21f): Comparative analysis of RENO, ECN and BECN based on transactional rates for 20KB under High congestion

4.6. Conclusion

4.6.1. ECN and BECN:

Following are the conclusions that are common to both ECN and BECN:

ECN and BECN enhancements improve on both bulk and transactional TCP traffic over TCP RENO. The improvement is more obvious for short transactional type of flows (popularly referred to as *mice*).

- Because less retransmits happen with ECN, less traffic on the network is generated. Although the relative amount of data retransmitted in our case is small, the effect could be higher when there are more contributing end systems. The absence of retransmits also implies an improvement in the goodput. This becomes very important for scenarios where bandwidth is expensive such as in low bandwidth links. This implies also that ECN lends itself well to applications that require reliability but do not benefit from retransmissions.
- The fact that ECN and BECN avoid timeouts by getting faster notification (as opposed to traditional packet dropping inference (from 3 duplicate ACKs or, even worse, timeouts) implies less time is spent during error recovery - this also improves goodput.
- ECN could be used to help in service differentiation where the end user is able to “probe” for their target rate faster. Assured forwarding [26] in the diffserv working group at the IETF proposes using RED with varying drop probabilities as a service differentiation mechanism. It has been shown that it is possible that multiple packets within a single window in TCP RENO could be dropped, even in the presence of

RED, likely leading to timeouts [21]. ECN and BECN end systems ignore multiple notifications, which help in countering this scenario resulting in improved goodput. The ECN end system also ends up probing the network faster (to reach an optimal bandwidth). [21] also notes that RENO is the most widely deployed TCP implementation today.

It is clear that the advent of policy management schemes introduces new requirements for transactional type of applications, which constitute a very short query and a response in the order of a few packets. ECN and BECN provide advantages to transactional traffic as we have shown in the experiments.

An important observation to make from our results is that we do not notice multiple drops within a single window of data. Thus, we would expect that our results are not heavily influenced by Reno's performance problems with multiple packets dropped from a window of data. We would expect that the results in this paper concerning the benefits of ECN would also apply to the benefits of ECN for New Reno or SACK implementations of TCP.

4.6.2. BECN

In the case of BECN there is an additional advantage that the feedback control loop is cut short since the sender receives the feedback about the congestion faster than ECN. This quicker notification enhances the performance of a BECN enabled stack over ECN enabled stack. Hence as the latency of the link increases BECN will perform better. Furthermore, BECN does not require any complicated changes at the host side since it is using ISQ that has been used in the past and hence does not require any changes in the host side.

4.7. Future Work

In this thesis we have analyzed two explicit congestion notification mechanisms viz. ECN and BECN and compared them with respect to RENO TCP stack. The results have conclusively shown us that there is a substantial improvement for flows that are ECN capable over flows that are not. Furthermore we believe that the advantage of ECN for other TCP variants will not be the same although there will be improvement. A future work of interest will be to repeat the above experiments that we have conducted with TCP variants such as SACK and NEW RENO. Furthermore we have observed that the relative advantage of BECN increases with increased latency and this is expected to benefit high latency networks such as satellite links etc. Hence it will be interesting to experiment with BECN for these high latency links. Another interesting direction of research would be to study the effect of introducing ECN/BECN on wireless links. In case of wireless links a packet drop is not necessarily an indication of congestion. Most of the packet losses in a wireless environment are due to the lossy nature of wireless links, which leads to a high error rate. For the above reason the assumption that packet loss is an indication of congestion is no longer valid for the wireless environment, hence for adding ECN/BECN support for wireless links the congestion notification method will have to be carefully researched over. Another interesting direction of research is to study the effect of ECN on diffserv. The addition of ECN to other transport protocols such as UDP etc is also an unexplored area of research that needs to be studied in detail.

References:

- [1] W. Stevens "TCP Slow Start, Congestion Avoidance, Fast Retransmit, and Fast Recovery Algorithms", *RFC 2001 Jan 1997*
- [2] M. Allman , V. Paxson , W. Stevens, "TCP Congestion Control", *RFC 2581 April 1999.*
- [3] Thompson K, Miller G.J, Wilder R, " Wide-Area Internet Traffic Patterns and Characteristics". *IEEE Networks Magazine, November/December 1997*
- [4] V.jacobson. "Congestion Avoidance and Control." In *Proceedings of SIGCOMM '88*, Stanford, CA, August 1988
- [5] Floyd & Henderson "The NewReno Modification to TCP's Fast Recovery Algorithm", *RFC 2582, April 1999.*
- [6] M. Mathis, J. Mahdavi, S. Floyd, A. Romanow "TCP Selective Acknowledgement Options "*RFC 2018 Oct 1996.*
- [7] Ramakrishnan, K.K., and Floyd, S., A "Proposal to add Explicit Congestion Notification (ECN) to IP" *RFC 2481, January 1999.*
- [8] Hadi Salim, J., Nandy, B., and Seddigh, N., A proposal for Backward ECN for the Internet Protocol (IPv4/IPv6), internet draft *draft-salim-jhsbnns-ecn-00.txt*, work-in-progress, June 1998.
- [9] Braden, Clark et all "Recommendations on Queue Management and Congestion Avoidance in the Internet" *RFC 2309, April 1998*
- [10] T. V. Lakshman, Arnie Neidhardt, Teunis Ott, The Drop From Front Strategy in TCP Over ATM and Its Interworking with Other Control Features, *Infocom 96, MA28.1.*
- [11] S.Floyd and V. Jacobson. " Random Early Detection Gateways for Congestion Avoidance". *IEEE/ACM Transactions on Networking*, **3**(1), August 1993.

- [12] J. Postel, Internet Control Message Protocol September 1981 *RFC 792*:
<http://www.ietf.org/rfc/rfc0792.txt>
- [13] F. Baker, Requirements for IP Version 4 Routers, *RFC 1812*, June 1995:
<http://www.ietf.org/rfc/rfc1812.txt>
- [14] S. Floyd and V. Jacobson, "Link sharing and Resource Management Models for packet Networks", *IEEE/ACM Transactions on Networking*, Vol. 3 No.4, August 1995.
- [15] Router and Host side changes for TCP for ECN <ftp://ftp.ee.lbl.gov/ECN/ECN-package.tgz>
- [16] The NIST Network Emulation Tool '<http://www.nntd.nist.gov/itg/nistnet/>'
- [17] tcpdump, a protocol packet capture & dumper program <ftp://ftp.ee.lbl.gov/tcpdump.tar.Z>
- [18] TCP dump file analysis tool: <http://jarok.cs.ohiou.edu/software/tcptrace/tcptrace.html>
- [19] W. Feng, D. Kandlur, D. Saha, K. Shin, Techniques for Eliminating Packet Loss in Congested TCP/IP Networks, U. Michigan CSE-TR-349-97, November 1997.
- [20] Raj Jain, "The art of computer systems performance analysis", John Wiley and sons QA76.9.E94J32 ,1991
- [21] Prasad Bagal, Shivkumar Kalyanaraman, Bob Packer, "Comparative study of RED, ECN and TCP Rate Control". <http://www.packeteer.com/technology/articles.htm>
- [22] The network performance tool <http://www.netperf.org/netperf/NetperfPage.html>
- [23] B.A. Mat. "An empirical model of HTTP network traffic. In proceedings. INFOCOMM'97.
- [24] Blake, Steve, et al., An Architecture for Differentiated Services. RFC 2475
- [25] Balakrishnan H, Padmanabhan V, Seshan S, Stern M and Randy H. Katz, "TCP

- Behavior of a busy Internet Server: Analysis and Improvements", *Proceedings of IEEE Infocom*, San Francisco, CA, USA, March '98
<http://www.cs.berkeley.edu/hari/papers/infocom98.ps.gz>
- [26] Heinanen J et al , Assured Forwarding PHB Group, <http://search.ietf.org/internet-drafts/draft-ietf-diffserv-af-06.txt>"
 - [27] S.Floyd. " TCP and Explicit Congestion Notification." *ACM Computer Communications Review*, **24**, October 1994.
 - [28] Kevin Fall, Sally Floyd, " Comparisons of Tahoe, RENO and Sack TCP", *Computer Communications Review*, V. 26 N. 3, July 1996, pp. 5-21
 - [29] E. Hashem. "Analysis of random drop for gateway congestion control." Rep. Lcs tr-465, Lav. Fot Comput. Sci., M.I.T., 1989
 - [30] P.Mishra and H.Kanakia. "A hop by hop rate based congestion control scheme." *Proc. SIGCOMM '92*, pp. 112-123, August 1992.
 - [31] UCB/LBNL/VINT Network Simulator - ns <http://www-mash.cs.berkeley.edu/ns/>
 - [32] K.K. Ramakrishnan and R.Jain. "A Binary feedback scheme for congestion avoidance in computer networks." *ACM Trans. Comput. Syst.*, **8**(2):158-181,1990