INTERNET MULTICAST CONGESTION CONTROL

A THESIS SUBMITTED TO THE GRADUATE SCHOOL OF NATURAL

AND APPLIED SCIENCES

OF THE MIDDLE EAST TECHNICAL UNIVERSITY

BY

KEREM ÖNAL

IN PARTIAL FULFILMENT OF THE REQUIREMENTS FOR

THE DEGREE OF MASTER OF SCIENCE

IN THE DEPARTMENT OF

ELECTRICAL AND ELECTRONICS ENGINEERING

FEBRUARY 2004

Approval of the Graduate School of Natural and Applied Sciences

_____

Prof. Dr. Canan Özgen
Director

I certify that this thesis satisfies all the requirements as a thesis for the degree of Master of Science.

_____

Prof. Dr. Mübeccel Demirekler
Head of the Department

This is to certify that we have read this thesis and that in our opinion it is fully adequate, in scope and quality, as a thesis for the degree of Master of Science.

_____

Asst. Prof. Dr. Cüneyt F. Bazlamaçcı
Supervisor

Examining Committee in Charge:

Prof. Dr. Hasan Güran _____

Prof. Dr. Semih Bilgen _____

Assoc. Prof. Dr. Buyurman Baykal _____

Dr. Atilla Özgit _____

Asst. Prof. Dr. Cüneyt F. Bazlamaçcı _____

ABSTRACT

INTERNET MULTICAST CONGESTION CONTROL

Önal, Kerem

M.S., Department of Electrical and Electronics Engineering

Supervisor: Asst. Prof. Dr. Cüneyt F. Bazlamaçcı

February 2004, 80 pages

Congestion control is among the fundamental problems of Internet multicast. It is an active research area with many challenges. In this study, an introduction to Internet congestion control and a brief literature survey of current multicast congestion control protocols is presented. Then two recently proposed "single-rate, end-to-end, rate based" class of protocols, namely LESBCC and TFMCC are evaluated with respect to their intersession fairness (TCP-friendliness), smoothness and responsiveness criteria. Throughout the experiments, which are conducted using a widely accepted network simulation tool 'ns', different topologies have been employed.

Keywords: Multicast, Congestion Control, Simulation, LESBCC, TFMCC

ÖZ

İNTERNET ÇOKLU YAYIN TIKANIKLIK KONTROLÜ

Önal, Kerem

Yüksek Lisans, Elektrik Elektronik Mühendisliği Bölümü

Tez Yöneticisi: Y. Doç. Dr. Cüneyt F. Bazlamaçcı

Şubat 2004, 80 sayfa

Tıkanıklık kontrolü internet çoklu yayındaki en temel sorunlardan biridir. Bu konu halen birçok zorluklar içeren aktif bir çalışma alanıdır. Bu çalışma içerisinde, internet tıkanıklık kontrolü için bir ön bilgi ve halihazır çoklu yayın tıkanıklık kontrol protokolleri sunulmuştur. Daha sonra, yakın zamanda önerilmiş bulunan "tek hızlı, uçtan uca, hız temelli" protocol sınıfından iki tanesi, yani LESBCC ve TFMCC, oturum arası denkserlik kriterleri (TCP-uyumluluk), akıcılık ve tepki hızları yönlerinden görgül olarak değerlendirilmişlerdir. Geniş olarak kabul görmüş bir ağ benzetim programı olan 'ns' kullanılan deneylerde değişik topolojiler denenmiştir.

Anahtar Kelimeler: Çoklu Yayın, Tıkanıklık Kontrolü, Benzetim, LESBCC, TFMCC

ACKNOWLEDGEMENT

I would like to express my deepest gratitude and appreciation to my supervisor Asst. Prof. Dr. Cüneyt F. Bazlamaçcı who inspired, encouraged and supported me at all levels of this study.

I would like to thank Eren Gürses and Ilgaz Korkmaz, whose friendship, support and suggestions made great contributions to this work.

I also want to thank Özgür Koçak, Serdar Dündar, Gökhan Ceylan, Serdar İnce, Emre Güriş, Bora Yeşiltepe and also Wolfgang Amadeus Mozart who have always injected happiness and excitement to my life.

The greatest thanks go to my family members for their infinite support.

I would like to thank my loveliest mother again for her wonderful personality. Thank you for everything!

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF ABBREVIATIONS

ACK: Acknowledgement

AIMD: Additive Increase Multiplicative Decrease

BGMP: Border Gateway Multicast Protocol

CLR: Current Limiting Receiver

DVMRP: Distance Vector Multicast Routing Protocol

FIFO: First In First Out

FTP: File Transfer Protocol

GAIMD: General Additive Increase Multiplicative Decrease

HTTP: Hypertext Transfer Protocol

IGMP: Internet Group Management Protocol

IP: Internet Protocol

LAN: Local Area Network

LE: Loss Event

LESBCC: Loss Event Oriented Source Based Multicast Congestion Control

LI: Loss Indicator

LPRF: Linear Proportional Response Filter

MTCP: Multicast Transport Control Protocol

NACK: Negative Acknowledgement

NCA: Nominee Congestion Avoidance

NS: Network Simulator

OTcl: Object Tool Command Language

PGM: Pragmatic General Multicast

PGMCC: Pragmatic General Multicast Congestion Control

PIM: Protocol Independent Multicast

QoS: Quality of Service

RTP: Realtime Transport Protocol

RTT: Round Trip Time

sLESBCC: Smooth LESBCC

SMTP: Simple Mail Transport Protocol

SRTT: Smoothed Round Trip Time

TCP: Transport Control Protocol

TFMCC: TCP-friendly Multicast Congestion Control

TFRC: TCP-friendly Rate Control

UDP: User Datagram Protocol

CHAPTER 1

INTRODUCTION

Multicasting allows information exchange among multiple senders and receivers. Nowadays many applications require multipoint delivery in the Internet such as audio/video conferencing, Internet games, one-to-many and many-to-many file distribution, distance learning and web cache updating. Therefore, Internet multicast is an emerging need for the development of the Internet and this must be done in a scalable and efficient way.

IP Multicast is used widely on the network layer for routing. It delivers source traffic to multiple receivers without adding any additional burden on the source or the receivers. There exist protocols to multicast in an efficient way such as Internet Group Management Protocol (IGMP) [Deering 1989], [Fenner 1997], [Cain 2002] which is used to build up and manage the multicast groups and Protocol Independent Multicast (PIM) [Adams 2002], [Fenner 2002], Distance Vector Multicast Routing Protocol (DVMRP) [Waitzman 1988] and Border Gateway Multicast Protocol (BGMP) [Kumar 1998] which is used to route the data packets.

On transport layer there is still no standard protocol released and accepted widely. UDP seems to be the easiest and the cheapest way. These kinds of approaches leave a massive work to be done to the application layer. As UDP serves no feedback from receivers to the senders, there is no knowledge about the reliability of data packets.

Pragmatic General Multicast (PGM) [Speakman 2000], Real-time transport protocol (RTP) [Schulzrinne 1996] and many other multicast transport layer protocols do not cover the congestion control part; or due to the needs of the protocol, they are not fair to the other users on the network. In fact, lack of attention to the congestion control issue results in severe service degradation or "Internet meltdown". This phenomenon was first reported in mid 1980s [Nagle 1984] and is technically called "congestion collapse".

Congestion collapse in today's Internet is prevented only by the congestion control mechanisms in TCP. Therefore the success of the Internet can be continued only if we use protocols that respond to network congestion by reducing the load presented to the network.

In this thesis, internet multicast congestion control is studied. The challenges and the approaches used to solve them are presented. Two of the recently proposed protocols are studied in detail using simulations. One of these protocols, namely

TFMCC, is in the state of an internet draft. The other one, i.e. LESBCC, is proposed more recently. The thesis aims two things: one is the implementation of the LESBCC protocol and its addition to the NS protocol stack and the second being the evaluation and detailed comparison of LESBCC with a widely accepted multicast protocol, TFMCC.

The thesis is organized as follows: Chapter 2 presents a literature survey on internet multicast congestion control. Unicast and multicast environments are separated and the major challenges in controlling the congestion in a multicast environment are discussed.

In Chapter 3, the existing approaches are classified and two of the more recently proposed protocols, namely LESBCC [Thapliyal 2002] and TFMCC [Widmer 2001], which are simulated in this study, are introduced and briefly overviewed.

In Chapter 4, LESBCC and TFMCC are investigated mainly in three parts. The results of the simulation experiments are presented in this chapter. The two protocols are compared empirically with respect to their intersession fairness criterion, protocol overhead, deployment issues and responsiveness to changes in network conditions.

Chapter 4 concludes the study and states some possible future work. Appendices give the C++ and OTcl code written to perform the simulations with NS.

CHAPTER 2

INTERNET CONGESTION CONTROL

2.1 Introduction

The Internet protocol architecture is based on a connectionless end to-end packet service using the IP protocol. IP protocol is flexible and robust, and the advantages of its connectionless design are widely accepted. Nevertheless, a good service is not expected under heavy load. Careless design of congestion control at transport layer can result in severe service degradation or Internet meltdown.

The original specification of TCP already included a window-based flow control. It was needed for those receivers which had lower data receiving speeds than their senders sending speeds. So, by this flow control, an overflow of the receiver's data buffer space available for that connection was prevented. Segments could have been lost either due to errors or to network congestion, but they had no effect on the flow-control window. In other words, the senders did

not respond, by any means, to congestion on the network in the original specification.

Congestion control mechanisms, which fix the Internet meltdown, were first provided by Van Jacobson [Jacobson 1988]. Now, congestion control mechanisms are required in any TCP implementation. During congestion, these mechanisms cause a TCP sender to reduce its data sending rate. In this case, TCP flows are responsive to congestion signals (i.e., dropped packets) from the network. This is the main idea that lies behind the TCP congestion control mechanisms, which prevent the congestion collapse of today's Internet.

2.2 Unicast

Since TCP is widely used, many research efforts are carried out to improve it. Many small improvements have been done. TCP is very useful and applicable for a wide range of applications that run fine with best effort flows. Despite all the improvements, however, TCP still does not work well with audio/video applications. Because of its additive-increase-multiplicative-decrease (AIMD) module, a short-term congestion in a path halves the data sending rate, which may not be acceptable for the users. Hence, UDP is widely used for streaming applications, which, on the other hand, incorporates unfair sessions on the Internet [RFC 2914].

One of the improvements on TCP's congestion control mechanism is presented by Yang and Lam [Yang 2000]. In TCP's AIMD mechanism, in congestion avoidance state, the window size is increased by α per window of packets acknowledged and it is decreased to β of the current value when there is congestion indication. TCP implementations uses $\alpha = 1$ and $\beta = 1/2$. Instead of these values, in GAIMD, $\alpha = 0.31$ and $\beta = 7/8$. According to the authors, this implementation is highly TCP-friendly and these GAIMD flows have reduced rate fluctuations compared to TCP flows.

Another solution is presented by TFRC [RFC 3448]. TFRC is a rate-based TCP-friendly congestion control protocol. It uses a TCP-friendly formula which imitates the long term throughput of TCP's windows-based approach. TFRC is also less aggressive than TCP so its long term rate behavior is more stable than TCP's saw-tooth like behavior.

TFRC briefly works as follows: the receiver measures packet loss rate P and sends feedback packet to the sender to report P. The sender receives this report and measures the RTT for this receiver. Now the sender has RTT and P and it already knows the packet size from the beginning. Then, it calculates the TCP-friendly sending rate based on the TCP throughput equation and sets its rate to this calculated value.

2.3 Multicast

UDP is the dominant protocol in today's multicast traffic. It does not have a congestion control mechanism and allows multicast traffic to use more bandwidth than other responsive traffic. This leads to serious problems such as unfair usage of the network resources and the congestion collapse, which occurs when the sources keep sending packets that are surely going to be dropped later by the network.

One way to avoid congestion is to use QoS schemes in the network layer. This way, there occurs no congestion in the network since enough resources are guaranteed to users. However, QoS requirements are likely to be supported only by a small fraction of Internet.

The other way, which will be the focus in this thesis, is to use transport layer mechanisms for multicast congestion control. The three major challenges in controlling the congestion in a multicast environment are scalability, heterogeneity and fairness.

2.3.1 Scalability

Scalability is a challenge for congestion control when the number of receivers is large. The scalability of the multicast congestion control protocol is achievable only by controlling the feedback messages efficiently. There must be a suppression mechanism to filter some of the feedback messages. For this purpose it is often helpful to distinguish between shared loss and independent loss. Figure 1 illustrates the two types of losses.

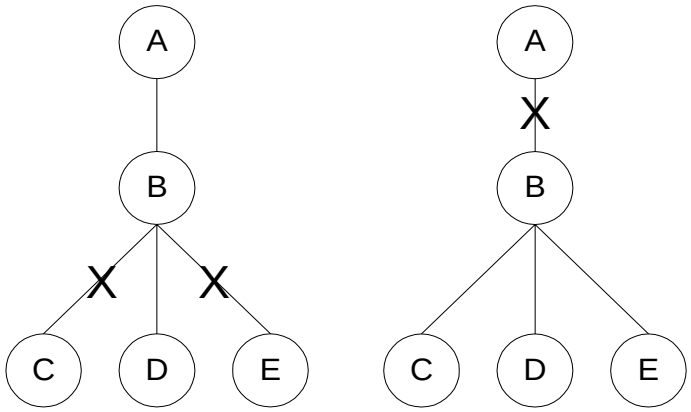

Figure 1: Independent and Shared Loss

(a) Independent losses. The two losses at links BC and BE are independent.

(b) Shared loss. The loss occurred at link AB. Nodes C, D and E all observe the same loss.

In fact, it is not clearly defined which loss is shared and which is independent. However, independent loss can be thought to occur on a less common link and hence affect a small number of receivers (Figure 1.a). All such losses should not be reported to the sender and perceived as an indication of congestion in the network. The "Drop-to-zero" problem [Rhee 1999] occurs when the sender overreacts to every packet loss on the links and transmits data at a very low rate

due to unnecessary retransmissions. Today, probabilistic schemes, representative based schemes and the filtering mechanisms [Bhattacharyya 1999] are all used to avoid this problem.

Shared losses occur on very common links, especially near the sender, in the multicast distribution tree and hence all the downstream receivers observe the losses and possibly report these (Figure 1.b). This time the "feedback implosion" problem occurs, which means that the sender receives much more feedback packets than it can handle. Feedback suppression techniques using a hierarchical structure or timer-based mechanisms can be used to overcome this problem. The probabilistic and the representative-based schemes mentioned in the previous paragraph can also help to avoid feedback implosion.

As a result, scalability increases as the system deals with smaller number of feedback information packets which decreases the computational power needed. Scalability also increases as the computational load and memory requirements decreases during the evaluation process of these feedback messages. If is often preferred to share this computational load both at the sender and the receiver side. Some or all of this computation can be done in the routers, but this clearly decreases the deployability of the scheme, which is not recommended for most cases.

2.3.2 Heterogeneity and interreceiver fairness

In a multicast group, receiver connections can vary from a dialup link to 100 Mbps LAN. Each receiver naturally wants to have a transmission rate that matches its receiving rate. As a result, heterogeneity of group members and network capacities is another major challenge for multicast congestion control. This leads to the interreceiver fairness, which requires that the transmission rate of a multicast group should satisfy the faster receivers in the group while not overwhelming the slower ones at the same time.

In single-rate protocols, the offered rate is dictated by the receiver with the worst performance so the throughput cannot exceed the slowest receiver's receiving capability. Therefore, the closer the offered throughput to the slowest receiver's receiving capability, the better the interreceiver fairness is.

2.3.3 Intersession fairness

A most basic and necessary requirement for end-to-end multicast congestion control is fairness among multicast and unicast sessions. A multicast flow must be responsive in a best effort network and should not use very high or low bandwidths compared to other traffic.

The definition of "intersession fairness" is the subject of a current debate and different definitions are possible. In the multicast congestion control literature, it is widely accepted that a proposed protocol must be TCP-friendly. The definition of TCP-friendliness is first given as TCP-compatible in [RFC 2309]:

"We introduce the term "TCP-compatible" for a flow that behaves under congestion like a flow produced by a conformant TCP. A TCP-compatible flow is responsive to congestion notification, and in steady-state it uses no more bandwidth than a conformant TCP running under comparable conditions"

As the definition is not quantitative it is not clear how to test whether a flow is TCP-friendly or not. And also there are numerous TCP implementations which have different throughputs under the same conditions. First test is done in an award-winning paper [Floyd 1999]:

"The test of TCP-friendliness does not attempt to verify that a flow responds to each and every packet drop exactly as would a conformant TCP flow. It does however assume a flow should not use more bandwidth than would the most aggressive conformant TCP implementation in the same circumstances"

In [Floyd 1999], main purpose is to stop unfairness between unresponsive flows and TCP using a control scheme in routers. Continuing to analyze [Floyd 1999], we see that the most aggressive TCP throughput in steady-state is defined as Eqn 1:

$$T \leq \frac{1.22 * B}{R * \sqrt{p}} \qquad \text{Eqn. 1}$$

where T is the maximum data sending rate in bytes/seconds, B is the packet size in bytes, assuming a fairly constant round trip time, R in seconds, and loss rate, p. And it is accepted, with some overhead, that a flow is TCP-friendly if its throughput, holds for the inequality below in Eqn 2:

$$T \leq \frac{1.45 * B}{R * \sqrt{p}} \qquad \text{Eqn. 2}$$

The TCP-friendliness ratio (F) is defined as Eqn 3 [Padhye 2000][Widmer 2000]:

$$F = \frac{T_M}{T_{TCP}} \qquad \text{Eqn. 3}$$

where T_M is the throughput of multicast session and T_{TCP} is the throughput of TCP session on the bottleneck link. In the ideal case, F equals to 1 showing the absolute fairness. Using the ratio between Eqn. 1 and Eqn. 2, we can conclude that (Eqn 4):

$$F = \frac{T_M}{T_{TCP}} \leq \frac{1.45}{1.22} \cong 1.19 \qquad \text{Eqn. 4}$$

is the uppermost limit defining TCP-friendliness.

As can be seen, F does not have a lower bound. A multicast flow is TCP-friendly even if its throughput is 5 times lower than TCP. This is actually related to performance, not TCP-friendliness. It is obvious that a multicast flow should not starve itself for the sake of TCP-friendliness. It should force TCP to be multicast-friendly. Therefore P, the inverse of F, can be used as a performance metric as in Eqn. 5. We are not satisfied with a multicast flow's performance if its P value is more than 1.19.

$$P = \frac{T_{TCP}}{T_M} \leq 1.19 \qquad \text{Eqn. 5}$$

In most of the papers that presents a new multicast congestion control protocol, TCP-friendliness is the only fairness criteria, which is then proven only by simulations without any quantitative detail. None of the protocols are accepted yet as a standard by IETF. To quote [Wang 1998]:

"First of all, there is no consensus on the fairness issue between multicast and unicast traffic, let alone a useful quantitative definition … we believe that a consensus on the definition of the relative fairness between multicast and unicast traffic is achievable once an algorithm shown to be "reasonably fair" to TCP is accepted by the Internet community"

The reason why TFMCC is chosen to be tested in this thesis is closely related to the above definition. TFMCC is at a status of internet draft and it is probably the

most widely accepted multicast congestion control algorithm. It uses the same TCP-friendly equation used in a unicast congestion control protocol, TFRC, which is a proposed standard by IETF.

In this thesis, the TCP-friendliness of LESBCC and TFMCC are both analyzed according to F. In cases where both protocols are unfair, LESBCC is compared to TFMCC.

On the other side of the coin, there are some claims that TCP-friendliness should not be mandatory for the multicast sessions. It can also be seen that it is not defined yet how well to test this approach. To quote [Wang 1998]:
"Should a multicast session be treated as a single session, which deserves no more bandwidth than a single TCP session, when they share network resources? Or should the multicast session be given more bandwidth than TCP connections because it is intended to serve more receivers? If the latter argument is creditable how much bandwidth should be given to the multicast session and how do we define "fairness" in this case?"

2.4 Classification of Schemes

The proposed multicast congestion control schemes can be broadly categorized according to whether they are single-rate or multirate, end-to-end or network-supported and window-based or rate-based.

2.4.1 Single-rate

In a single-rate scheme, all receivers of a multicast session observe the same data rate. In general the slowest receiver in the session determines the sending rate. The difference between the data sending rate and the maximum receiving rate of the slowest receiver must be as small as possible. The goal is to achieve the maximum value of a predefined interreceiver fairness [Jiang 1998]. The single-rate schemes usually cannot achieve good interreceiver fairness because of the high heterogeneity among network paths and receivers. However, the advantage of this scheme is that it is easier to implement and to deploy and the support from intermediate nodes is not mandatory.

2.4.2 Multirate

In multirate schemes, source sends data at multiple rates to receivers with different capabilities. The primitive way is "simulcasting" [Cheung 1996], [Jiang 2000] in which the same original data is encoded into a number of streams with different rates. The better way is layered multicast [Legout 2000], [Byers 2001], which divides the data into several layers. Different multicast groups receive different layers. There is an order among the layers in the cumulative layering, and no such ordering exists in the non-cumulative layering schemes. Multi-rate transmission can also be achieved by router filtering [Luby 1999], transcoding [Assuncao 1996], and store-and-forward approaches.

Despite the numerous technical advances and significant effort, a suitable multirate multicast congestion control scheme is still not deployed for wide area. It is often extremely complex to test and validate such schemes and implementations require a great deal of effort. If store-and-forward mechanism is used, this scheme may also require excessive storage at routers. Figure 2 shows the classification according to single-rate or multirate.

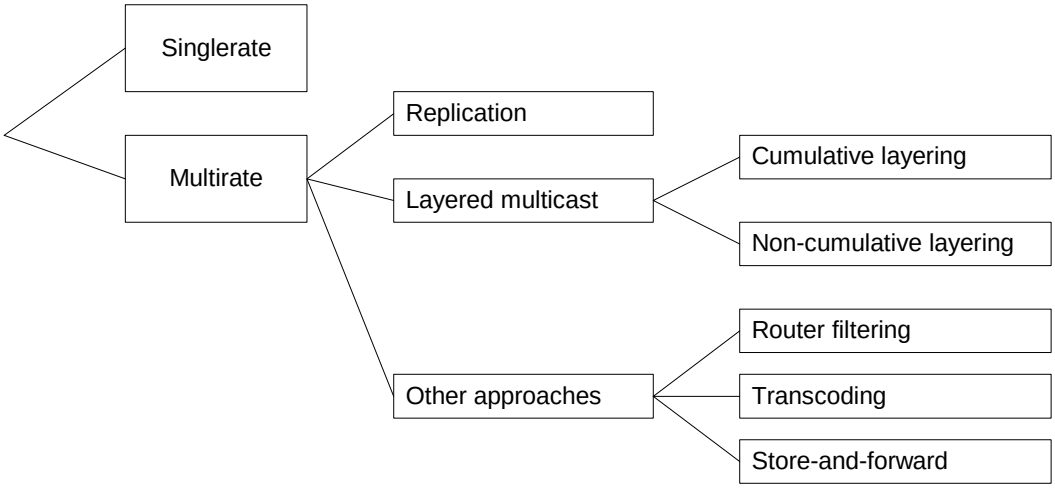

Figure 2: Classification of multicast schemes according to rate type.

2.4.3 End-to-end

End-to-end schemes do not require network support beyond multicast delivery. Most of the multicast congestion control protocols fit in this category because implementation and deployment is easier. All congestion control functionality is provided by the senders and receivers. End-to-end schemes can be divided into sender-based, receiver-based and hybrid schemes. In a sender-based scheme, the sender adjusts the transmission rate using the feedback from receivers. In receiver based schemes receivers calculate and report the desired transmission rate to the sender. Sender decides the appropriate transmission rate. Receiver-driven schemes are more scalable but also are more complex and harder to deploy. A hybrid scheme is receiver-driven but the sender also adjusts sending rates.

2.4.4 Network-supported

These kinds of schemes perform better by adopting additional functionality, such as feedback aggregation and router filtering, in the network. But they highly increase the complexity in the network. It is much harder to implement and deploy than end-to-end protocols. Figure 3 shows the classification of multicast schemes according to network support availability.

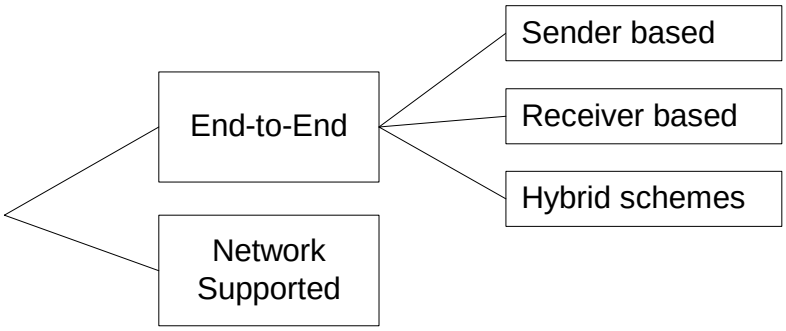

Figure 3: Classification of multicast schemes according to network support availability.

2.4.5 Window-based

In a window-based scheme, either the sender or receivers maintain a congestion window like the TCP. The congestion window represents the amount of data which may be sent in one RTT. The window size decreases when congestion is detected and increases with some timeout mechanism when there is no detected congestion. The sending rate is adjusted by this window size. The windows can be common for all receivers of a multicast group or distinct for some specific

group or even for each receiver. Since the common window is usually set according to the slowest receiver, this approach worsens the interreceiver fairness principle and restricts the throughput of the multicast session to a value that is much lower than the value allowed in the network [Golestani 1999].

2.4.6 Rate-based

In a rate-based scheme, the transmission rate is adjusted directly, through a probing or equation-based approach. In the probing approach, the transmission rate is decreased when congestion is detected and increased when there is no congestion. In the equation-based approach, using measured loss probability and RTT values, the proper transmission rate is calculated using the TCP throughput models [Padhye 1998]. Figure 4 shows the classification of multicast schemes according to being rate or window based.

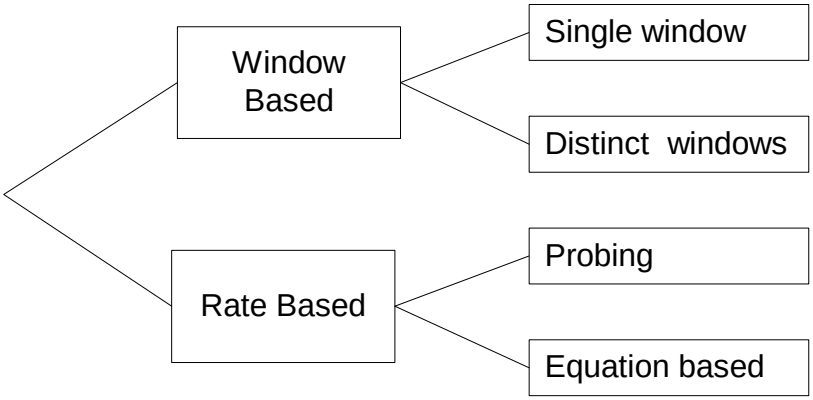

Figure 4: Classification of multicast schemes according to being rate or window based.

CHAPTER 3

MULTICAST CONGESTION CONTROL PROTOCOLS

3.1 Introduction

In this section, we briefly describe the mechanisms of some multicast congestion control protocols and give a detailed description of two multicast congestion control protocols, which are investigated in next chapter. Both are chosen to be single-rate and end to end schemes in order to be able to compare the performances on a common ground.

3.2 NCA

Nominee Congestion Avoidance [Kasera 2000] is a single-rate TCP-friendly multicast congestion control protocol. It consists of two parts: a nomination algorithm and a rate adjustment algorithm.

The purpose of the nomination algorithm is to dynamically select a nominee representing the worst path. Each receiver periodically sends the estimated loss probability and RTT to its upstream active server. An active server identifies the worst performing receiver among its children based on a TCP-friendly formula and reports that information upstream. Eventually, the sender will identify the worst performing receiver of the entire group as the nominee and ask it to send an ACK for every packet it receives. The rate adjustment algorithm operates in a similar way as TCP NewReno [Floyd 1999] using ACKs from the nominee. The main difference between them is that the algorithm does not retransmit packets on detecting losses, since NCA is a congestion control protocol decoupled from the error control functionality.

3.3 PGMCC

PGMCC [Rizzo 2001] is very similar to NCA. It is again a single-rate TCP-friendly multicast congestion control protocol.

The sender continuously monitors receiver reports embedded into NACKs coming from receivers. Then it selects the group representative, the acker, as the receiver with the worst throughput according to the control scheme being used. A window-based congestion control scheme similar to TCP congestion control is

run between the sender and the acker, which then sends positive ACKs for each data packet.

The critical part in this protocol is to select the right acker timely. PGMCC does not use any TCP-friendly equation. As can be seen, the main idea is the same with NCA but the acker or nominee selection mechanism is somewhat different.

3.4 MTCP

Multicast TCP (MTCP) [Rhee 1999] is a hierarchical network-supported window-based protocol for multicast flow and congestion control as well as error control. The hierarchy includes the sender as the root, receivers as leaves and Sender Agents (SA) in between.

The sender and each SA maintain a congestion window (cwnd) and a transmit window (twnd). The cwnd estimates the congestion level of the network and is maintained using congestion control mechanisms similar to TCP Vegas. The twnd indicates outstanding packets at the sender or an SA, which is increased when a packet arrives at the SA (or the sender) and decreased when the packet is acknowledged by all the children of the SA (or the sender). Each SA sends to its upstream SA (or the sender) congestion summaries, which include a minimum congestion window (minCwnd) and a maximum transmit window (maxTwnd).

The minCwnd is the minimum value of the SA's own cwnd and the cwnds reported by its immediate downstream children. The maxTwnd is the maximum value of the SA's own twnd and the twnds reported by its immediate downstream children. SAs whose children are leaf nodes just include their own cwnd and twnd in the congestion summary.

Congestion summaries are piggybacked on ACKs and NACKs or sent periodically when ACKs and NACKs are lacking to prevent protocol deadlocks. Each receiver also sends an advertised window (awnd) upstream indicating the number of available buffers. The awnds are aggregated along the hierarchy. Defining a current window (curwnd) at the sender as the difference between its minCwnd and maxTwnd and the sender's awnd as the minimum value reported by its children, the number of packets sent each time should be no more than min(curwnd, awnd).

3.5 LESBCC

The LESBCC [Thapliyal 2002] fits into the class of single-rate, end-to-end, rate-based probing schemes. It is purely sender-based. The only work done by each receiver is to send a one bit loss indicator (NACK) per packet loss observed by that receiver. Sender leverages a set of filters and an RTT estimator is used to form an input to a rate-based AIMD module.

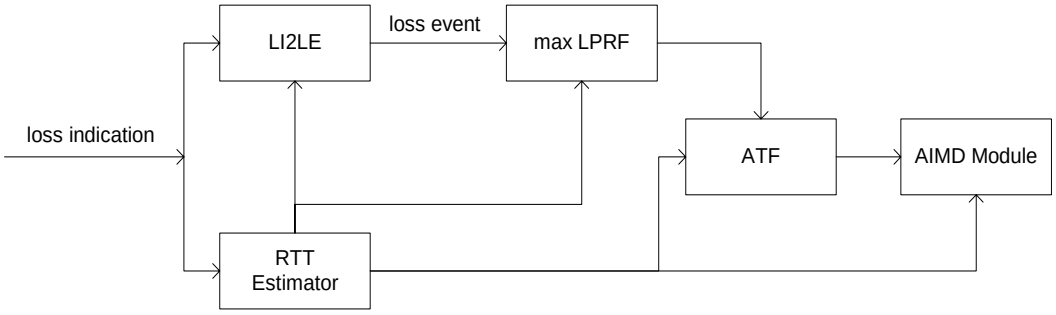

Figure 5: Cascaded filter model of LESBCC

The filters address all the main pieces of the single-rate multicast congestion control problem. Figure 5 illustrates the outline of the building blocks of the scheme implemented at the multicast source. The modules work as follows:

3.5.1 RTT Estimation

All filters and the AIMD module need an RTT estimate. It works very similar to the TCP timeout procedure, i.e., it calculates a smoothed RTT (SRTT) and a mean deviation which approximates the standard deviation σ by the following method.

There is no ACK in LESBCC's mechanism. Sender records the time when it sends a packet. If a NACK for that packet comes from a receiver, the difference

between the current time and the recorded time gives us an RTT sample, CRTT. Otherwise no RTT value can be calculated. So, only a packet which is not received by at least one of the receivers affects the SRTT value. The absolute value of the difference between SRTT and CRTT is the current deviation, δ. SRTT and σ are calculated using these two samples, CRTT and δ as in Eqn. 6 and Eqn 7.

$$SRTT = 0.875 * SRTT + 0.125 * CRTT \quad \text{Eqn. 6}$$

$$\sigma = 0.875 * \sigma + 0.125 * \delta \quad \text{Eqn. 7}$$

3.5.2 LI2LE Filter

When a receiver detects a packet loss it sends a NACK to the sender. NACK is named as loss indication (LI) in LESBCC. In this filter for every receiver LIs are converted to loss events (LE). An LE is a per-receiver binary number which is 1 or 0. If one or more LIs are generated per (SRTT + 2σ) per-receiver LE is 1 and and it is 0 otherwise. When an LI is converted to an LE, the time is recorded at the sender for that receiver's last pass ($T_{LastPassed}$). If, in a period of SRTT + 2σ duration, a new LI arrives from that receiver then it is filtered. Otherwise it passes as an LE and the timestamp $T_{LastPassed}$ is updated to the current time.

3.5.3 Max-LPR Filter

The goal of this filter is to pass the number of LEs corresponding to what the source would have received from the worst case receiver on the average. This filter is an extension of Bhattacharya et al's Linear Proportional Response Filter (LPRF) [Bhattacharya 2001].

Sender saves the number of LE's of every receiver in an array. LE_{max} is the greatest value throughout the array and LE_{all} is the total number of LE's received from all of the receivers. This filter passes an LE to the next filter with a probability of LE_{max}/ LE_{all}.

If most of the packet drops within the system are shared losses then sender receives N LIs for every packet drop. The probability decreases to 1/N, where N is the number of receivers. On the other hand, if most of the packet drops are independent then the probability increases with respect to the shared loss case accordingly.

In addition, LE count of each receiver is decreased by 10% every 100 SRTT. This is to respond actively to the recent changes in the network conditions.

3.5.4 Adaptive Time Filter

This filter simply drops excess LEs passed by Max-LPRF in any RTT to enforce at most one rate reduction per SRTT + 4σ. In addition, the filter also imposes a "silence period" of 0.5(SRTT + 4σ) when no packets are sent. The goal is to reduce the probability of losing any control traffic or retransmissions during this phase as suggested in [Natu 2001].

3.5.5 AIMD Module

The AIMD module works to imitate the behavior of the AIMD module of TCP. LESBCC is rate-based. If an LI passes all filters then the data sending rate is halved. Every SRTT+2σ, rate is increased by B/(SRTT+2σ) where B is the packet size. Hence the sending rate is increased by 1/(SRTT+2σ) in terms of packets per seconds. This corresponds to an increase in the window size by one packet every SRTT+2σ in TCP. The following table explains the behavior. Initially, assume that the data sending rate is R packets per second. Third column is equal to the rate times SRTT+2σ.

Time(s)	Rate (pkts/s)	Number of packets sent in SRTT+2σ (pkts)
0	R	R*(SRTT+2σ)
SRTT+2σ	R+1/ (SRTT+2σ)	R*(SRTT+2σ)+1
2(SRTT+2σ)	R+2/ (SRTT+2σ)	R*(SRTT+2σ)+2

Figure 6: Additive increase behavior of LESBCC

3.5.6 Extension to AIMD Module

AIMD module used in TCP is not designed for smooth data delivery. All AIMD modules use two constants. TCP increases the congestion window bye one packet when there is no congestion so the *additive constant* is 1. It halves the data sending rate, which dramatically disrupts the smoothness, in response to single packet drop, so the *multiplicative constant* is 0.5. In other words, TCP uses AIMD(1,1/2). In order to be TCP-friendly LESBCC uses the same logic at the cost of loosing its smoothness.

In its original paper, a solution has already been proposed to increase the smoothness of the LESBCC. The approach uses the TFRC equation as used in TFMCC. It is an optional citation in the paper and it is not fully tested in various conditions. In TFMCC, the calculation of this equation is done at the receivers to reduce the computational load on the source. The aim is to increase the scalability. LESBCC is a sender-based multicast congestion control protocol. The

calculation of this equation brings an additional computational complexity to the O(N) state requirements of both LI2LE and maxLPR filter. Also there is a need to calculate the packet loss in the system, which is used in the TFRC equation.

As a result we prefer to use a much simpler extension to LESBCC only by changing the constants of its AIMD module. In [Floyd 2000], AIMD(1/5,7/8) or AIMD(2/5,7/8) has already been suggested for the smooth delivery of data while being TCP-friendly at the same time. In [Yang 2000], AIMD(0.31,7/8) is defined as GAIMD and it is claimed that GAIMD is also TCP-friendly. In the next chapter, we implement these schemes in LESBCC and observe satisfactory results.

3.6 TFMCC

TFMCC [Widmer 2001] is a single-rate multicast congestion control protocol designed to provide smooth rate change over time. TFMCC extends the basic equation-based control mechanisms of TFRC into the multicast domain. The fundamental idea is to have each receiver evaluate a control equation Eqn. 8 derived from the model of TCP's long-term throughput. And then use this to directly control the sender's transmission rate.

$$T_{TCP} = \frac{8S}{RTT(\sqrt{\frac{2p}{3}} + 12p(1+32p^2)\sqrt{\frac{3p}{3}})} \quad \text{Eqn. 8}$$

where,

T_{TCP} is transmission rate in bits/sec,

S is the packet size in bytes,

RTT is the round trip time in seconds,

p is the steady state loss event rate, between 0 and 1.

An overview of TFMCC functionality is as follows:

- Each receiver measures the packet loss rate.
- The receiver measures or estimates the round-trip time to the sender.
- The receiver uses the control equation to derive an acceptable transmission rate from the measured loss rate and round-trip time.
- The receiver sends the calculated transmission rate to the sender.
- A feedback suppression scheme is used to prevent feedback implosion while ensuring that feedback from the slowest receiver always reaches the sender.
- The sender adjusts the sending rate from the feedback information.

In TFMCC, the receiver that the sender believes to have the current lowest expected throughput of the group is selected as the current limiting receiver (CLR). The CLR sends continuous, immediate feedback to the sender without

any suppression, so the sender can use the CLR's feedback to adjust the transmission rate. In addition, any receiver whose expected throughput is lower than the sender's current rate sends a feedback message, and to avoid feedback implosion, biased feedback timers in favor of receivers with lower rates are used.

3.6.1 Measuring the Loss Event Rate

In TFMCC, a receiver aggregates the packet losses into loss events, defined as one or more packets lost during a round-trip time. The number of packets between consecutive loss events is called a loss interval. As in Eqn. 9, the average loss interval size can be computed as the weighted average of the m most recent loss intervals $l_k, \ldots, l_{k-m+1}$:

$$l_{avg}(k) = \frac{\sum_{i=0}^{m-1} w_i l_{k-i}}{\sum_{i=0}^{m-1} w_i} \qquad \text{Eqn. 9}$$

The weights w_i are chosen so that very recent loss intervals receive the same high weights, while the weights gradually decrease to 0 for older loss intervals. The loss event rate p used as an input for the TCP model is then taken to be the inverse of l_{avg}. A more through discussion of this loss measurement mechanism can be seen in the original TFMCC paper [Widmer 2001].

3.6.2 Round-trip Time Measurements

Each receiver starts with an initial RTT estimate that is used until a real measurement is made. A receiver measures the RTT by sending a timestamped feedback to the sender, which then echoes the timestamp and receiver ID in the header of a data packet. An exponentially weighted moving average (Eqn. 10) is used to prevent a single large RTT measurement from greatly impacting the sending rate.

$$t_{RTT} = \beta \cdot t_{RTT}^{inst} + (1-\beta) \cdot t_{RTT} \qquad \text{Eqn. 10}$$

The recommended value of β for the CLR is 0.05 while all other receivers are recommended to use β = 0.5 due to their less infrequent RTT measurements.

CHAPTER 4

SIMULATION WORK

4.1 Validation of the code of LESBCC for NS

To the best knowledge of the author, a well-defined, fully stated and reproducible simulation experiment does not exist in the congestion control protocol literature encountered so far. Most experiments lack the necessary details and some that are mandatory for reproduction. Unfortunately, the experiments conducted in the original LESBCC paper are also of this type. Figure 8 is a graph taken from this original source and its corresponding experiment is conducted using the LESBCC code written in this thesis.

The topology is presented in Figure 7. Packet size is 560 bytes. Link delays are all 10 ms. Queue sizes of all routers are 125 packets. Droptail policy is used in queues. Each link from the senders to router 1 has a bandwidth size of 6 Mbps. The link between routers has a bandwidth of 6 Mbps for each flow (totally 102 Mbps). Each link from router 2 to the receivers has a bandwidth size of 5 Mbps. "*Main Source*" is the LESBCC source that sends data to all four receivers. Set *i* has four TCP sources that sends data to Receiver *i* for *i* = 1,2,3,4.

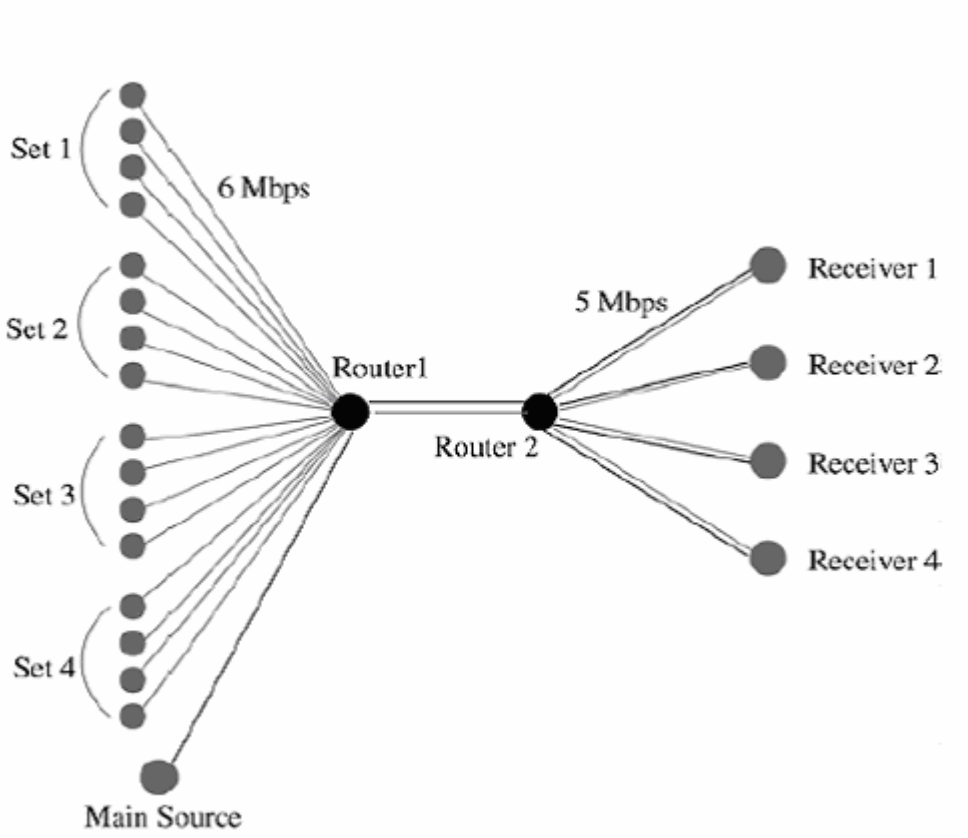

Figure 7: Validation experiment topology

The corresponding experiment is chosen for validation purposes because it is the one in the paper with the most details and parameter information. In this setup, there exist some TCP and LESBCC parameters that can affect the behavior illustrated on the corresponding graph. Some of these parameters, such as the initial SRTT and the initial sending rate, may affect the transient behavior and some others, such as the TCP window size and the receivers' random backoff

time used as a delay before sending a feedback to the source, may effect the steady state behavior. However, these are lacking parameters.

In this verification step, we assume that the TCP receiver window size is large enough and does not limit the TCP's behavior. For some experiments in the LESBCC paper, multicast receivers delayed the feedback up to 20% of the initial RTT. In our experiments, none of the receivers use a delayed feedback.

The result of the verification run is presented in Figure 9. We believe that the results are similar comparing the general characteristics and given the lack of various parameters the written code can be accepted as valid to a certain degree.

Our NS implementation is given in Appendix A whereas the algorithm of LESBCC is given in Appendix: Pseudo code in the original paper [Thapliyal 2002] for further comparisons.

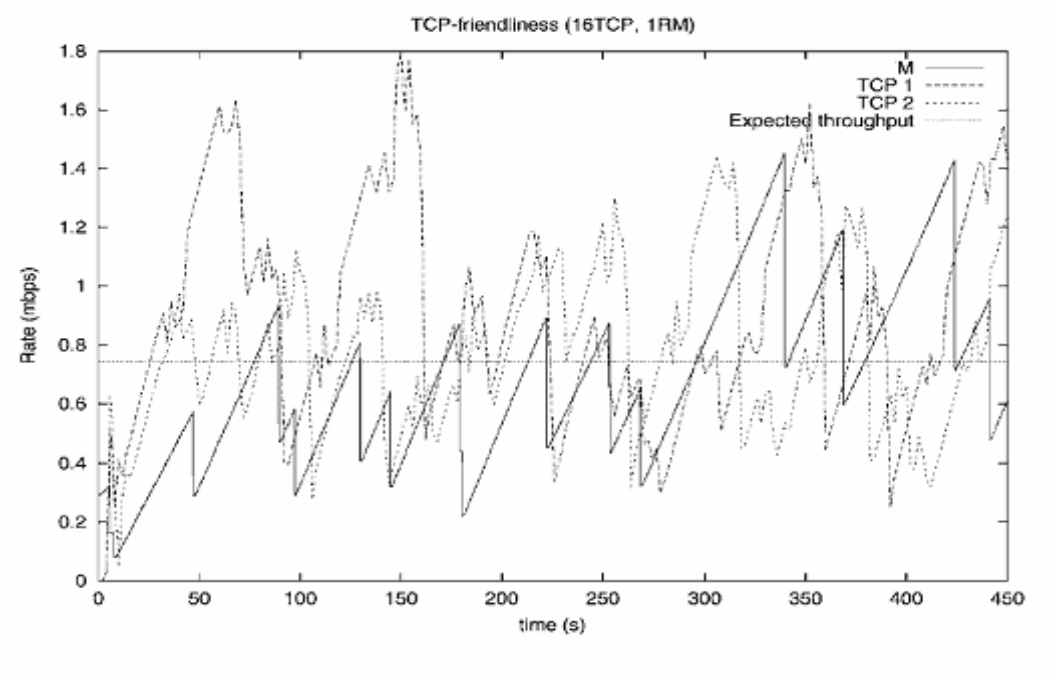

Figure 8: Average Throughput of LESBCC vs. TCP

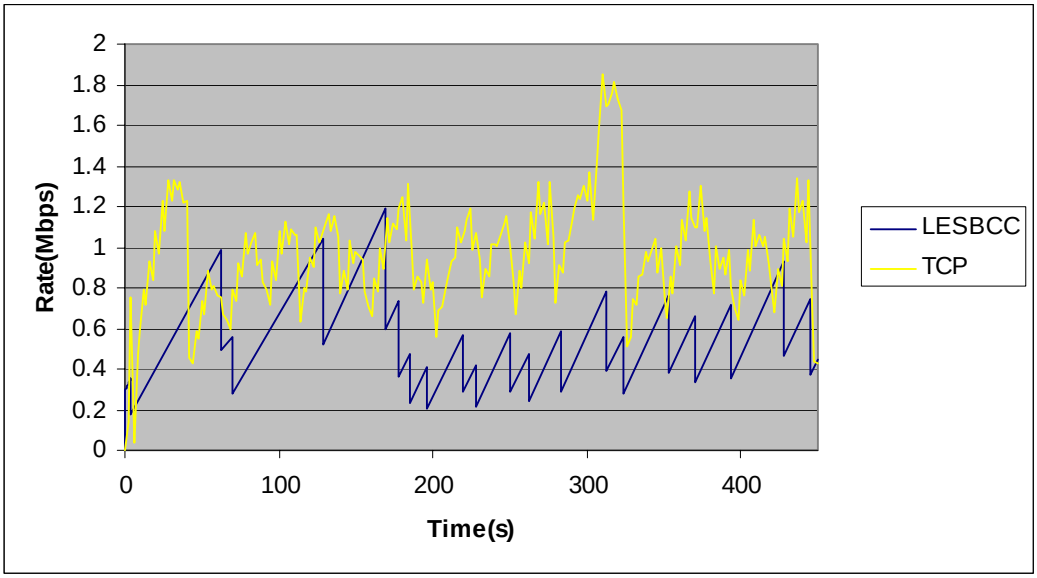

Figure 9: Average Throughput of TFMCC vs. TCP

4.2 Experiments

The purpose of the experiments in this chapter is to empirically evaluate the performances of LESBCC and TFMCC and to compare them with each other using the simulation tool, NS-2. In each experiment, we analyze the TCP-friendliness, smoothness and responsiveness of the corresponding protocols.

A framework has already been proposed for the simulation and testing of multicast congestion control schemes [Handley 2000]. In the design of our experiments and in the choice of the topologies, this framework is used to the greatest possible extent. The source model used in the simulations is the File Transfer Protocol (FTP) with a packet size of 1400 bytes. Unless otherwise specified, all links are lossless, each link has a propagation delay of 1ms and a bandwidth of 10 Mbps. The TCP version used in the experiments is Reno since it is the most widely deployed version. Droptail queues are used at the routers to mimic the behavior of the current Internet. The TCP window size is set to 1000 packets so that it is large enough not to impact its own congestion control mechanisms. The clock granularity of TCP is set to 10ms.

Correct queue sizing is an important issue for all the experiments. Queue sizes on the non-bottleneck links are not important since there will be no buffering. However there is no consensus on how to determine the bottleneck queue size

before the experiments. TCP needs the network to buffer an RTT worth of data to use the full capacity and hence the aim is to allow the TCP to fill the pipeline from source to destination and use the link efficiently. As a rule of thumb, the queue size is recommended as at least 4*BW*RTT (in bits or packets) where BW is the bandwidth size of the bottleneck (in bps or pkt/s) and RTT is the fixed round trip time (in seconds) including only the link delays [Handley 2000].

4.2.1 Experiment 1: Shared Loss

The topology shown in Figure 10 is used in the first experiment. We have a TCP session competing with a multicast session over a bottleneck link. The bottleneck link A, between routers, has a propagation delay of 50 ms and its bandwidth is 500 Kbps. The multicast session has one sender and three identical receivers.

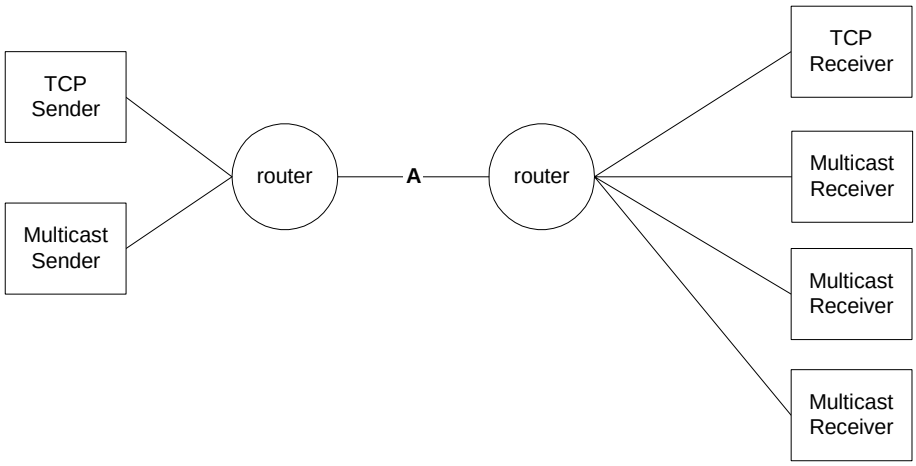

Figure 10: Multicast session competing with TCP over a bottleneck link

According to the rule of thumb, the queue that feeds link A should have a size of at least 19 packets and a queue size of 30 packets is selected for this experiment with some safety margin.

In Figure 11, the throughputs of both LESBCC and TCP are observed to be very close to each other. F is at most 1.14 throughout the experiment so LESBCC can be said to be TCP-friendly. As was mentioned in Chapter 2: Intersession Fairness, the TCP-friendliness constant $F=T_M/T_{TCP}$ should be less than 1.19, where T_M is the throughput of the multicast session, T_{TCP} is the throughput of the TCP session on the bottleneck link. All packet drops are due to the congested link so in multicast session they are of type "shared loss" for all the multicast receivers. Hence, the sender receives 3 NACKs for each packet drop but the LESBCC multicast sender seems to filter the excessive NACKs successfully.

In Figure 11, TFMCC is also observed to be TCP-friendly and F is at most 1. Actually, it is perfectly TCP-friendly. Therefore both approaches are suitable to be used in the existence of bottleneck links from TCP fairness point of view.

TCP-friendliness definition does not cover the performance issues. As was discussed earlier, the inverse of F can be used for the performance criteria. TFMCC cannot get a fair share bandwidth, i.e., TCP's throughput is more than 1.19 times TFMCC's throughput for 400 seconds. LESBCC converges

approximately three times faster than TFMCC. This is an expected result because LESBCC's additive constant of AIMD module is 1. According to [Floyd 2000] TFMCC is designed to increase its rate at most 0.28 packets per RTT to achieve the desired smoothness. In general, the smoother a congestion control algorithm, the less responsive it is.

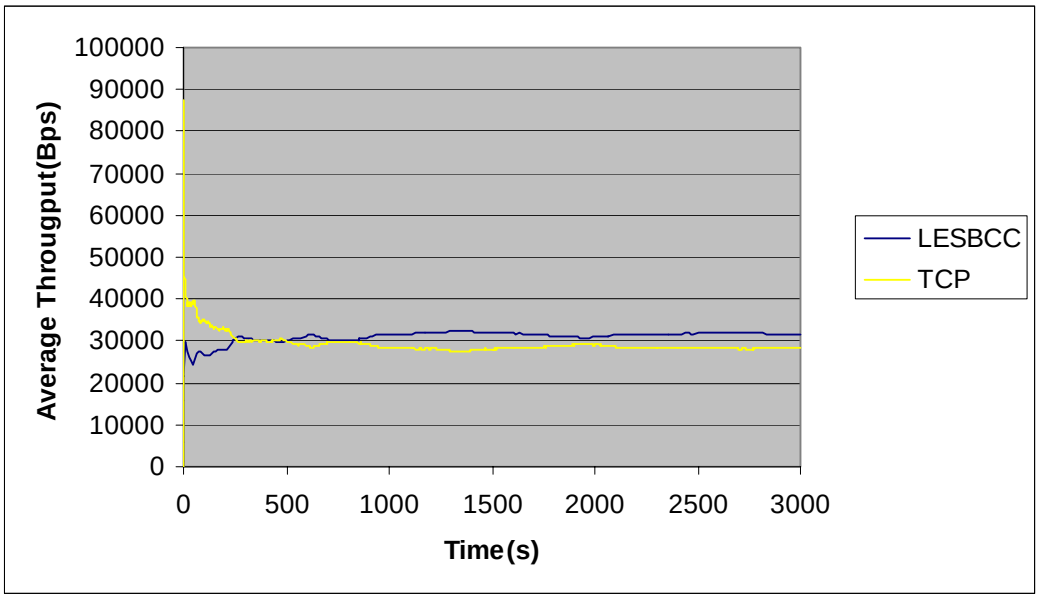

Figure 11: Average Throughput of LESBCC vs. TCP

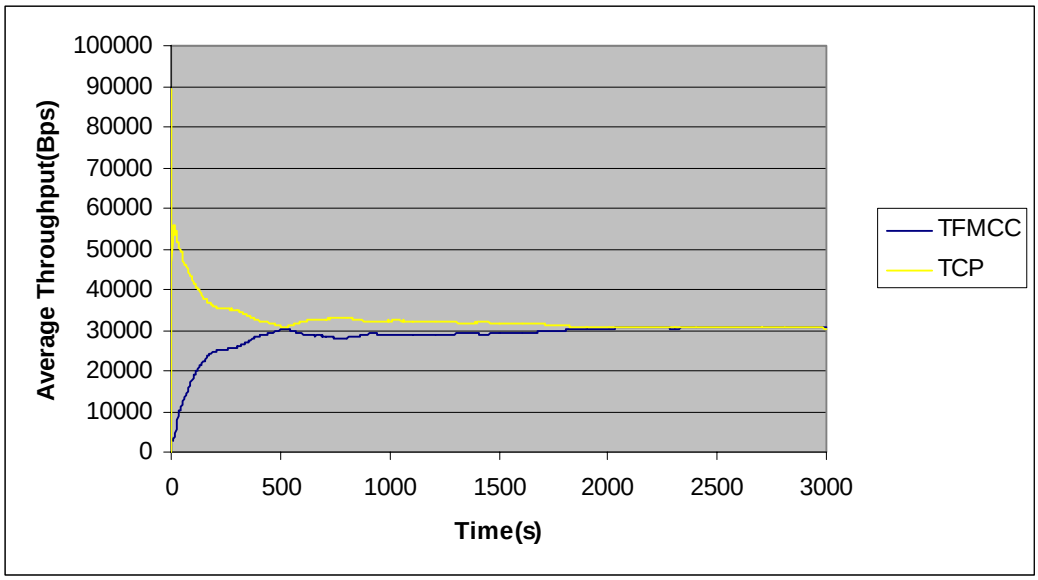

Figure 12: Average Throughput of TFMCC vs. TCP

For short duration connections, a TFMCC multicast stream, although it is TCP-friendly, cannot get a fair share of the bandwidth and its throughput is observed to be much less than the TCP. It catches the TCP performance level only in the long run.

Figure 13 shows the instantaneous throughput comparison of LESBCC vs. TCP for the last 500 seconds of the simulation. It shows that LESBCC eliminates the drop-to-zero problem. LESBCC increases its rate at those times that TCP Reno decreases its rate to zero but this transient behavior does not lead to unfairness. This transient behavior is inherent to TCP Reno. Most TCP implementations decrease their window size to the initial value in case of a timeout. This initial value is changing according to the implementation but it is approximately 1 or 2 packets which is a very low value. However, TCP Reno also has to wait for a

timeout to recover from the multiple losses per window. This disadvantage makes TCP Reno drop to zero more than necessary. Different TCP versions such as NewReno, SACK, Vegas are proposed to overcome this problem. As they are not widely deployed however, the simulations on congestion control world are still recommended to be performed using TCP Reno [Rizzo 2001] [Widmer 2001].

In Figure 14, it is observed that TFMCC keeps its rate even more stable than LESBCC. It does not drop to zero either. This is an expected result because one of the main goals of TFMCC is to keep the rate change low to be suitable for applications such as audio/video traffic. Observe that TCP is also smoother in Figure 14 than in Figure 13. This can be explained by comparing the burstiness. The nature of LESBCC traffic is more aggressive, responsive and bursty than TFMCC traffic, which also affects TCP to be bursty.

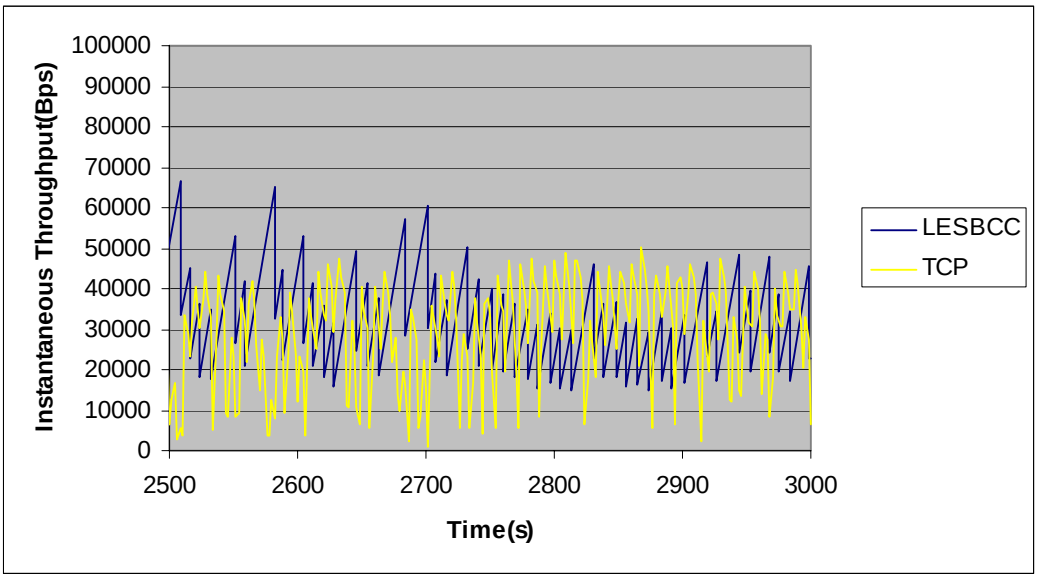

Figure 13: Instantaneous throughput comparison of LESBCC and TCP

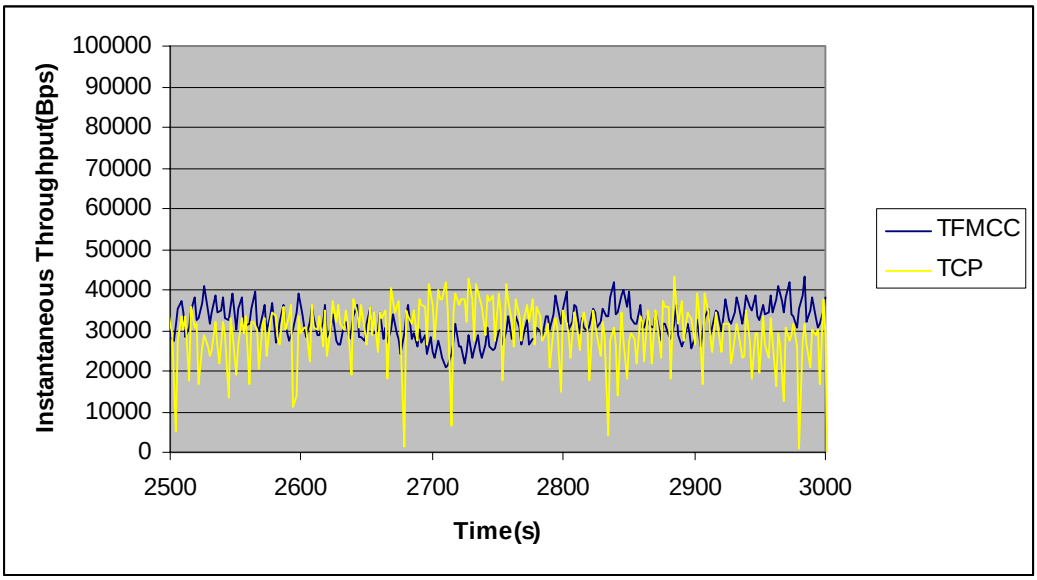

Figure 14: Instantaneous throughput comparison of TFMCC and TCP

4.2.2 Experiment 2: Independent Loss

It is worth noting that in the previous experiment all NACKs are generated almost at the same time. In experiment 2, the topology given in Figure 15 is used so that the loss generation times are independent from each other. One multicast session is competing with two TCP sessions. For Link A and Link B, the propagation delays are 50 ms and the bandwidths are 500 Kbps.

In this experiment losses occur near the receivers and effect only the corresponding receiver which is called independent loss in the multicast literature. Queue size is set to 30 packets for the bottleneck links for Figure 16 and Figure 17.

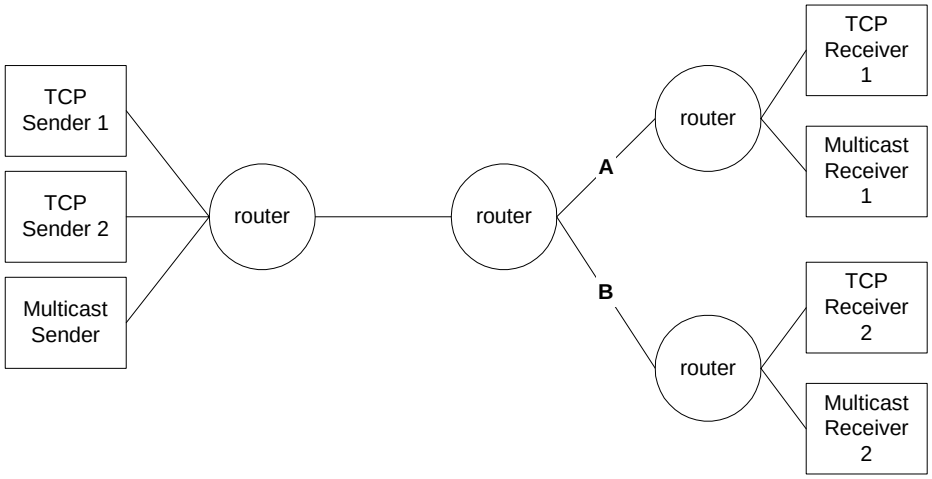

Figure 15: Multicast session competing with TCP where losses are independent.

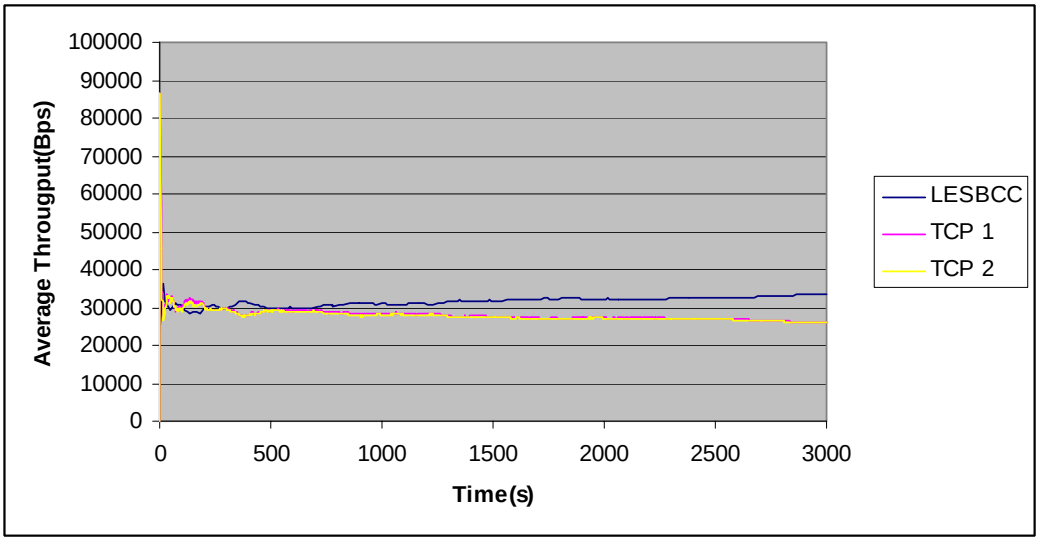

Figure 16: Throughput of LESBCC vs. two TCP sessions (queue size=30pkts)

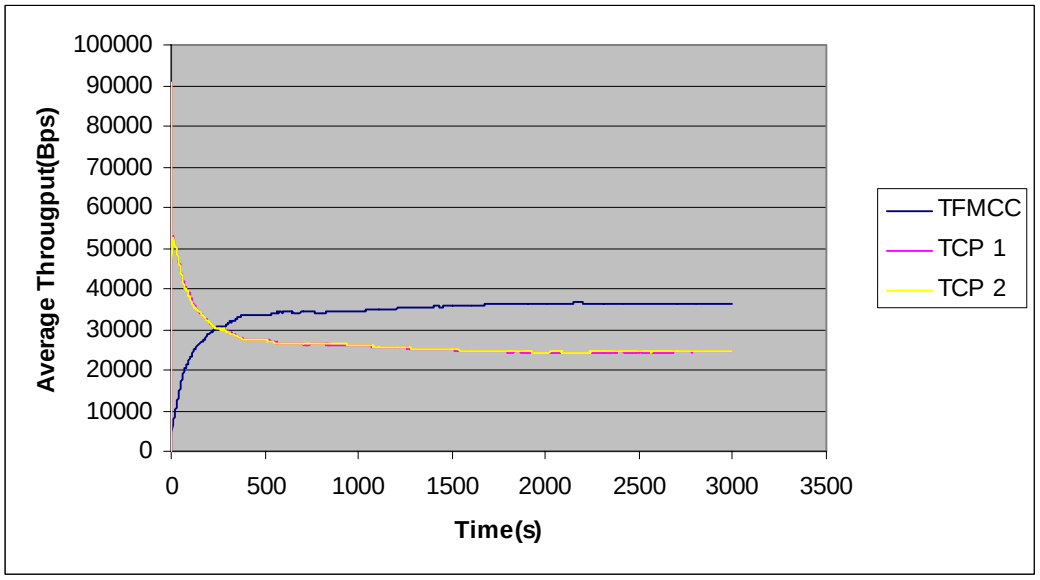

Figure 17: Throughput of TFMCC vs. two TCP sessions (queue size=30pkts)

LESBCC is starving TCP as time passes in Figure 16. Also in Figure 17 TFMCC receives much more bandwidth than TCP. F is around 1.5 for the last 500 seconds. Before making comments on these results, we repeat this experiment

when queue sizes are equal to 50 packets and we observe the results in Figure 18 and Figure 19.

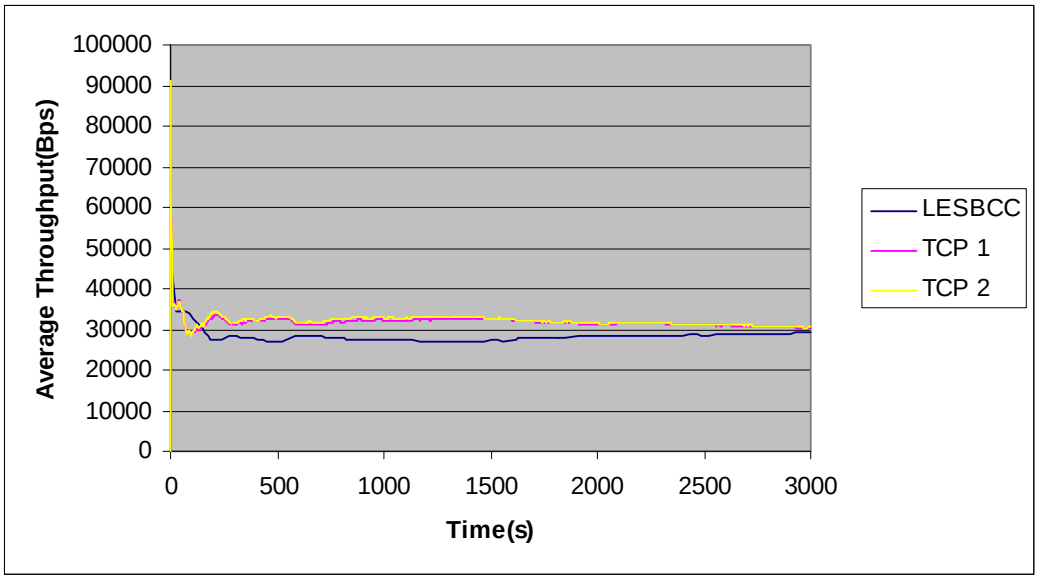

Figure 18: Throughput of LESBCC vs. two TCP sessions (queue size=50pkts)

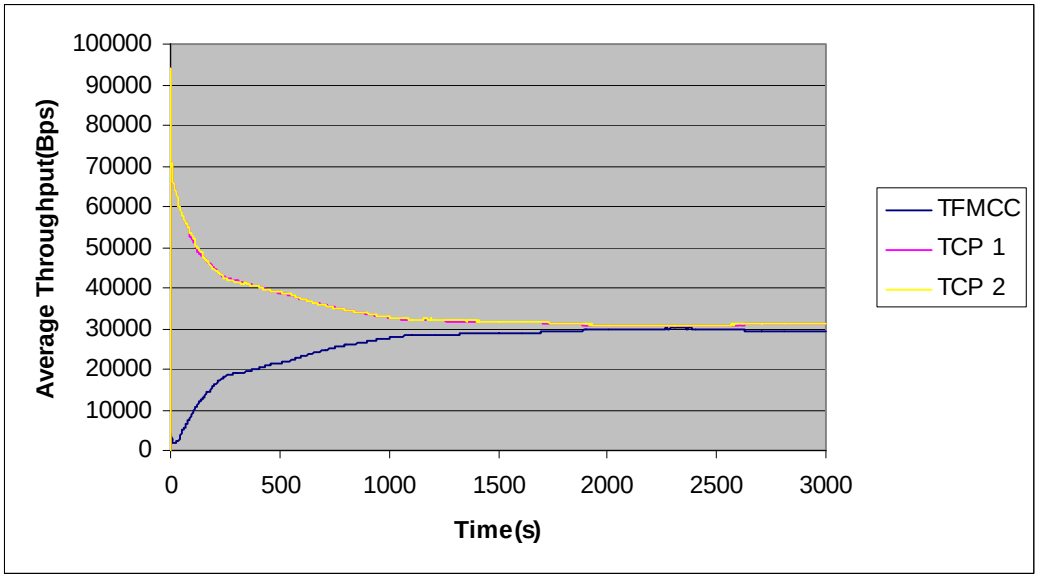

Figure 19: Throughput of TFMCC vs. two TCP sessions (queue size=50pkts)

Both LESBCC and TFMCC becomes TCP-friendly when we increase the bottleneck queue size. The reason for this lies in the spacing of the data packets and buffer requirements: Both TFMCC and LESBCC spaces out the data packets, while TCP sends them back-to-back if it can send multiple packets making TCP more sensitive to nearly-full queues [Widmer 2001]. A queue size of 30 packets is not enough for three sources hence we need more buffer space in this case. Multiple losses per window decrease the robustness in TCP Reno [Handley 2000]. Selecting queue sizes larger decreases the multiple losses per window and increases the TCP throughput.

Nevertheless, end users cannot determine the queue sizes on routers. Both protocols should be TCP-friendly in small queue sizes too. In the current experiment, we can say that, both protocols starve TCP in small queue sizes and TFMCC is more sensitive to small queues for the TCP-friendliness metric but is more TCP-friendly with an appropriate queue size.

For the performance issue, LESBCC's performance metric P is larger than the desired value, 1.19, throughout the experiment. But TFMCC reaches this value only after 1000 seconds. This also shows that LESBCC can use the available bandwidth more efficiently and timely than TFMCC.

Comparing the last 500 seconds instantaneous throughput graphs given in Figure 20 and Figure 21, comparable data sending rates are observed for both protocols.

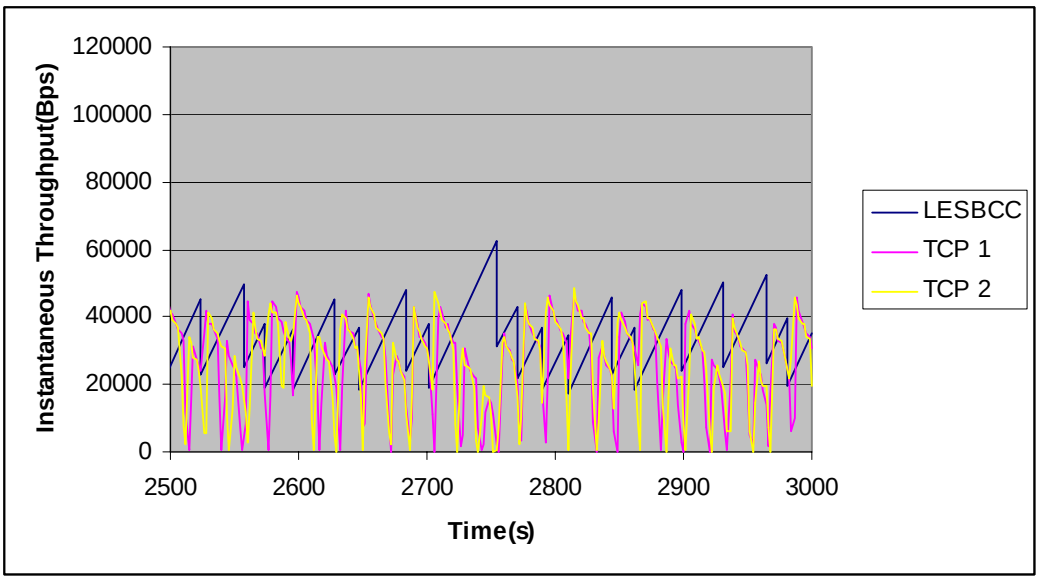

Figure 20: Instantaneous throughput of LESBCC vs. two TCP sessions (queue size=50pkts)

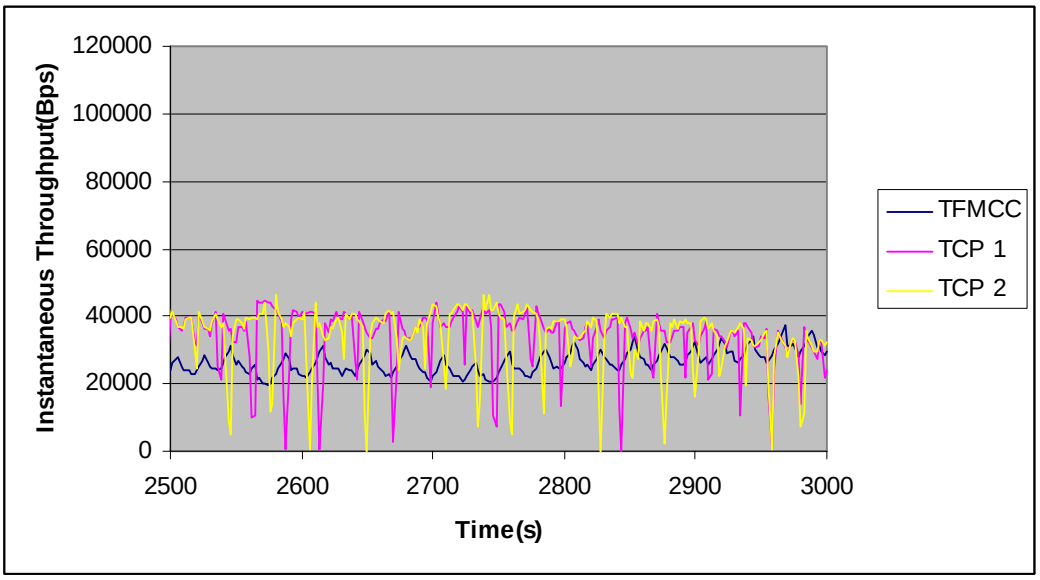

Figure 21: Instantaneous throughput of TFMCC vs. two TCP sessions (queue size=50pkts)

4.2.3 Experiment 3: Different Bandwidths

In this experiment, the topology in Figure 15 is used again but link B has a bandwidth of 1 Mbps this time. So TCP session 2 throughput is expected to be twice that of TCP 1. In this case, the multicast flow should not increase its rate while observing free bandwidth in link A. Queue size is 50 for link A as usual and 100 for link B as its bandwidth-delay product is twice that of link A.

Analyzing Figure 22 and Figure 23, running the experiment for 2000 seconds is enough to see that both multicast protocols are TCP–friendly. Their transmission speeds are successfully determined by the slowest receiver. Again LESBCC responds much faster than TFMCC as in the previous experiments.

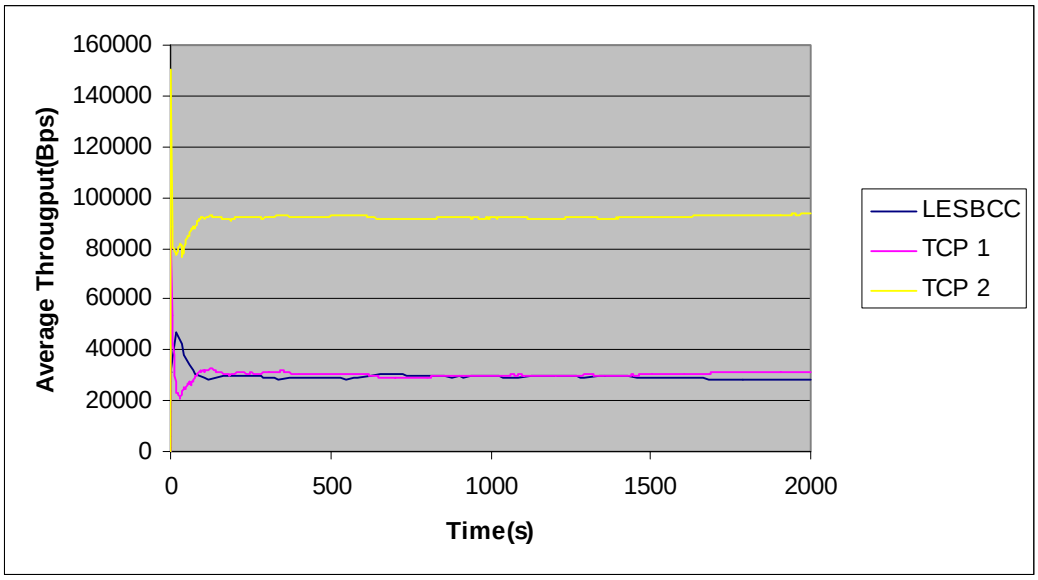

Figure 22: Throughput of LESBCC vs. two TCP sessions on different bandwidths

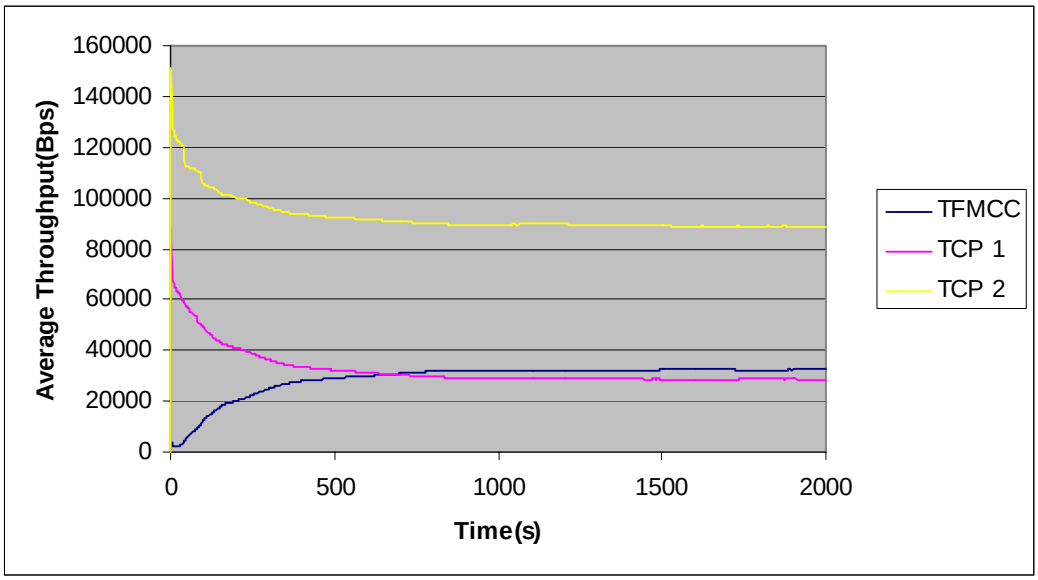

Figure 23: Throughput of TFMCC vs. two TCP sessions on different bandwidths

Inspecting the instantaneous throughputs in the last 500 seconds, we observe that the bandwidth difference between the receivers does not affect the behaviors of the protocols.

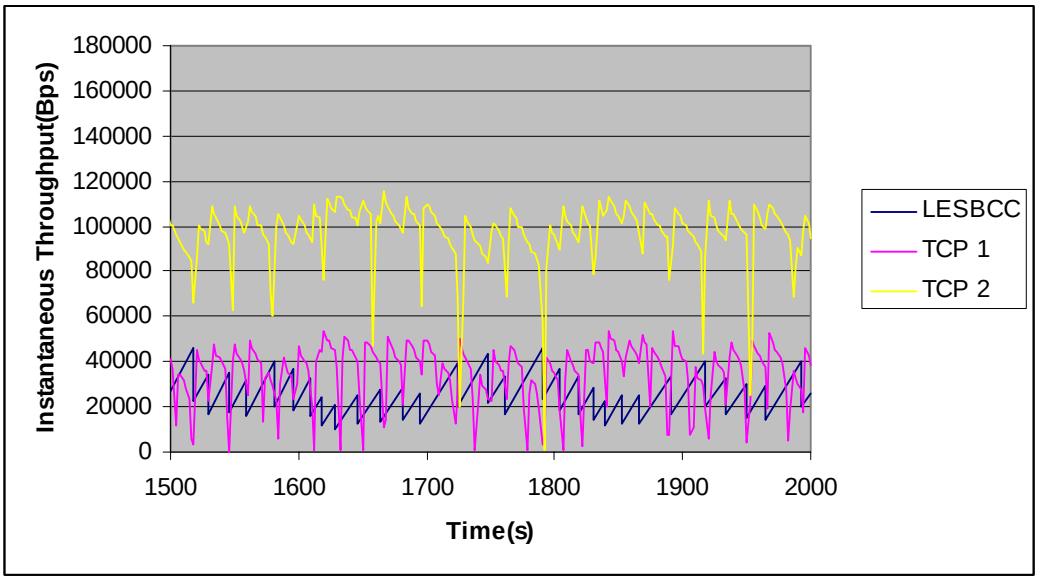

Figure 24: Instantaneous throughputs of LESBCC vs. two TCP sessions

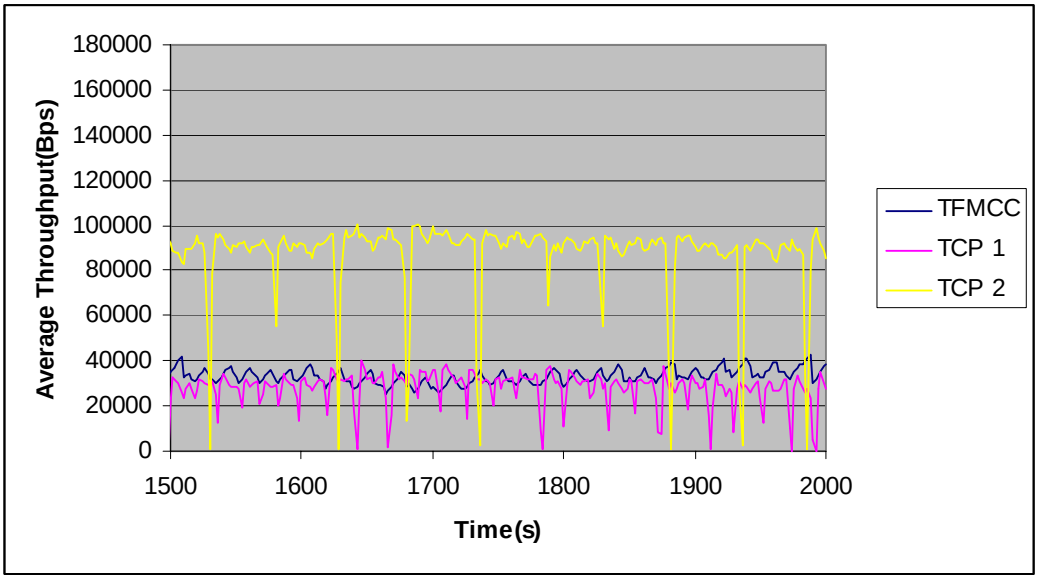

Figure 25: Instantaneous throughputs of TFMCC vs. two TCP sessions

4.2.4 Experiment 4: One far receiver

For the previous experiments, all receivers have the same propagation delay to the sender. In experiment 4, we use the topology given in Figure 26. Link A has a bandwidth of 500 Kbps and propagation delay of 10 ms. Link B has a bandwidth of 10 Mbps and propagation delay of 50 ms. As the bottleneck is nearer to the sender than the receivers, losses are all shared. But the sender receives the NACKs from the third receiver later than the other receivers for the same packet drop. This phase effect should not change the behavior of a multicast congestion control protocol.

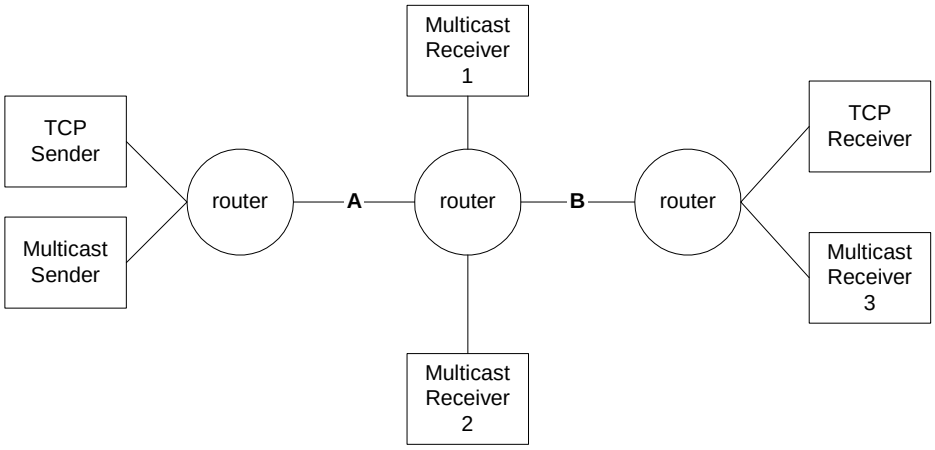

Figure 26: Multicast session having receivers with different RTTs.

There are two senders as in Experiment 1. From the given topology, a queue size of 30 packets should again be sufficient. When the experiment is carried out with

a queue size of 30 packets, both multicast protocols are observed to perform twice as fast as TCP. To understand the situation, post simulation RTT values are analyzed. Figure 27 shows the SRTT value calculated by the TCP sender. SRTT is 0.42 seconds for the steady state and CRTT is at most 0.57 seconds. Bottleneck bandwidth is 500Kbps. So BW*RTT is 20 packets at steady state and is at most 25 packets throughout the experiment for a healthy TCP connection. Therefore, we can say that the RTT estimation module in both protocols does not work well in this case. In other words, receivers nearer to the sender affect the RTT estimations of the protocols and cause the sender send faster than the required rate.

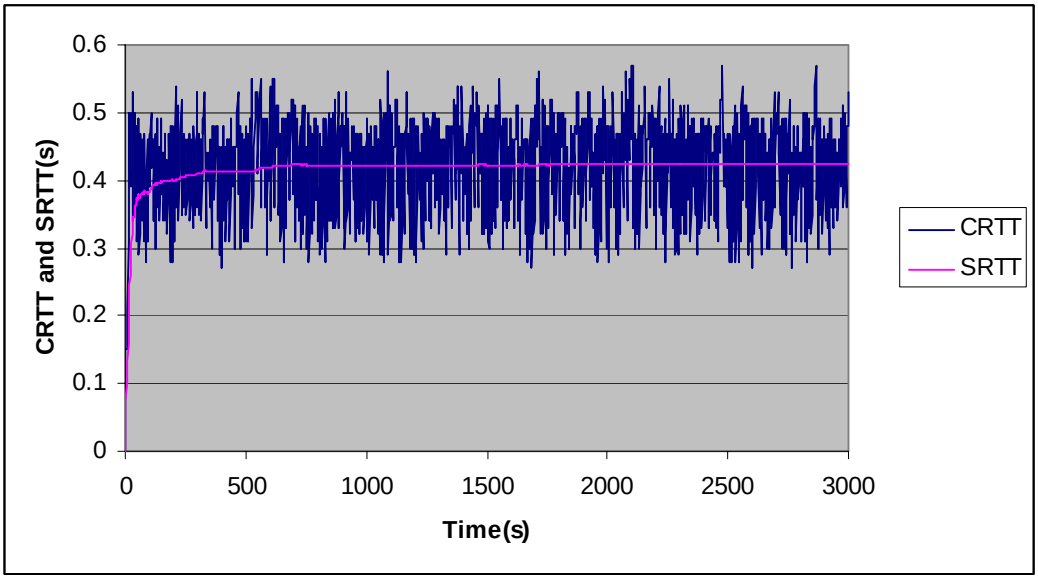

Figure 27: CRTT and SRTT value calculated by TCP (queue size=30packets)

The same experiment is then repeated using a queue size of 50 packets. Both protocols estimation modules perform better in this case, i.e. when the queue size

is larger. In Figure 28, the experiment is extended to run for 4000 seconds to see that TFMCC is increasing its sate slowly but in a stable manner. At the end of the simulation, F is 1.26 and increasing.

In Figure 27, LESBCC becomes 1.27 times that of TCP around 1600th second. End of the simulation, the F value observed is 1.24 and it never falls below 1.19.

For the estimation module in LESBCC, if RTT sample has a value less than c times the calculated SRTT value it is not included in the calculation of SRTT *"to bias the average RTT higher"* [Tahpliyal 2002], (i.e. if RTT<c*SRTT then ignore). This constant is 0.5 in the original paper. And for the TFMCC, the β constants (0.5 and 0.05 in the original paper), mentioned in Chapter 3 TFMCC section, can be adjusted. The adjustment of these two values is an empirical approach and they need to be adjusted carefully in a wide range of topologies.

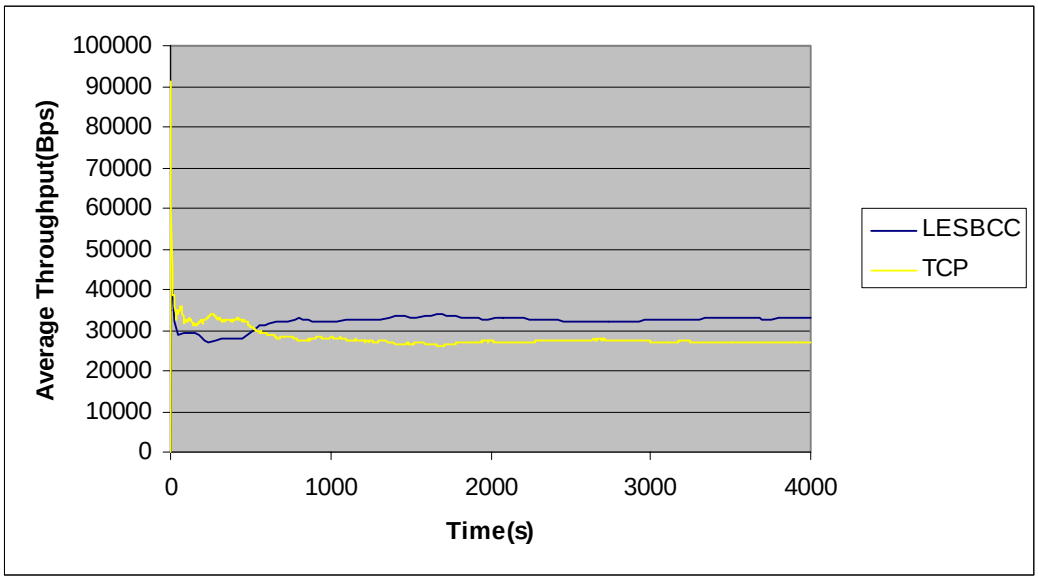

Figure 28: Throughput of LESBCC vs. two TCP sessions

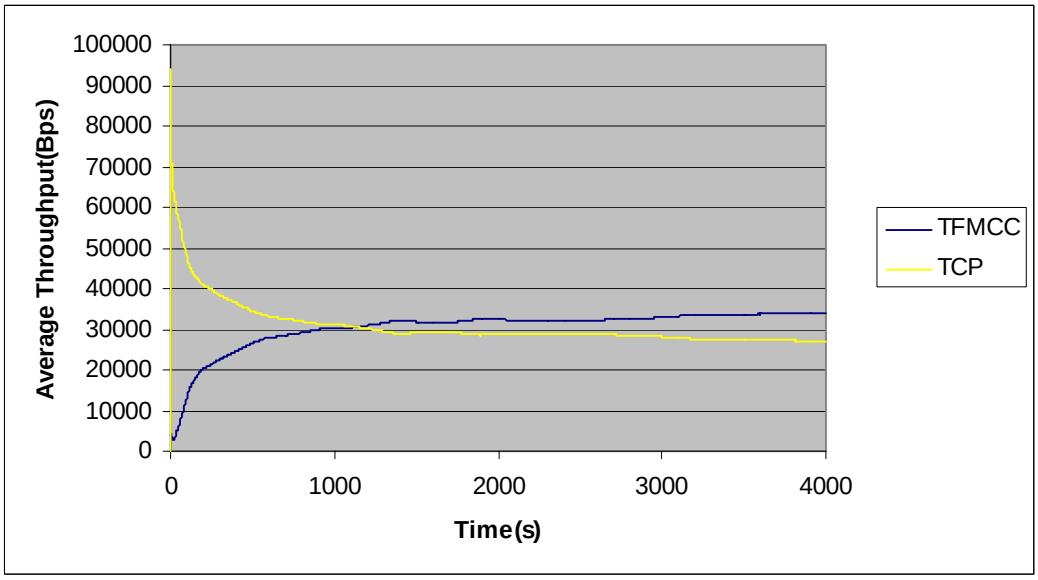

Figure 29: Throughput of TFMCC vs. two TCP sessions

4.2.5 Experiment 5: Smooth LESBCC (sLESBCC)

LESBCC has an AIMD module that is not designed for smooth delivery. It imitates the behavior of the AIMD module in TCP. It increases the sending rate by one packet size per RTT when there is no congestion. In case of congestion it halves the data sending rate. This is called AIMD(1,1/2). As was mentioned in Chapter 3: Extension to AIMD module in LEBSCC, there are different AIMD schemes, which are claimed to be smoother while being TCP-friendly. It is recommended to use 1/5, 2/5 and 0.31 instead of 1 for the additive constant and 7/8 instead of 1/2 for the multiplicative constant. All three options have been tried in our simulations and it is observed that AIMD(1/5,7/8) is the best choice for LESBCC.

Figure 30 shows the result of experiment 1 carried out with sLESBCC. It responds slower similar to TFMCC. Its TCP-friendliness and performance characteristics are similar to LESBCC, i.e., between the required bounds. F does not exceed 1.17 throughout the experiment.

However, sLESBCC responds much slower than TFMCC. This can be explained as follows: TFMCC can change its rate increase speed between 0.14 and 0.28 packets per RTT according to the network conditions [Floyd 2000]. LESBCC can increase its speed 0.2 per RTT when there is no congestion. TFMCC is expected

to respond approximately one and half times faster than sLESBCC, which is also proven by our simulations. But TCP-friendliness and performances are very similar on steady state conditions. Testing sLESBCC with AIMD(0.31,7/8) responds as TFMCC but it is TCP-friendliness is not satisfying. We prefer a slower but a TCP-friendly version, which is maybe the most important requirement from a multicast congestion control protocol point of view. To quote [RFC 2357]:

"…congestion control mechanisms that operate on the network more aggressively than TCP will face a great burden of proof that they don't threaten network stability."

More for the performance and TCP-friendliness debate can be summarized as follows: performance needs can change according to the needs of the end-users but TCP friendliness is a factor that threatens the network stability.

Figure 31 proves our previous statements. Smooth LESBCC is really smoother than LESBCC. Also TCP flow running with sLESBCC is smoother as the burstiness of the system decreases. Smooth LESBCC does not drop to zero either.

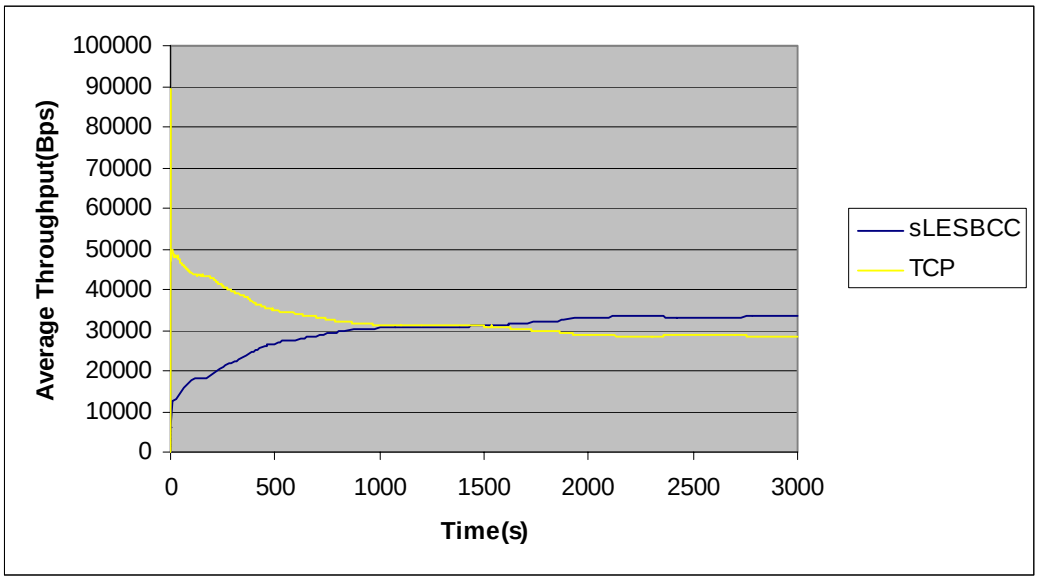

Figure 30: Experiment 1 using smooth LESBCC

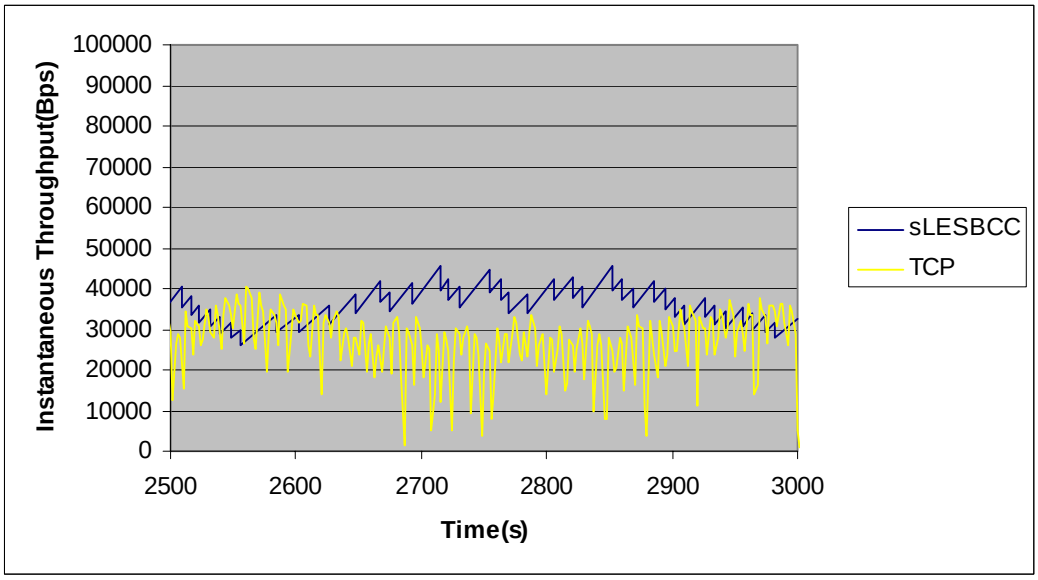

Figure 31: Experiment 1 using smooth LESBCC

For experiment 2, sLESBCC is again TCP-friendly (Figure 32 and Figure 33). Its performance is in the acceptable bounds. P is 1.1 at the end of the simulation.

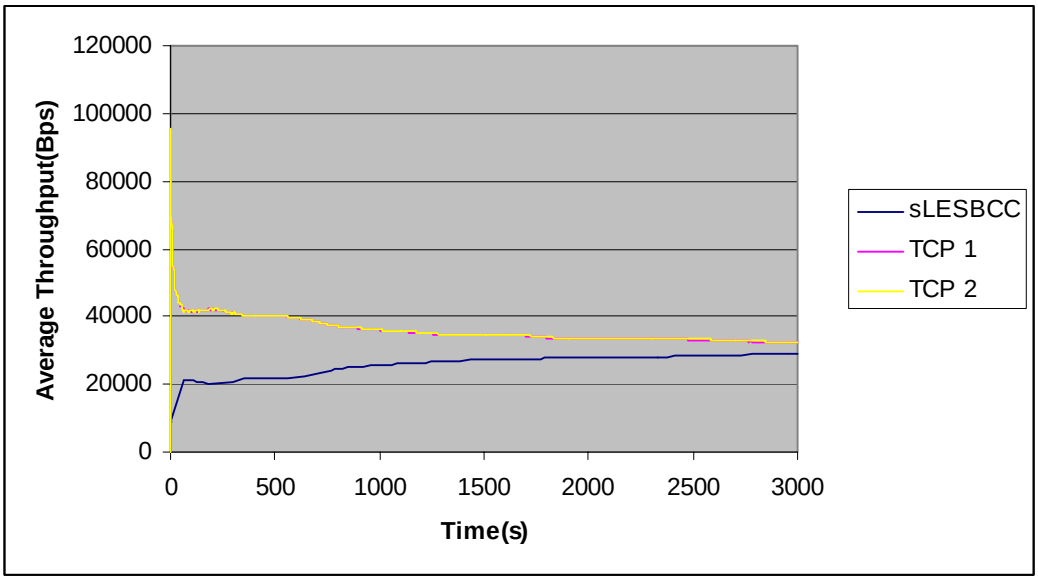

Figure 32: Experiment 2 using smooth LESBCC

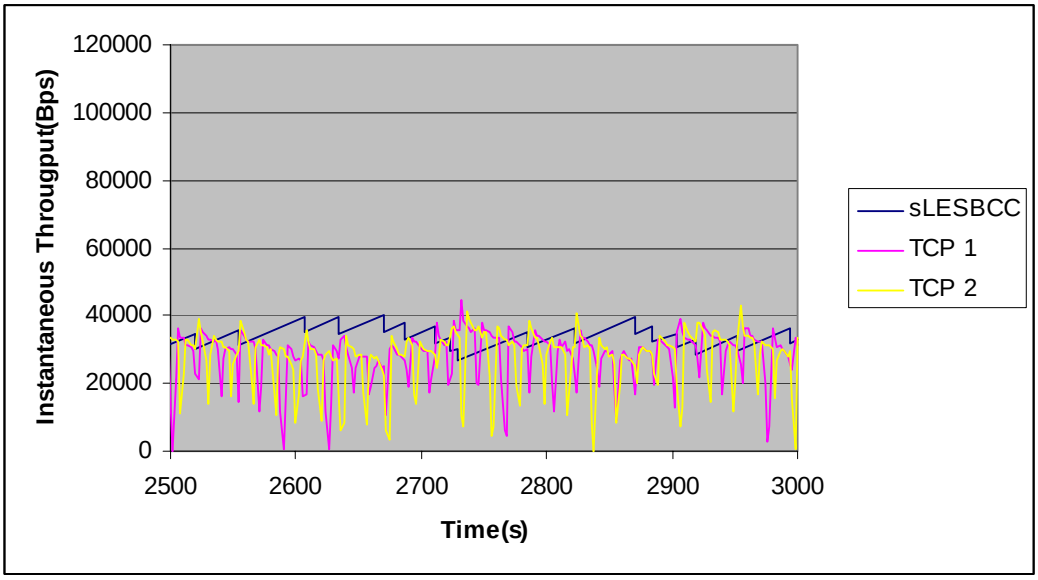

Figure 33: Experiment 2 using smooth LESBCC

In experiment 3, sLESBCC performs well. Its rate is successfully determined by the slowest receiver. It is much smoother than LESBCC. It has all the desired properties for this case (Figure 34 and Figure 35).

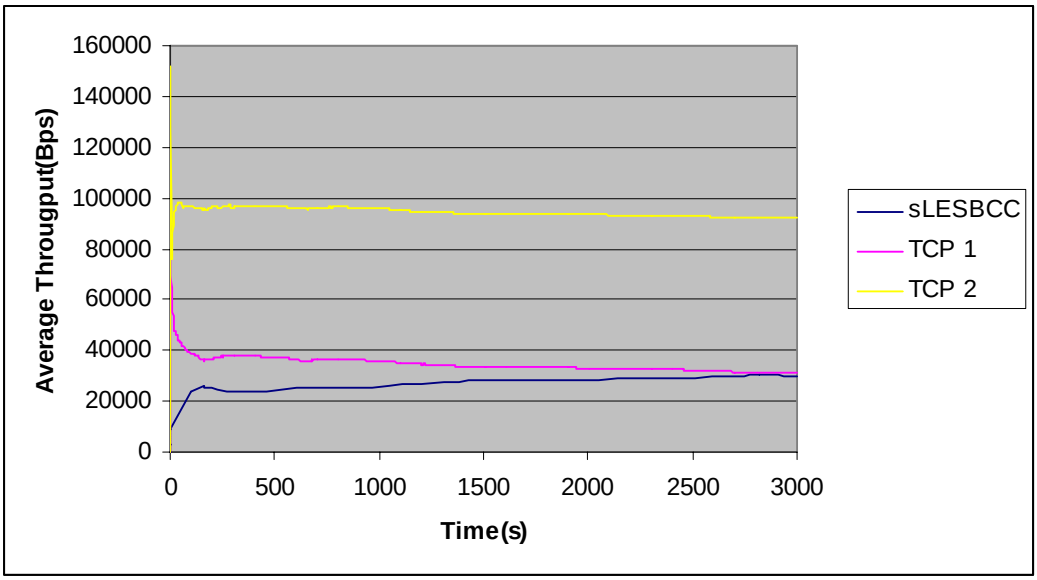

Figure 34: Experiment 3 using smooth LESBCC

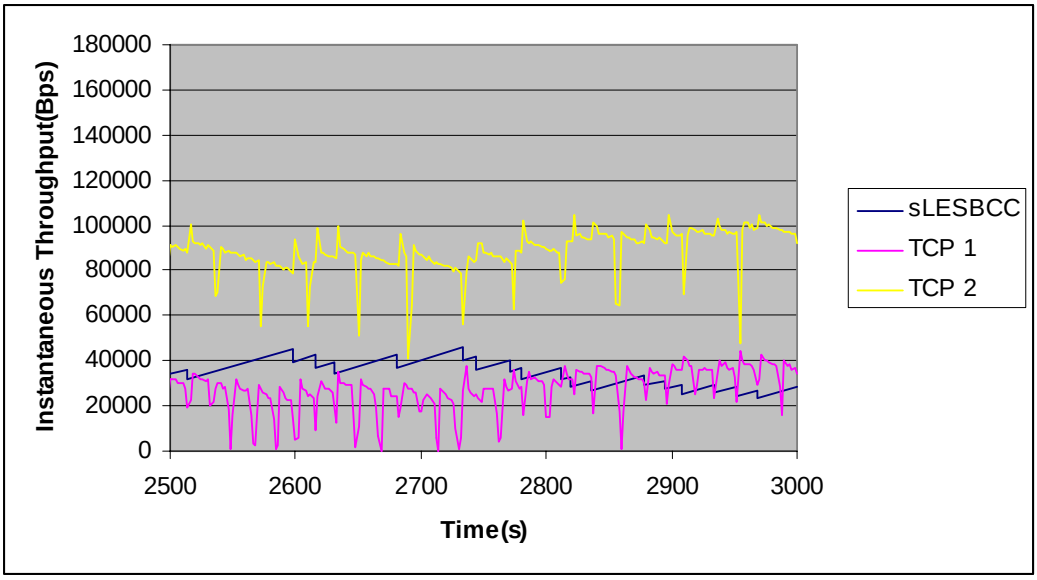

Figure 35: Experiment 3 using smooth LESBCC

Repeating experiment 4 with sLESBCC gives an expected result. In the RTT estimation module there is no difference with LESBCC. sLESBCC starves TCP in this topology as TFMCC and LESBCC (Figure 36). The better adjustment of the constants gives better results in LESBCC and sLESBCC. As a last word for this experiment; even if none of the protocols are TCP-friendly well enough, they respond to congestion. Using any of them will be a much better choice for the network than using an unresponsive flow such as UDP.

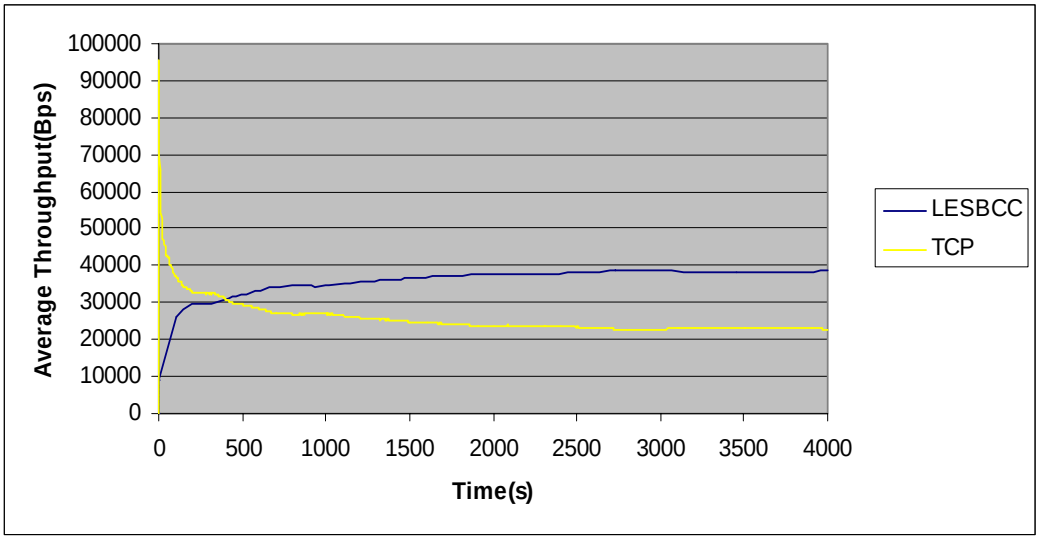

Figure 36: Experiment 4 using smooth LESBCC

CHAPTER 5

CONCLUSION

Multicast is slowly developing from experimental areas to a service on the Internet. A new network layer protocol is needed to achieve a scalable and effective multicast. There exist many standard protocols in use for the routing and the management of a multicast session. Unfortunately, UDP is used at the transport layer, which causes the multicast session not to be friendly to other sessions. These kinds of approaches are not responsive to variations in the network conditions. Still, there is not any standard protocol deployed at the transport layer for multicast that behaves fairly to other sessions while meeting the needs of the application layer at the same time. As a result, congestion control remains to be one of the fundamental and challenging problems of Internet multicast.

In this thesis, first, the congestion control problem is defined. Its history and the main approach to control the congestion for unicast sessions is given. The challenges are listed and then a literature survey is given to overview the current solutions to the congestion control problem in the multicast domain.

Then two of the recent solutions are focused on and detailed simulations are carried out to verify their fairness characteristics and to compare their relative performances. Both protocols are applicable as a remedy for the urgent need in this area.

The contributions of the thesis are twofold. One is the implementation of the LESBCC protocol and its addition to the NS protocol stack and the second being the evaluation and detailed comparison of LESBCC with the existing multicast protocol TFMCC.

TFMCC is widely accepted as a multicast congestion control protocol. It belongs to the "single-rate, end-to-end, rate based" class of protocols. LESBCC is proposed later but the question whether it is a successful alternative to TFMCC or not remained to be open. The present work tests and compares LESBCC in detail with TFMCC with respect to their TCP-friendliness, smoothness and responsiveness criteria.

In this study it is observed and concluded that long-term throughput of TFMCC exceeds TCP in a topology, where the distance of the receivers to the source vary considerably.

Simulation study reveals the relative performance of LESBCC compared to TFMCC. It is concluded that LESBCC is TCP-friendly and its transient period is

three times shorter than TFMCC in many cases. Hence, LESBCC should be used instead of TFMCC for those cases where the smoothness of the data delivery is not important. Main drawback of LESBCC is its AIMD module that behaves similar to TCP. Therefore, those applications performing badly with TCP cannot run satisfactorily using LESBCC either.

There are already some recommended adjustments for the AIMD module in TCP to make it smoother. In this study, we observed that AIMD(1/5, 7/8) is the most appropriate choice for LESBCC. When implemented, LESBCC changes its throughput smoothly similar to TFMCC and this smoothed scheme does not affect the TCP-friendliness, which was already proven analytically in [Floyd 2000] for unicast. With these new AIMD parameters, LESBCC transient period becomes worse and approximately 1.5 times longer than TFMCC.

As future work, more protocols can be studied. Especially, the multicast multirate solutions should also be investigated since they are more scalable and fairer to the receivers but of course at the cost of complexity.

REFERENCES

1. Adams A., Nicholas J., Siadak W., ”Protocol independent multicast dense mode (PIM-DM)”: Protocol specification (revised)”, Internet draft, draft-ietf-pim-dm-new-v2-01.txt (February 2002)

2. Assuncao P., Ghanbari M., “Multi-casting of MPEG-2 video with multiple bandwidth constraints,” in *Proc. 7th Int. Workshop Packet Video*, Brisbane, Australia, pp. 235–238. (March 1996)

3. Bhattacharyya S., Towsley D., Kurose J., “The loss path multiplicity problem in multicast congestion control,” in Proc. IEEE INFOCOM, New York, pp. 856–863 (March 1999)

4. Bhattacharyya S., Towsley D., Kurose J. “A novel loss indication filtering approach for multicast congestion control” Computer Communications 24(5-6) pp. 512-524 (2001)

5. Braden B. et al., “Recommendations on queue management and congestion avoidance in the internet,” Network Working Group, RFC 2309, (April 1998)

6. Byers J., Luby M., Mitzenmacher M., “Fine-grained layered multicast,” in *Proc. IEEE INFOCOM*, AK, pp. 1143–1151 (April 2001)

7. Cain B., Deering S., KouvelasI., Thyagarajan A., “Internet group management protocol, version 3”, Internet draft, draft-ietf-idmr-igmp-v3-09.txt (January 2002)

8. Cheung S., Ammar M., Li X., “On the use of destination set grouping to improve fairness in multicast video distribution,” in *Proc. IEEE INFOCOM*, San Francisco, CA, pp. 553–559 (March 1996)

9. Deering S., “Host extensions for IP multicasting”, Network Working Group, RFC 1112 (August 1989)

10. Fenner B., Handley M., Holbrook H., Kouvelas J., “Protocol independent multicast sparse mode (PIM-SM): Protocol specification (revised)”, Internet draft, draft-ietf-pim-sm-new-v2-01.txt (March 2002)

11. Fenner W., “Internet group management protocol, version 2”, Network Working Group, RFC 2236 (November 1997)

12. Floyd S, Fall K., “Promoting the Use of End-to-End Congestion Control in the Internet”, IEEE / ACM Transactions on Networking, (August 1999), Winner of the Communications Society William R. Bennett Prize Paper Award, 1999.

13. Floyd S., Henderson T., “The NewReno modification to TCP’s fast recovery algorithm”, Network Working Group, RFC 2582 (1999)

14. Golestani S., “Fundamental observations on multicast congestion control in the internet,” in *Proc. IEEE INFOCOM*, New York, pp. 990–1000 (March 1999)

15. Handley M, Byers J, Horn G, Luby M, Vicisano L, “More thoughts on reference simulations for reliable multicast congestion control schemes”, Digital Fountain Inc, Technical report (August 2000)

16. Jacobson V., “Congestion avoidance and control”, in Proc. ACM SIGCOMM, pp. 158–173, (August 1988)

17. Jiang T., Ammar M., Zegura E., “Interreceiver fairness: A novel performance measure for multicast ABR sessions,” in Proc. ACM SIGMETRICS, Madison, WI, pp. 202–211 (June 1998)

18. Jiang T., Ammar M., Zegura E., “On the use of destination set grouping to improve interreceiver fairness for multicast ABR sessions,” in *Proc. IEEE INFOCOM*, Tel Aviv, Israel, pp. 42–51 (March 2000)

19. Kasera S., Bhattacharyya S., Keaton M., Kiwior D., Kurose J., Towsley D., Zabele S., “Scalable fair reliable multicast using active services”, IEEE Network Magazine (Special Issue on Multicast), vol. 14, no. 1, pp.48-57 (January/February 2000)

20. Kumar S. et al., “The MASC/BGMP architecture for interdomain multicast routing,” in Proc. ACM SIGCOMM, Vancouver, BC, Canada, pp. 93–104 (August/September 1998)

21. Legout A., Biersack E., “PLM: Fast convergence for cumulative layered multicast transmission schemes,” in *Proc. ACM SIGMETRICS*, Santa Clara, CA, pp. 13–22 (June 2000)

22. Luby M., Vicisano L., Speakman T., “Heterogeneous multicast congestion control based on router packet filtering,” RMT Working Group (1999)

23. Natu N., Rajagopal P., Kalyanaraman S., "GSC: A Generic Source-based Congestion Control Algorithm for Reliable Multicast," Journal of Computer Communications, Vol 24, No. 5-6, pp. 575-589 (March 2001)

24. Padhye J., "Model-based Approach to TCP-friendly Congestion Control", PhD Thesis, University of Massachusetts Amherst, USA, (2000).

25. Padhye J., *et al.*, “Modeling TCP throughput: A simple model and its empirical validation,” in *Proc. ACM SIGCOMM*, Vancouver, BC, Canada, pp. 303–314 (August/September 1998)

26. Floyd S., “Congestion Control Principles” ,RFC 2914, (September 2000)

27. Floyd S., *et al.*, “TCP Friendly Rate Control (TFRC): Protocol Specification”, RFC 3448, (January 2003)

28. Rhee I., Balaguru N., and Rouskas G., “MTCP: Scalable TCP-like congestion control for reliable multicast,” in Proc. IEEE INFOCOM, New York, pp. 1265–1273 (March 1999)

29. Rizzo L., Vicisano L., Handley M., “PGMCC single rate multicast congestion control: Protocol specification,” draft-ietf-rmt-bb-pgmcc-01.txt (February 2001)

30. Schulzrinne H., Casner S., Frederich R., Jacobson V., “RTP: A transport protocol for real-time applications,” Network Working Group, RFC 1889, (1996)

31. Speakman T., et al., “PGM Reliable Transport Protocol Specification,” Internet draft, draft-speakman-pgm-spec-04.txt (2000)

32. Thapliyal P., Sidhartha, Li J., Kalyanaraman S., “LESBCC: Loss-Event Oriented Source-Based Multicast Congestion Control”, Multimedia Tools and Applications, Vol. 17, No. 2-3, pp. 257-294, (July/August 2002)

33. Waitzman D., Partridge C., Deering S., “Distance vector multicast routing protocol (DVMRP)”, Network Working Group, RFC 1075 (November 1988)

34. Wang H., “Achieving Bounded Fairness for multicast and TCP traffic in the Internet” proceedings of ACM SIGCOMM vol.28 no.4 pp. 81-92 (September 1998)

35. Widmer J., "Equation-based Congestion Control", MSc Thesis, Department of Mathematics and Computer Science, University of Mannheim, Germany, 2000.

36. Widmer J., Handley M., “Extending Equation-based Congestion Control to Multicast Applications” SIGCOMM (August 2001)

37. Yang Y., Lam S., “General AIMD congestion control” Proceedings of ICNP, Osaka, Japan (November 2000)

Appendix A: C++ Code

NS is a discrete event simulator targeted at networking research. NS provides substantial support for simulation of TCP, routing, and multicast protocols over wired and wireless (local and satellite) networks. It is an event driven simulator with an extendible background engine written in C++ and uses Otcl as command and configuration interface.

The NS version used in this study is 2.1b9. Original TFMCC code written for NS is used. The code of LESBCC can not be found, and so, are written for the simulation. An overview but not the exact code is presented below.

The function “start” initializes all the components needed and starts the sender to send data and also starts the rate increase timer.

```
void Sender::start()
{
  srtt=0.1;
  dev=0;

  for(int j=0;j<50;j++)
    timeLastPass[j]=0;

  silenceFlag=0;
  congestionFlag=0;

  for(int j=0;j<50;j++)
    LEcount[j]=0;
```

```
  LEcountAll=0;
  LEcountMax=0;

  statsLI2LE=0;
  statsMaxLPRF=0;
  statsATF=0;
  statsRateReduction=0;

  mss = pktSize_;
  interval = mss/rate_;

  rateIncreaseTimer.resched(srtt+2*dev);
  progressTimer.resched(5);
  sendmsg(pktSize_);
}
```

The function “estimateRTT” calculates a value, which depends on the round trip times of all packets send, in order to be used by filters.

```
void Sender::estimateRTT(double timeSent){
  double crtt = Scheduler::instance().clock() - timeSent;
  if(crtt<srtt/2)
      return;
  double timeDiff = crtt – srtt;
  srtt = srtt + 0.125*timeDiff;
  timeDiff = (timeDiff<0) ? -timeDiff : timeDiff;
  dev += 0.125*(timeDiff-dev);
}
```

The three functions below implements the filters defined in the LESBCC protocol. By these filters, the multicast tree appears to be a unicast path for the sender.

```
int Sender::LI2LEFilter(int receiverID){
  statsLI2LE++;
  double timeNow = Scheduler::instance().clock();
  if((timeNow-timeLastPass[receiverID])<=(srtt + 2*dev)){
    return 0;//filter LI
  }
  //update last pass for this receiver
  timeLastPass[receiverID] = timeNow;
  return 1;//pass LI as LE
}

int Sender::maxLPRFilter(int receiverID){
  statsMaxLPRF++;
  LEcount[receiverID]++;
```

```
  LEcountMax =
(LEcountMax<LEcount[receiverID])?LEcount[receiverID]:LEcountMax;
  LEcountAll++;
  double pAccept = (double)LEcountMax / (double)LEcountAll;
//a random number between 0 and 1
  double randomNumber = Random::uniform();
  if(pAccept>randomNumber){
    return 1;
  }
  return 0;
}

int Sender::ATFilter(){
  statsATF++;
  if(congestionFlag){
    return 0;
  }

  silenceFlag = 1;
  double silencePeriodTime = srtt/2+2*dev;
  silencePeriodTimer.resched(silencePeriodTime);
  odataTimer.resched(silencePeriodTime);

  congestionFlag = 1;
  double congestionEpochTime = silencePeriodTime + srtt + 4*dev;
  congestionEpochTimer.resched(congestionEpochTime);
  rateIncreaseTimer.resched(congestionEpochTime);
  return 1;
}
```

The function “timeout” is the code that is called when a timer expires.

```
void Sender::timeout(int type, void *data)
{
  switch(type) {

  case TIMER_ODATA:
    sendmsg(pktSize_);
    break;

  case TIMER_SILENCE_PERIOD:
    silenceFlag = 0;
    break;

  case TIMER_CONGESTION_EPOCH:
    congestionFlag = 0;
    break;

  case TIMER_RATE_INCREASE:
    noOfRateInc++;
    if(noOfRateInc%100==0){
      LEcountAll*=0.9;
      LEcountMax*=0.9;
      for(int i=0;i<25;i++)
```

```
        LEcount[i]*=0.9;
    }
    if(!congestionFlag){
      //AIMD additive increase part
      //a is 1 for LESBCC and 0.2 for sLESBCC
      rate_ += a*mss/(srtt+2*dev);
      interval = mss/rate_;
    }
    rateIncreaseTimer.resched(srtt+2*dev);
    break;
  }

}
```

The function “sendmsg” build up the data packet to be sent and sends it. It also sets the timer of the next packet sending time.

```
void Sender::sendmsg(int nbytes, const char *flags = 0)
{
  odata_seqno_++;

  // Create a packet
  Packet *pkt = allocpkt();

  hdr_cmn *hc = HDR_CMN(pkt);
  hc->size_ = nbytes;

  hdr_pgm *hp = HDR_PGM(pkt);
  hp->seqno_ = odata_seqno_;

  timeSent[odata_seqno_] = Scheduler::instance().clock();

  // Send out the packet.
  send(pkt, 0);
  odataTimer.resched(interval);
}
```

When an LI is received by the sender from any receivers, the function “handle_nak” is called. It updates the statistics used by the filters but the main task is to decide if the rate will be halved or not.

```
void Sender::handle_nak(Packet *pkt){

  hdr_pgm *hp = HDR_PGM(pkt);
  hdr_ip *hip = HDR_IP(pkt);
```

```
  stats_.num_naks_received_++;

  estimateRTT(timeSent[hp->seqno_]);
  int receiverID = (int)hip->saddr();
  if(LI2LEFilter(receiverID))
     if(maxLPRFilter(receiverID))
        if(ATFilter()) {
          statsRateReduction++;
          //AIMD multiplicative decrease part
          //b is 0.5 for LESBCC and 0.875 for sLESBCC
          rate_ /= b;
          interval *= b;
        }
}
```

Appendix B: OTcl Script

In Otcl side of NS, mainly, the network topologies are designed. Otcl is a script so it does not need any compilation. The main idea is: implement your protocol in C++ then run your simulations with different topologies without changing the C++ part. The code below is a simple topology design to compare the performances of TCP and LESBCC.

```
set ns [new Simulator -multicast on]

#
# Create multicast group
#
set group [Node allocaddr]
puts "Group addr: $group"

for {set k 0} {$k < 8} {incr k} {
  set n($k) [$ns node]
}

proc makelinks { bw delay queueSize pairs } {
        global ns n
        foreach p $pairs {
                set src $n([lindex $p 0])
                set dst $n([lindex $p 1])
                $ns duplex-link $src $dst $bw $delay DropTail
                $ns queue-limit $src $dst $queueSize
        }
}

makelinks 0.5Mb 50ms 30 {{ 2 3 }}
makelinks 10Mb 1ms 30 {
{ 0 2 }
{ 1 2 }
{ 3 4 }
{ 3 5 }
```

```
{ 3 6 }
{ 3 7 }
}
#
#Set routing protocol
#
set mproto DM
set mrthandle [$ns mrtproto $mproto {}]

#
#Setup a TCP connection
#
set tcpSrc [new Agent/TCP/Reno]
$tcpSrc set packetSize_ 1400
$ns attach-agent $n(0) $tcpSrc


set sink [new Agent/TCPSink]
$ns attach-agent $n(4) $sink
$ns connect $tcpSrc $sink

#Setup a FTP over TCP connection
set ftp [new Application/FTP]
$ftp attach-agent $tcpSrc

#
# Create Sender
#
set src [new Agent/PGM/Sender]
$ns attach-agent $n(1) $src
$src set dst_addr_ $group
$src set dst_port_ 0
$src set packetSize_ 1400
$src set rate_ 1440

#
# Create receivers
#
for {set k 5} {$k < 8} {incr k} {
  set rcv($k) [new Agent/PGM/Receiver]
  $ns attach-agent $n($k) $rcv($k)
  $ns at 0.01 "$n($k) join-group $rcv($k) $group"
}

set timeEnd 3000
$ns at 0.1 "$tcpSrc openOutputFile case1.tcp.txt"
$ns at 0.1 "$src start case1.lesbcc.txt"
$ns at 0.1 "$ftp start"
$ns at $timeEnd "$ftp stop"
$ns at $timeEnd "$src stop"
$ns at $timeEnd "$tcpSrc closeOutputFile"
$ns at $timeEnd "exit 0"

$ns run
```