MULTIPATH TRANSPORT AND FLOW DIVISION FOR MULTIHOMED HOSTS

A

Thesis Submitted

in Partial Fulfilment of the Requirements

for the Degree of

DOCTOR OF PHILOSOPHY

by

Samar Shailendra

Department of Electronics and Electrical Engineering
Indian Institute of Technology Guwahati
Guwahati - 781 039, INDIA.

January, 2013

Certificate

This is to certify that the thesis entitled "**MULTIPATH TRANSPORT AND FLOW DIVISION FOR MULTIHOMED HOSTS**", submitted by **Samar Shailendra** (09610204), a research scholar in the *Department of Electronics & Electrical Engineering, Indian Institute of Technology Guwahati*, for the award of the degree of **Doctor of Philosophy**, has been carried out by him under our joint supervision and guidance. The thesis has fulfilled all requirements as per the regulations of the institute and in our opinion has reached the standard needed for submission. The results embodied in this thesis have not been submitted to any other University or Institute for the award of any degree or diploma.

Prof. Sanjay K. Bose
Deptt. of Electronics & Electrical Engg.
Indian Institute of Technology Guwahati
Guwahati - 781039, Assam, India.

Prof. Ratnajit Bhattacharjee
Deptt. of Electronics & Electrical Engg.
Indian Institute of Technology Guwahati
Guwahati - 781039, Assam, India.

Acknowledgements

I am greatful to *Prof. Sanjay K. Bose* & *Prof. Ratnajit Bhattacharjee* for providing me an opportunity to carry out my PhD work under their supervision. Their ability to ask the right questions and attention to smallest detail of the research has never ceased to surprise me. Their constant emphasis on clear communication helped me a lot. They have enriched my life in many significant ways. I thank them from the bottom of my heart for always being there with me during all kinds of distress.

I would like to thank my doctoral committee members: Dr. Purandar Bhaduri, Dr. P. R. Sahu, and Dr. A. Rajesh for sparing time out of their busy schedule to evaluate my progress and enrich this work with their invaluable suggestions and feedbacks. I thank Dr. Kannan Karthik for greatly helping me with my academic activities. I would like to thank other faculty members for helping me with my research work in innumerable ways.

My special thanks to Mr. Sanjib Das and Mrs. Jharna to extend their untiring help and support for various experimental activities. I thank Mr. Utpal for always being available related to any lab problem. I would also like to thank everyone in High Frequency Lab and Image & Signal Processing Lab who made these years really enjoyable.

I am really fortunate to find Samdarshi and Shashank and accepting my friendship. I enjoyed all my discussions, technical as well as non-technical with them. The wittiness of Samdarshi has always brought a new aspect to even the most obvious things. Shashank, a man of few words, has always been their to listen to me in my low days. I cherish our continuing friendship. I am also thankful to all my colleagues for providing me help and support during the course of my stay at IITG.

One person stands out during my tenure at IITG for whom I will always be thankful. Sneha, a wonderful person, a great researcher has always been there to discuss my problems at length. I have enjoyed various discussions with her on multiple subjects not only just technical. We always have different perspective towards things. Mostly our discussion ends up with “Let’s agree to disagree”. I have enjoyed and remember every moment of her company.

I express my deep appreciation to my wife Jolly for her unconditional support and understanding for all these moments. This PhD would not have been possible without her help and support. I

extend my love to my son, Suhas who always kept me motivated by asking “when will you come not to return?”. I extend deepest gratitude to my Grandpa for always being their with me with his thoughts, suggestions, and guidance not only during this PhD rather throughout my life. I am thankful to my Parents, my brother and in-laws whose love and encouragement made this research possible.

I also extend my thanks to the staff of EEE office and Academic Office for helping me out with all sorts of academic activities during my stay at IITG. Last but not the least, I thank everyone who have made my stay here wonderful and memorable.

(Samar Shailendra)

Abstract

This research primarily deals with the problem of flow division in the context of emerging multihomed protocols like the Stream Control Transmission Protocol (SCTP). In this work we augment SCTP to increase its throughput and overall reliability by incorporating flow division strategies which divide a flow on multiple paths. Optimizing this flow division will enhance the system throughput while maintaining the QoS required by the flow.

We propose a new MultiPath SCTP (MPSCTP) protocol as an extension of SCTP to support concurrent usage of multiple paths. We have demonstrated that our modifications provide better performance in comparison to CMT-SCTP, a current multipath variant of SCTP. We have proposed Bandwidth Estimation based Resource Pooling (BERP) congestion control, a novel congestion control for MPSCTP. This algorithm is based on a linked congestion control approach which treats all available resources as a single pool of resources and in the event of packet loss decreases the congestion window in the ratio of bandwidth estimates of the paths.

Existing multipath protocols do not optimize the flow division on the available multiple paths. There have been various efforts in the past to divide data flow on multiple paths. Flow division at source depends upon the nature of application as well. We have considered two classes of application namely delay sensitive applications and delay insensitive applications. For these two classes of applications, we have proposed Min-Max optimization and Delay Insensitive Optimization respectively. A true implementation of these optimizations would require complete network statistics to be available at the source. Since, this is difficult in an Internet-like scenario; suitable heuristics are required which can closely approximate these optimizations. In this work, we have proposed suitable heuristics for the Min-Max Optimization and Delay Insensitive Optimization.

We have also implemented the MPSCTP protocol in Linux kernel. We have incorporated our changes in Linux kernel. We have also proposed *Tx-CWND*, a new strategy for flow division.

According to this strategy the packets are transmitted on the path with largest *cwnd*. This is much simpler to implement in the Linux kernel and we have also demonstrated that this provides results similar to the *RTX-CWND* strategy proposed in literature.

Contents

List of Figures

List of Tables

Glossary

TCP	Transmission Control Protocol
UDP	User Datagram Protocol
SCTP	Stream Control Transfer Protocol
SIGTRAN	Signaling Transport
IETF	Internet Engineering Task Force
HOL	Head-of-Line
DOS	Denial of Service
SACK	Selective Acknowledgement
LS-SACK	Load Sharing-Selective Acknowledgement
LS-SCTP	Load Sharing-Stream Control Transmission Protocol
TSN	Transmission Sequence Number
PSN	Path Sequence Number
CMT	Concurrent Multipath Transfer
cmpSCTP	Concurrent Multipath SCTP
ASN	Association Sequence Number
SFR	Split Fast Retransmission
WM^2-SCTP	Wireless Multi-path multi-flow SCTP
QoS	Quality of Service
FID	Flow Identifier
FSN	Flow Sequence Number
MPTCP	Multi-Path Transmission Control Protocol
ATM	Asychronous Transfer Mode
RTT	Round Trip Time
cwnd	Congestion Window

FTP	File Transfer Protocol
HTTP	Hyper Text Transmission Protocol
AMPL	A Modeling Language for Mathematical Programming
MPSCTP	Multipath Stream Control Transmission Protocol
ACK	Acknowledgement
cumPSNAck	Cumulative PSN Acknowledgement
cumTSNAck	Cumulative TSN Acknowledgement
cumAck	Cumulative Acknowledgement
PSNR	Peak Signal to Noise Ratio
BERP	Bandwidth Estimation Based Resource Pooling
MTU	Maximum Transmission Unit
MSS	Maximum Segment Size
ssthresh	Slow Start Threshold
CBR	Constant Bit Rate
BDP	Bandwidth Delay Product

List of Publications

Journal Publications

1. Samar Shailendra, Ratnajit Bhattacharjee, Sanjay K. Bose, "MPSCTP: A Simple and Efficient Multipath Algorithm for SCTP ", *IEEE Communication Letters*, vol. 15, issue: 10, pp. 1139-1141, October 2011, doi: 10.1109/LCOMM.2011.080811.110866.

2. Samar Shailendra, Ratnajit Bhattacharjee, Sanjay K. Bose, "An Implementation of Min-Max Optimization for Multipath SCTP through Bandwidth Estimation based Resource Pooling Technique", *International Journal of Electronics and Communications (AEU), http://dx.doi.org/10.1016/j.aeue.2012.08.008.*

International Conferences

1. Samar Shailendra, Ratnajit Bhattacharjee, Sanjay K. Bose,"Optimized flow based modeling of multi-path multi-homed transmission", *Communication Systems (ICCS), 2010 IEEE International Conference on* , pp. 411-415, 17-19 November 2010, doi: 10.1109/ICCS.2010.5686517.

2. Samar Shailendra, Ratnajit Bhattacharjee, Sanjay K. Bose,"Optimized Flow Division Modeling for Multi-Path Transport ", *IEEE INDICON-2010*, pp: 1-4, December 2010, doi: 10.1109/INDCON.2010.5712713.

3. Samar Shailendra, Ratnajit Bhattacharjee, Sanjay K. Bose,"Improving Congestion Control for Concurrent Multipath Transfer through Bandwidth Estimation based Resource Pooling", *8th international Conference on Information, Communications and Signal Processing (ICICS) 2011*, pp:1-5, 13-16 December 2011, doi: 10.1109/ICICS.2011.6174285.

4. Samar Shailendra, Ratnajit Bhattacharjee, Sanjay K. Bose,"MPSCTP: A Multipath Variant of SCTP and its performance comparison with other Multipath Protocols", *Communication Systems (ICCS), 2012 IEEE International Conference on*, pp:1-4, November 2012, doi: 10.1109/ICCS.2012.6406154.

1

Introduction

Contents

In computer networks, the Transport Layer provides communications between two end hosts. This is the fourth layer in the layered networking architecture as shown in Figure 1.1. It transports data between two end applications using the Network Layer services. The current network layer uses Internet Protocol (IP) to carry out its functionality. However, IP is connectionless and unreliable in nature. The Transport Layer provides connection oriented, flow controlled and reliable communications over the otherwise connectionless and unreliable IP Protocol. Unlike other lower layers, the Transport Layer provides end-to-end control as shown in Figure 1.1 and is therefore better aware of the end-to-end path characteristics than the lower layers in the networking stack. This feature makes transport layer a suitable candidate to implement flow and congestion control.

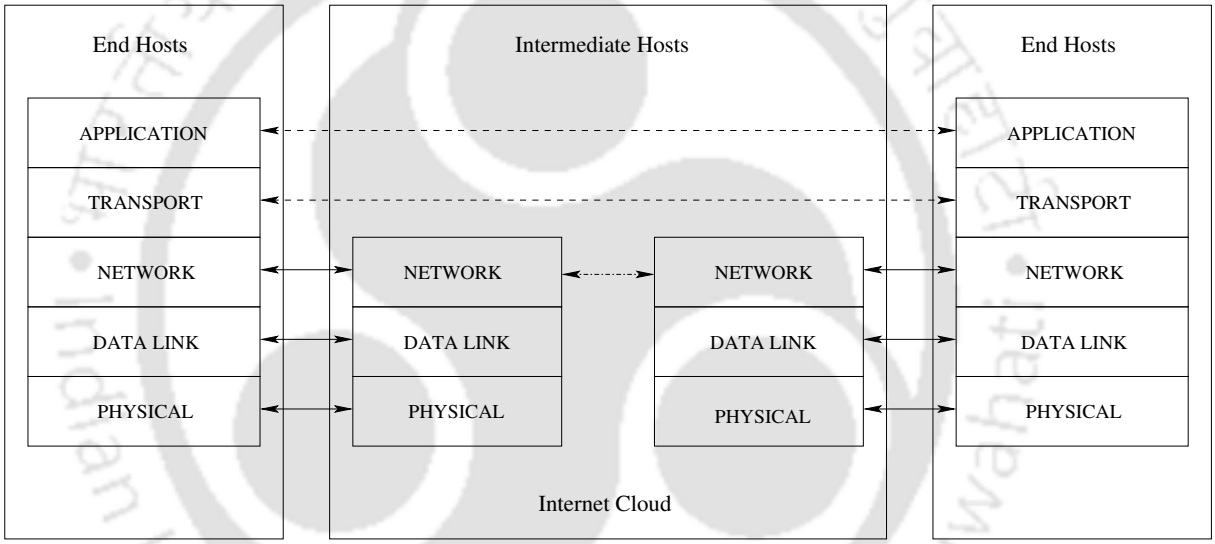


Figure 1.1: Internet Network Architecture and Layer Interaction.

Current Transport Layer protocols like TCP and UDP use only one single path for data transmission between end hosts. There have been several efforts in the past to enable multipath transmission for better resource utilization and error resilience. Several approaches have been presented in the literature to enable multipath transmission at different layers. Applications like download accelerators use multiple connections to improve the system throughput and load balancing [1, 2]. These applications establish several connections using multiple sockets. The basic disadvantage of this approach is that the application layer does not have direct knowledge about the path conditions. Moreover, load balancing at the application layer increases the design complexity of the application itself. Several proposals have been made for multipath routing at the network layer [3–5]

where authors have suggested routing packets to the destination using different paths. However, the network layer is not aware of the complete path characteristics of the network. Hence, any decision at the network layer would require that the complete path information is made available at the router – this is not what is typically done in router implementations. Multipath transmissions at the transport layer are implemented using multi-homed hosts which can support multiple network interfaces simultaneously for data transmission.

1.1 SCTP - a multihomed transport protocol

In 2001, IETF proposed the Stream Control Transfer Protocol (SCTP) [6] as a new transport protocol with multihoming support. The underlying motivation behind SCTP was more than just to provide a multihoming protocol. SCTP was originally designed as a reliable protocol for Signaling Transport (SIGTRAN) [7–11]. However, its versatility makes SCTP a potential candidate as an alternate transport protocol for the Internet. SCTP was proposed with several salient features which are listed below.

(i) Message Oriented

SCTP is a message oriented protocol. Unlike TCP which is byte oriented and does not preserve the message boundaries, SCTP transports the entire message and passes it on to the upper layer at the receiver side.

(ii) 4-way Handshake

Unlike TCP which uses 3-way handshake, SCTP uses 4-way handshaking before establishing a connection. This is done using cookie mechanism as shown in Figure 1.2.

In TCP, a client initiates a connection request with SYN message. In TCP [12], the server reserves resources for this connection as soon as it receives the SYN request and replies with the SYN-ACK message. Hence, an attacker can launch multiple SYN requests to tie up all the server resources while legitimate requests may be denied the service. In contrast, a SCTP client initiates a connection request with an INIT message and the server responds with an INIT-ACK to this INIT request. Along with the INIT-ACK, the server sends a state cookie as well. However, unlike the SYN-ACK message in TCP, the resources are not reserved at this point. The client now replies with COOKIE-ECHO with the state cookie received.

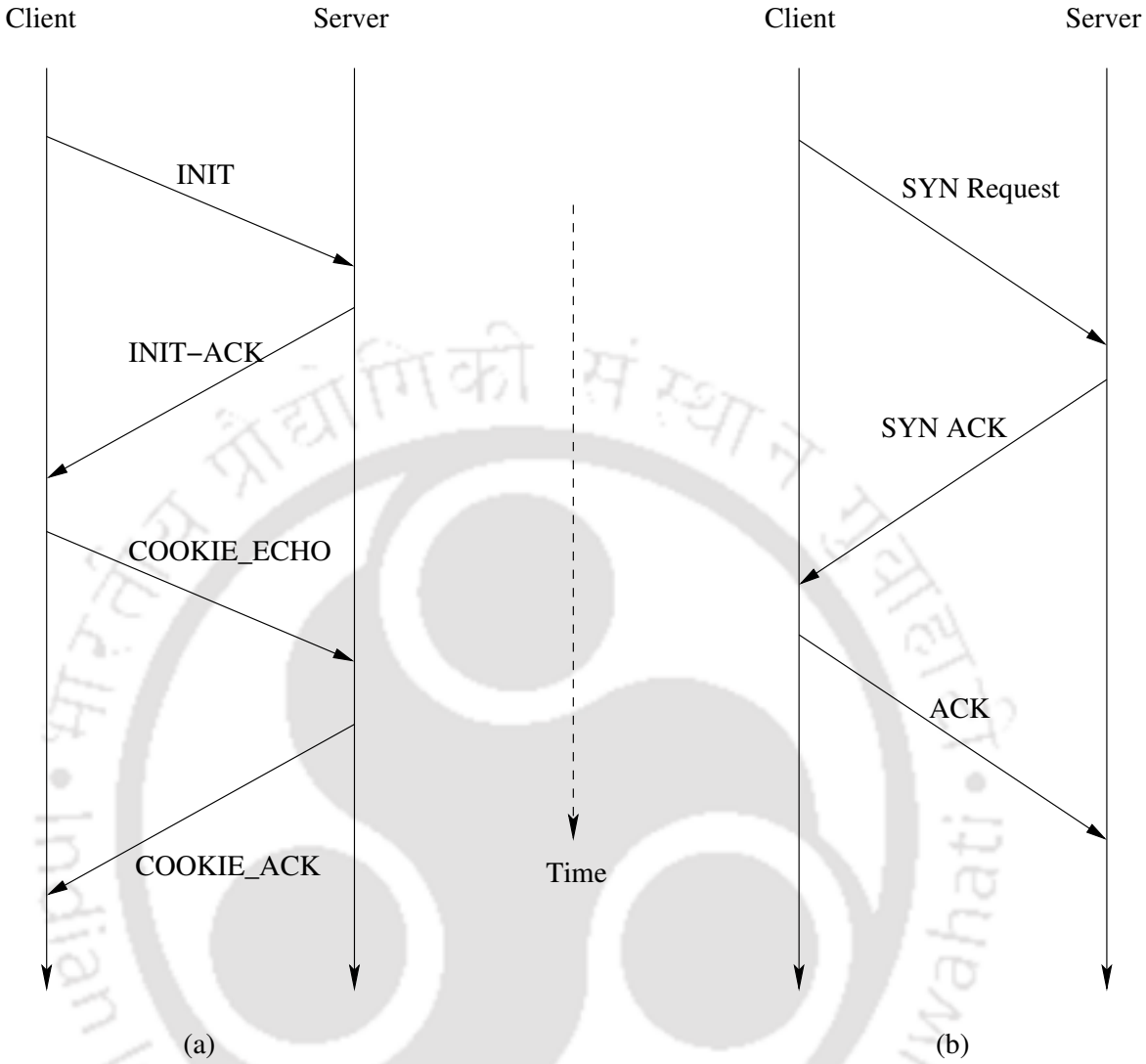


Figure 1.2: (a) SCTP Handshake Mechanism (b) TCP Handshake Mechanism.

On receiving a valid COOKIE-ECHO message, the server allocates the resources and replies back with COOKIE-ACK. Until the server receives the COOKIE-ECHO message, it remains unaware about any association with the corresponding client and no resources are reserved at the server side. At this point server validates the state cookie. The resources are allocated only if the cookie is validated and it has not already expired. This makes SCTP robust to various Denial of Service (DoS) attacks [13]. Though the 4-way handshake may look like an extra overhead in comparison to the simpler TCP handshaking, in SCTP data can be bundled with last two handshake messages to reduce the associated overhead. However, care needs to be taken to ensure that handshake messages always precede user data in an SCTP packet.

(iii) Multistreaming

Multistreaming is a feature where data can be divided into multiple streams such that the delivery of data in one stream does not affect the data delivery from another stream. This means the contents of one stream can be delivered to the application at the receiver side even though another stream has encountered some packet loss, e.g. images and text of a webpage can be transmitted using different streams to the destination. Even if the data belonging to the image is lost, the text contents which belong to another stream can still be delivered to the application. This helps in decreasing the Head-of-Line blocking that one can encounter in the TCP protocol [14, 15].

(iv) Unordered data delivery

SCTP supports unordered data delivery as well. SCTP has options flag as shown in Figure 1.3. If U-flag is set, SCTP delivers the received data to the upper layer even though it is not in sequence. This feature is very useful while delivering any "out-of-band" data. If the U-flag is set for a chunk, SCTP bypasses the entire ordering mechanism and immediately delivers the data to the upper layer.

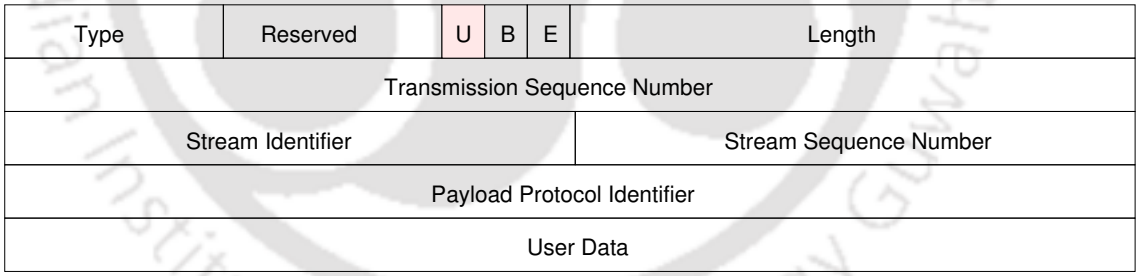


Figure 1.3: SCTP Chunk Header.

(v) Multihoming

Multihoming is an interesting feature of SCTP. Multihoming is possible in a network scenario where end nodes support multiple interfaces as shown in Figure 1.4. SCTP connects through all the available interfaces. The connections between two end points are collectively referred to as an association. In SCTP, multiple IP addresses connect through one SCTP port. SCTP uses one of the available paths as the primary path for data transmission. In the event of primary path failure or degradation, SCTP can switch from the primary path to any other

alternatively available path for the data transfer. This feature allows SCTP to provide better error resilience and fault tolerance as compared to other protocols. In SCTP, retransmissions may be sent on alternate paths as well. SCTP uses heartbeat messages to know the health of candidate paths and provides a seamless changeover to alternate paths which is transparent to the end users.

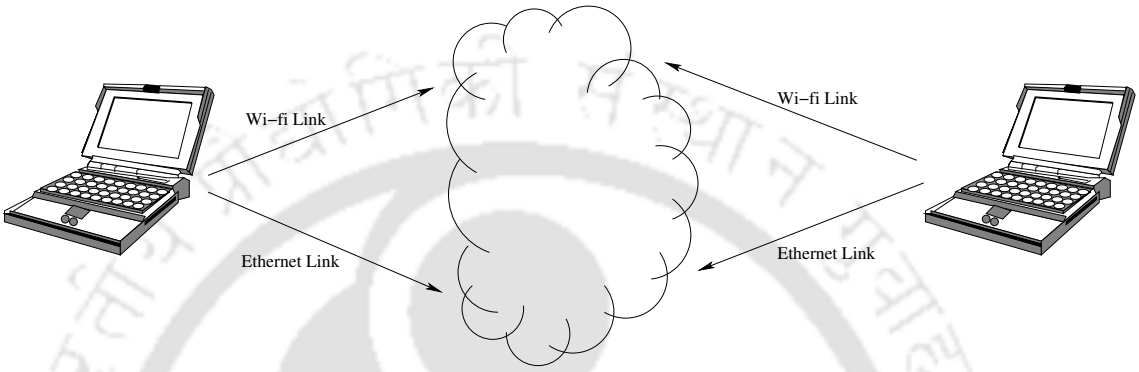


Figure 1.4: Multihoming in SCTP.

A comparison of SCTP with TCP, UDP and Multipath TCP (MPTCP[†]) is provided in Table 1.1.

1.2 Motivation of the Present Work

Concurrent usage of multiple paths is a relatively new concept and technologies driving multipath usage are not mature enough. In earlier days multiple interfaces were rare. Today, it is not unusual to find computers with more than one network interface. For example, almost every computer sold today usually has two interfaces, one for a wired connection along with a wireless interface. Handheld devices like mobile phones and tablets also have multiple interfaces like 3G, GPRS and Wi-Fi. Hence, these devices have the potential to connect to multiple interfaces simultaneously. (Wi-Fi interface can be configured to interface with more than one access point which would also effectively provide multiple network interfaces [16].) Moreover, connecting through multiple interfaces not only improves the performance in terms of better error and network failure resilience but can also help in improving performance during call handover, i.e. a mobile device connected to the base stations of different cells simultaneously is less likely to experience outage

[†]MPTCP is a recent proposal from IETF. Its standards are still in an experimental state.

Table 1.1: Features of various Transport Protocols

Services/Features	SCTP	TCP	UDP	MPTCP
Connection-oriented	Yes	Yes	No	Yes
Full duplex	Yes	Yes	Yes	Yes
Reliable data transfer	Yes	Yes	No	Yes
Partial-reliable data transfer	Optional	No	No	No
Ordered data delivery	Yes	Yes	No	Yes
Unordered data delivery	Yes	No	Yes	No
Flow control	Yes	Yes	No	Yes
Congestion control	Yes (Based upon TCP Congestion Control)	Yes	No	Yes (Based upon resource pooling principle)
ECN capable	Yes	Yes	No	Yes
Selective ACKs	Yes	Optional	No	Optional
Sequence Numbering Levels	Single	Single	n/a	Two
Preservation of message boundaries	Yes	No	Yes	No
Path MTU discovery	Yes	Yes	No	Yes
Application PDU fragmentation	Yes	Yes	No	Yes
Application PDU bundling	Yes	Yes	No	Yes
Multistreaming	Yes	No	No	No
Multihoming	Yes	No	No	Yes

and call dropping during mobility from one cell region to another.

Simultaneous usage of multiple paths requires the source flow to be split between the different candidate paths. Improper division can lead to inefficient usage of network resources where the throughput of the multipath protocol becomes worse than the throughput where only one path is used. It therefore becomes essential that the flow is split on the candidate paths such that the overall network resources are properly utilized and the system performance is at least as good as that using only one path. Along with flow division, the congestion control mechanism for such protocols also needs to be revisited. TCP flow and congestion control has proven its stability over time. Current congestion control mechanisms such as the TCP congestion control is designed keeping only one single path usage at a time. It may be noted that though SCTP is a multi-homed

protocol, it still uses only one path at a time, i.e. it cannot do multipath transmission using more than one path concurrently. SCTP designates one of the paths as the primary path and prefers using it for data transfer. In the event of primary path degradation, it uses one of the secondary paths and switches back to the primary one once it is restored. (Hence, TCP like congestion control works all right with SCTP.) However, such an algorithm is not suitable for true multipath protocols where more than one path is used concurrently. Congestion control mechanisms need to be suitably modified if multipath transport is to be used effectively.

1.3 Thesis Contribution

This thesis aims at improving the current state of the art of network usage by implementing multipath transport over an SCTP protocol, utilizing network resources efficiently and providing end users a better network usage experience. Several optimization approaches considering the nature of application and their respective QoS have been proposed. In particular, the following are the main contributions of this thesis.

(a) Flow Division at source:

This thesis introduces the concept of flow division at source. To implement multipath transport, it is essential to split the flow on multiple paths such that a pre-defined end-to-end objective is achieved. As discussed in subsequent chapters, there might even be scenarios where not actually using a particular path may improve the performance of the system or where the throughput may actually decrease when the number of paths is increased. Optimum division of flow on multiple paths would therefore be important for effective use of multipath transport protocols. We suggest some of the different ways in which flow division can be done and indicate that the actual technique used would depend on the nature of the application for which transport is required.

(b) Multipath Protocol:

Multipath SCTP (MPSCTP), a new multipath protocol based on SCTP has been proposed. MPSCTP uses two levels of sequence numbering to isolate the sequencing at the path from the application data sequencing. This requires changes in the chunk header as well as in the acknowledgement structure of the original SCTP protocol. Several SCTP algorithms also need

to be modified for faithful transmission of data from the source to the destination. Our proposed MPSCTP approach demonstrates significant performance improvements over an earlier attempt, Concurrent Multipath Transfer (CMT)-SCTP [17], to augment SCTP for multipath transport.

(c) Congestion Control Algorithm:

Both MPSCTP and CMT assume disjoint paths between the source and the destination. However, this is a very strong assumption in an Internet like scenario. SCTP uses a TCP like congestion control algorithm. In multipath scenario, this algorithm captures unduly more bandwidth on shared links. This thesis proposes a new congestion control algorithm which assumes all the resources as a single pool of resources and does not harm fellow flows on the shared link. This congestion control uses bandwidth estimation based resource pooling for adjusting the congestion window for the source node and has been incorporated in our proposed scheme.

(d) Heuristic for Flow Division:

This thesis provides suitable heuristics for the proposed flow division optimizations. In a typical scenario the optimization problem needs to be solved every time a decision is to be made. However, this is highly computationally intensive because it requires the complete network statistics to be known at the source. This is relatively difficult in an Internet like scenario where the path characteristics and link traffic change dynamically. We give heuristic methods which perform well based on much simpler end-to-end parameter measurements.

(e) MPSCTP Implementation in Linux kernel:

This thesis also provides a sample implementation of MPSCTP in Linux kernel. This implementation is based upon the SCTP implementation [18] in the Linux kernel. Our implementation modifies the SCTP implementation such that it is able to provide simultaneous data transfer on multiple paths. These changes are transparent to the end users. The users and the applications can continue using the same APIs without any changes. Our current implementation supports only two concurrent paths though extending it for more paths should be feasible.

1.4 Thesis Organization

This thesis is organized as follows. A structural organization of this thesis and the corresponding publications are also shown in Figure 1.5.

Chapter 1, the current chapter, discusses the problem statement and the motivation behind these problems. This chapter highlights the salient features of SCTP and its comparison with other transport protocols. This chapter also summarizes the thesis contribution and provides a brief outline of the thesis organization.

Chapter 2 presents a literature survey of related work in the area of SCTP, multipath protocols, flow division and congestion control.

Chapter 3 explores various methods for optimization of flow division at the source. Various optimization models are suggested for this and their relative performances are compared and discussed. We consider here the problem of optimally dividing the flow between multiple disjoint paths at the source. Since different applications may have different QoS requirements [19], we also consider the nature of the application for the optimal flow division.

Chapter 4 introduces our proposed Multipath SCTP (MPSCTP), a multipath variant of SCTP, which can use available candidate paths in a concurrent fashion. This chapter introduces Bandwidth Estimation based Resource Pooling (BERP) congestion control and its performance results. Along with these, this chapter also provides a comparative study of some popular multipath protocols.

Chapter 5 gives heuristic approaches for the flow division techniques proposed in Chapter 2 for two classes of applications, i.e. delay sensitive and delay insensitive applications. In this chapter, we propose a heuristic for the Min-Max Optimization and demonstrate that a practical implementation of this heuristic can be closely approximated by the BERP algorithm. We have also proposed a suitable heuristic to implement Delay Insensitive Optimization in this chapter.

Chapter 6 presents an implementation of MPSCTP protocol in the Linux kernel. In this chapter, we discuss the changes required to implement a MPSCTP module in the Linux kernel. The performance of this implementation is studied both in an emulated network as well as in the actual campus LAN environment of IIT Guwahati.

Lastly **Chapter 7** concludes this thesis with a summary of the work done and includes some suggestions that may be investigated in future research.

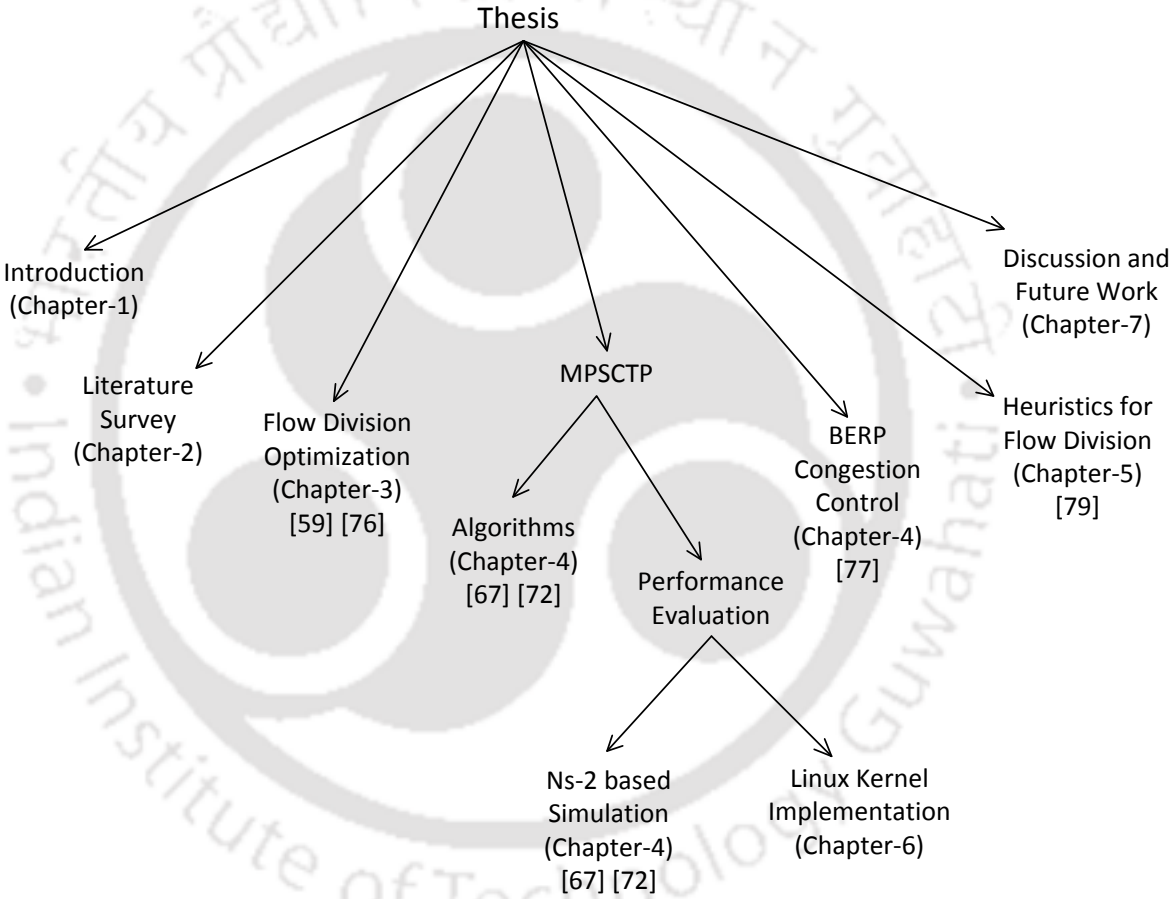


Figure 1.5: Thesis Organization.

2

Literature Survey: Review of Related Work

Contents

The Internet has evolved as a global network that connects different computer networks across the world. In the Internet, TCP [12] and UDP [20] are the most widely used transport protocols for data communications. These protocols do not provide native support to multihoming, a scenario where the end node supports multiple interfaces, as UDP inherently does not bind to any address while TCP binds to only one address and uses only one path even when several paths are available between the source and the destination. While efforts have been made in the past to incorporate multiple path usage in TCP, these have not been popular or very successful. Parallel TCP [21] is one such scheme which was designed to use multiple paths between a source and its destination; however, it performs poorly when there are congestion losses and is also unfair to fellow TCP flows.

In 2001, SCTP was proposed to carry signaling data for SIGTRAN. IETF had proposed Signaling Transport (SIGTRAN) [7, 8] as an extension of the Signaling System (SS7) [9–11] protocol family. SIGTRAN supports the same functionality as SS7 but provides mechanisms to carry PSTN signals over IP Network. However, these signaling messages cannot use UDP as their transport protocol because UDP does not have any reliability mechanism. This leaves TCP as an option but TCP has several problems. In TCP if a packet is lost all the subsequent packets are blocked unless the lost one is not retransmitted and reliably delivered. This is called Head-of-line (HOL) [14] blocking and severely affects the protocol throughput. TCP is byte oriented and does not preserve the message boundaries. TCP timers might take even minutes before the upper layers are being notified for lost connection. Moreover, TCP has several Denial of Service (DoS) attacks [22]. Hence, IETF proposed Stream Control Transmission Protocol (SCTP) based upon another proposal called Multi-network Datagram Transmission Protocol (MDTP) [23]. Later, SCTP was realised to be a potential candidate as an alternate transport protocol for Internet. SCTP [6, 24–27] was proposed with salient features like four way handshaking, multistreaming and multihoming. Multistreaming is a feature where data can be divided in multiple streams such that the delivery of data in one stream does not affect the data delivery over another stream. This helps in decreasing the Head-of-Line blocking that one can encounter in the TCP protocol [14, 15]. Unlike TCP which uses three way handshaking, SCTP uses four way handshaking. This makes SCTP robust to various Denial of Service (DoS) attacks [13]. Multihoming is possible in a network scenario where a single SCTP endpoint can support multiple IP addresses. SCTP then uses one of the available paths as the primary path for data transmission. In the event of primary path failure or degradation of the

primary path, SCTP can switch from the primary path to any other alternatively available path for the data transfer. This feature allows SCTP to provide better error resilience. To augment this multihoming feature of SCTP, efforts have also been made to use the available multiple paths in a concurrent fashion to provide true multihoming. Using available paths in a concurrent fashion has been demonstrated to improve the performance and error resilience of the network.

2.1 Multipath Protocols

Several variants of SCTP and TCP have been proposed in the literature to support multipath transfer and load sharing. The major contributions in this are summarized next.

2.1.1 Load Sharing SCTP

This protocol [28] is focused on using multiple available paths simultaneously for data transfer. The authors have included a path monitoring module to monitor the active paths in the network as shown in Figure 2.1.

The path monitoring module maintains a list of active paths to be used for transmission. A path is marked as inactive if there are more timeouts than the $Path.Max.Retrans$ which is typically assumed to be "5" in [28]. The sender keeps monitoring the inactive paths using the heartbeat chunks. The protocol adds another field which is combination of a 4 bit path identifier and 12 bit path sequence number. The path identifier identifies the path in the entire association while path sequence number is used to identify the sequence of the packet on every path.

The minimum receiver buffer requirement for the receiver (B_{min}) is given by (2.1) where B_i is the bandwidth for i^{th} path and there are N active paths at any given time and RTT_{max} is the maximum *Round Trip Time*.

$$B_{min} = (\sum_{i=1}^{N} B_i) \times RTT_{max} \tag{2.1}$$

The protocol introduces a new SACK structure Load Sharing SACK (LS-SACK) chunk. This chunk also includes a time stamp and a per path cumulative acknowledgement.

2.1.2 Independent per Path Congestion Control SCTP

SCTP in its traditional form uses a single congestion window. This is intuitively correct as well because native SCTP implementation uses only the primary path for the data transmission

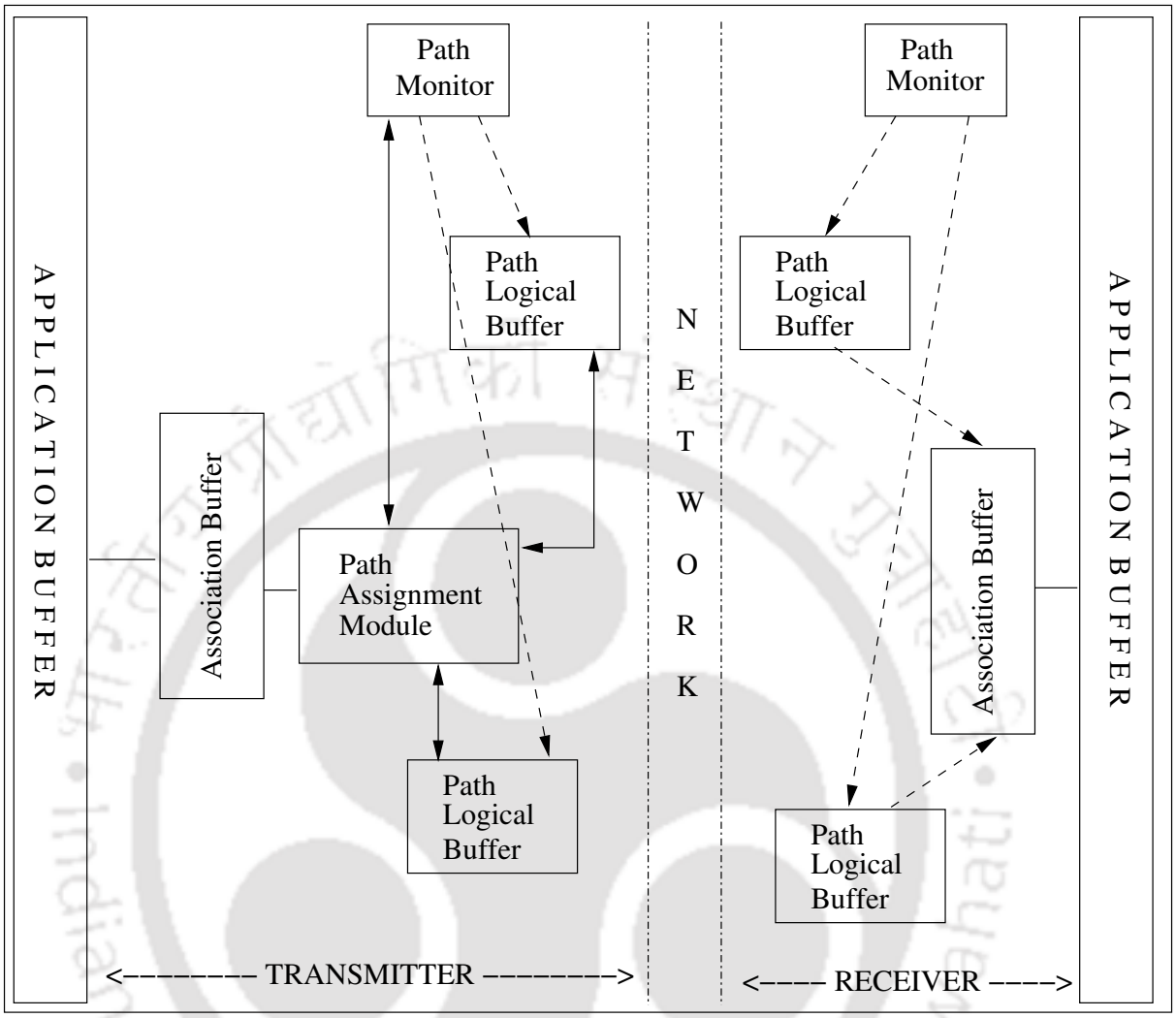


Figure 2.1: LS-SCTP Architecture Diagram

and a secondary path is used only if the primary path degrades or becomes unusable. When the original primary path comes up again, SCTP resumes using that for data transmission. In this work, Ye et al. [29] propose to use multiple available paths in tandem. This necessitates the use of multiple congestion windows for each candidate path. In this work, the authors have also suggested that retransmissions on the alternate path lead to the better throughput and the delay performance in the network. The authors have introduced the concept of an implicit path sequence number to support multipath transmission. This number is used to identify the packets being sent on a particular path. The advantage of this approach is that unlike *Load Sharing SCTP*, this does not require any modification in the chunk header of the SCTP. However, the source and

destination side algorithms need to be changed for the proper updates of the congestion window. Moreover, the interpretation of the various SACK fields is also required to be redefined for the proper acknowledgement of the data. Here, the authors have not described the details of how reassembly would be done at the destination. Moreover, this scheme requires that a Transmission Sequence Number (TSN) to path sequence number mapping be maintained at the transmitter which would detrimentally impact the scalability of the protocol.

The authors in this work have studied the load sharing on two paths and compared that with the situation where the two paths are used independently. Performance results using load sharing have also been reported.

2.1.3 Concurrent Multipath Transfer SCTP

Concurrent Multipath Transfer (CMT) [17, 30] is another multipath protocol that has been reported earlier. When multiple disjoint end-to-end paths are available, CMT uses them simultaneously to increase the application throughput. It may be noted that CMT does not add any additional packet identifier, either explicitly or implicitly. However, to identify the packet reordering (a natural consequence of multipath transmission), to isolate packet loss from packet reordering, to handle fast retransmission and to decrease the increased ACKs due to holes in the packet sequence numbers received at the destination, CMT proposes three algorithms which require modifications both at the sender side and the receiver side. CMT suggests acknowledging the data received on two different paths in a single delayed acknowledgement. This simultaneously increases the congestion window (*cwnd*) of both the paths; this phenomenon is referred to as *aggregated cwnd overgrowth*. The authors argue that this phenomenon happens only during the slow start phase and is not TCP-unfriendly and that it actually improves the sequence space sharing on the candidate paths. This work has also been supplemented with five retransmission policies: RTX-SAME, RTX-ASAP, RTX-SSTHRESH, RTX-CWND and RTX-LOSSRATE [31, 32]. This protocol shares the TSN sequence space among available candidate paths and ensures that the packets and corresponding acknowledgements can be transmitted on different paths. However, this does create a problem that RTT calculations may no longer be proper which would lead to poor performance if the RTTs of the candidate paths differ significantly.

2.1.4 Concurrent Multipath SCTP

Concurrent Multipath SCTP (cmpSCTP) [33] has been proposed based upon the design of LS-SCTP. However, unlike LS-SCTP, cmpSCTP does not have any path monitoring module because this protocol uses all the available paths all the time. This protocol splits the 32 bit Transmission Sequence Number (TSN) into two parts consisting of 4 bit Path identifier and 28 bit TSN. However, this splitting of sequence number reduces the sequence space available for packet numbering. Moreover, the choice of 4 bit for the path number is conservative because using more paths in parallel does not improve the performance of the protocol. This protocol also does not discuss any dynamic addition or removal of the paths. Using a very large number of many parallel paths at connection establishment slows down the performance of the protocol. The protocol also introduces a 32 bit Association Sequence Number(ASN), unique among all the paths, as shown in Figure 2.2 which is a counterpart of TSN in the original SCTP scheme.

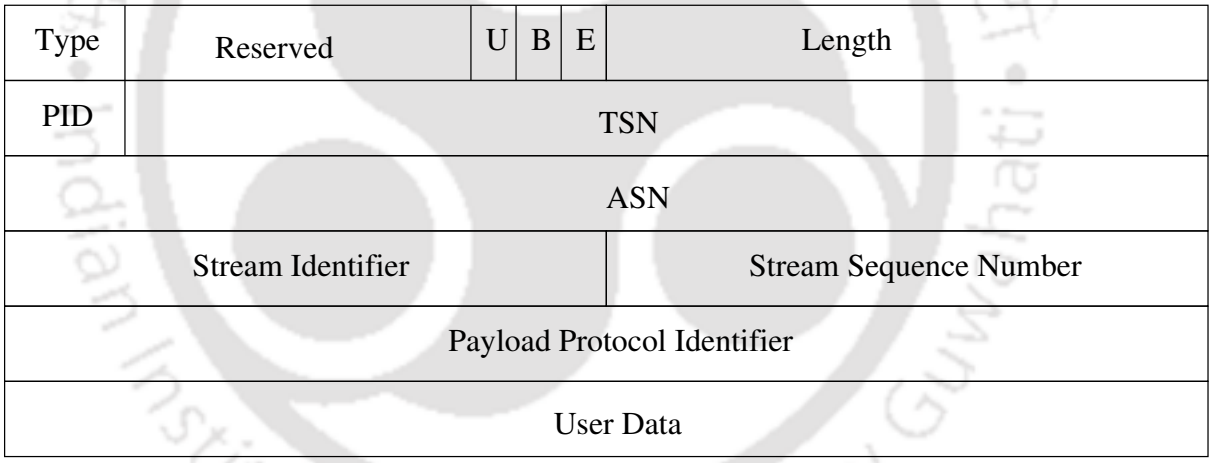


Figure 2.2: Modified cmpSCTP Chunk Header

The Selective Acknowledgement (SACK) structure has also been modified as shown in Figure 2.3. The authors propose to maintain multiple buffers for each available path and also maintain an independent advertised receiver window for each path. Its SACK contains a time stamp to indicate the order of arrival of SACK from each path.

This protocol suggests that retransmissions be done on the path with the lowest probability of error though the actual measurement of the probability of error is left as an open issue. The protocol proposes the flexible path management which can be useful in mobile scenarios. During handover, the protocol suggests that the connections be maintained with the neighboring base

Type = 3	Chunk Flags	Chunk Length
Cumulative ASN Acknowledgement		
Time Stamp		
PID # 1	Advertised Receiver Window	
PID # 2	Advertised Receiver Window	
Number of Fragements (N)		Number of Duplicates (M)
GapAck Block #1 Start		GapAck Block #1 End
.....		
GapAck Block #N Start		GapAck Block #N End
Duplicate ASN #1		
.....		
Duplicate ASN #M		

Figure 2.3: Modified cmpSCTP SACK Header

station as well. This will reduce the call drop probability during the handover process. However, supporting this feature would require further modifications in the protocol and increased signaling.

2.1.5 Best Load Sharing SCTP

Joe et al. [34] have presented a load sharing technique in SCTP to support simultaneous data transfer on multiple paths. The authors call it *Best Load Sharing* in SCTP. Here, a path state table is built for each path where the availability of a path is represented by a dimensionless quantity M. This quantity depends upon various parameters like the cost (C), time (T), bandwidth (B), security (S) etc.

$$M = C.P_1 + D.P_2 + T.P_3 + B.P_4 + S.P_5 + ... \tag{2.2}$$

All these quantities are assumed to be normalized and dimensionless with values between 0 and 1. P_i is the weighting factor for each component such that $\sum P_i = 1$. However, this work does not

elaborate upon the methodology of the measurement of the various parameters.

The use of multiple paths introduces false holes in the sequence number. This causes unnecessary fast retransmissions at the sender end. To prevent unnecessary fast retransmissions caused by path switching, the authors have proposed a *Split Fast Retransmission* (SFR) algorithm. This is actually inherited from the SFR Algorithm of CMT-SCTP. Following this strategy, the fast retransmission algorithm is run independently on each candidate path. Each path maintains an ordered set of TSNs and a mapping of the respective TSNs being sent on each path.

2.1.6 Wireless Multi-path Multi-flow SCTP

Wireless Multi-path multi-flow SCTP (WM^2-SCTP) [35] takes into account the respective QoS of different streams. It groups them together based upon their QoS requirements and selects the suitable path accordingly. Like cmpSCTP, this implementation also uses a multi-buffer architecture i.e. every connection and each subflow will have its own send buffer.

Type = 3	Chunk Flags	Chunk Length
Cumulative ASN Acknowledgement		
PID # 1	Cumulative PSN Ack	
FID # 2	Cumulative FSN Ack	
Advertised Path Flow Receiver Window		
Number of Fragements (N)		Number of Duplicate FSN (M)
GapAck Block #1: FSN Offset Start		GapAck Block #1: FSN Offset End
.....		
GapAck Block #N: FSN Offset Start		GapAck Block #N: FSN Offset End
Duplicate FSN #1		
.....		
Duplicate FSN #M		

Figure 2.4: Modified WM^2-SCTP SACK Header

However unlike cmpSCTP, it uses three level sequence numbering. A subflow identifier (FID) and a subflow sequence number (FSN) have been added to identify the subflow on each path as shown in Figure 2.4 . The corresponding SACK structure has also been modified as shown in Figure 2.5. FIDs are assigned depending upon the priority of the flow. Smaller FIDs takes precedence while selecting the path.

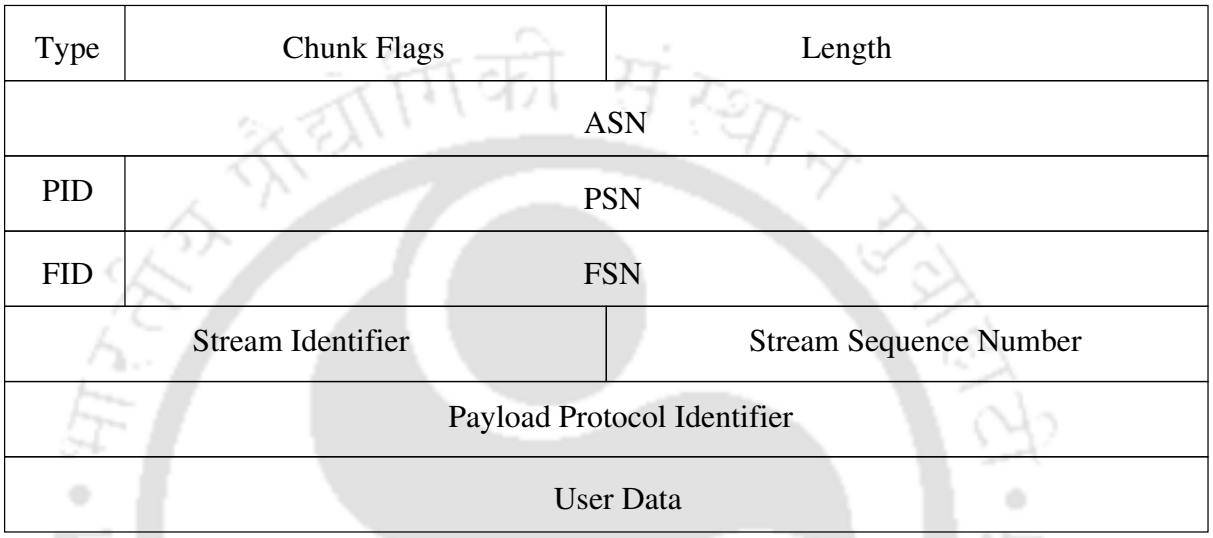


Figure 2.5: Modified WM^2-SCTP Chunk Header

In this paper, the authors have not discussed the use of Path Sequence Number (PSN) in the SACK. Apparently, the FSN is able to identify the flow on each path. The FID is also able to identify the path to which the flow belongs. Nevertheless, the concept of dividing data based upon the QoS requirements is an interesting idea and requires due consideration in future work.

2.1.7 Multi-Path TCP (MPTCP)

Currently IETF is working on Multipath-TCP as a multipath alternative to TCP [36,37]. Unlike the traditional TCP Protocol, MPTCP divides the transport layer in two sublayers as shown in Figure 2.6.

The MPTCP layer just below the Application layer does path management and scheduling while the subflow layer implements the interface with the MPTCP layer and performs congestion control. Path Management is responsible for detecting and using multiple paths between the source and the destination. The scheduling function breaks the byte-stream received from the application into segments which are transmitted on one of the available subflows. The subflow interface takes

care of transmitting the data from the scheduling component over the specified path. MPTCP implements two levels of sequence numbers, one for the congestion level sequencing and the other for subflow level sequencing. MPTCP has also been supplemented with coupled congestion control. More details on coupled congestion control are presented in Section 2.3.1.

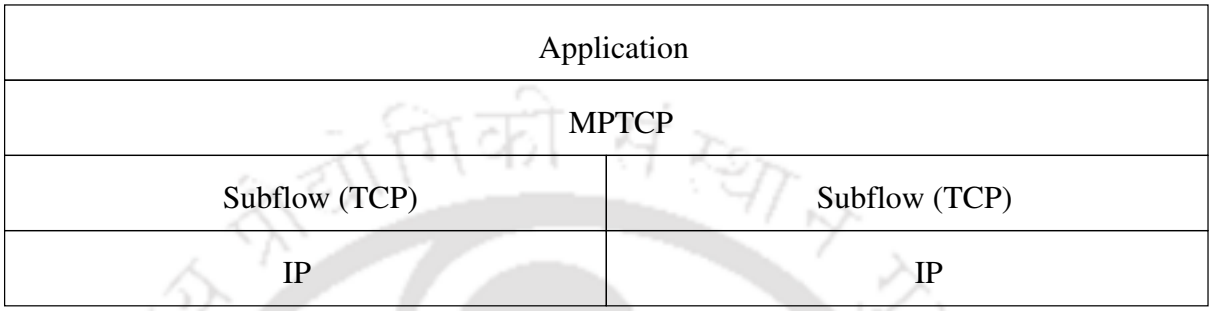


Figure 2.6: MPTCP Protocol Stack

We have summarized the various features of above protocols in Table 2.1. Several multipath protocols have been proposed in the literature but most of them face the fairness issues. They tend to capture unduly more bandwidth on the shared links. The protocol may have to deal with paths with very different characteristics. It is possible that one of the interface used by the protocol is wired interface while the other one is wireless. Hence, path management for multipath protocols is a major problem. Because, the underlying paths may have very different properties, flow management for these available paths is another potential area of research.

2.2 Flow Division at Source

The objective of flow division at source is to divide the data flow from the source on a number of available paths so as to optimize some selected end-to-end performance parameter.

The concept of distributing the flow at the source has been discussed in various contexts in the literature. The distribution of packets on multiple paths for ATM networks has been discussed in [38]. The authors have established that a per packet flow distribution performs better than a per connection flow distribution. Load balancing in multi-path source routing [39] in wireless Ad-Hoc networks has been proposed in [40]. An MPLS network based load division mechanism has been proposed in [41]. The problem of flow division at source based on shared bottleneck measurement has been studied in [42]. In this work, packet delay has been used as the bottleneck link measure.

Table 2.1: Features of various Multipath Transport Protocols reported in literature

	Load Sharing SCTP [28]	Independent per path congestion control SCTP [29]	CMT [17]	cmpSCTP [33]	Best Load Sharing SCTP [34]	WM^2-SCTP [35]	MPTCP [36]
# of sequence numbers	1	1	1	2	1	3	2
Congestion Control	TCP based	TCP based but independent for each path	TCP based	TCP based	TCP based	TCP based	Resource Pooling based
Changes in SACK header	Yes	No	No	Yes	No	Yes	-
Changes in chunk header	No	No	No	Yes	No	Yes	-
Retransmission Policies	Same as SCTP	Same as SCTP	5 different policies proposed	On path with smallest probability of error	Same as SCTP	Same as SCTP	Same as TCP
Flow division	None	None	None	None	Based on link parameters	QoS based	None
Fairness on shared link	No	No	No	No	No	No	Yes

Kitatsuji et al. [43] have presented flow distribution based on the traffic characteristics over the network.

2.3 Congestion Control

In computer networks, congestion arises when a link is carrying so much data that it is not able to provide the promised quality of service to the end users. A network under congestion exhibits symptoms like increase in queuing delay due to large queues at routers, higher packet losses due to overflow of buffers at network components like the intermediate routers and the inability of the network to provide access to new users. Congestion causes severe reduction in the network throughput and underutilization of valuable networking resources. However, if we do not carry enough traffic on the network, even then the network resources remain underutilized. A typical

scenario of throughput with increasing load in the network is shown in Figure 2.7. The region of the curve where throughput decreases with increase in load is called *congestive collapse*. In an ideal situation, we would like our network to operate somewhere between the points 'A' and 'B' as shown in Figure 2.7.

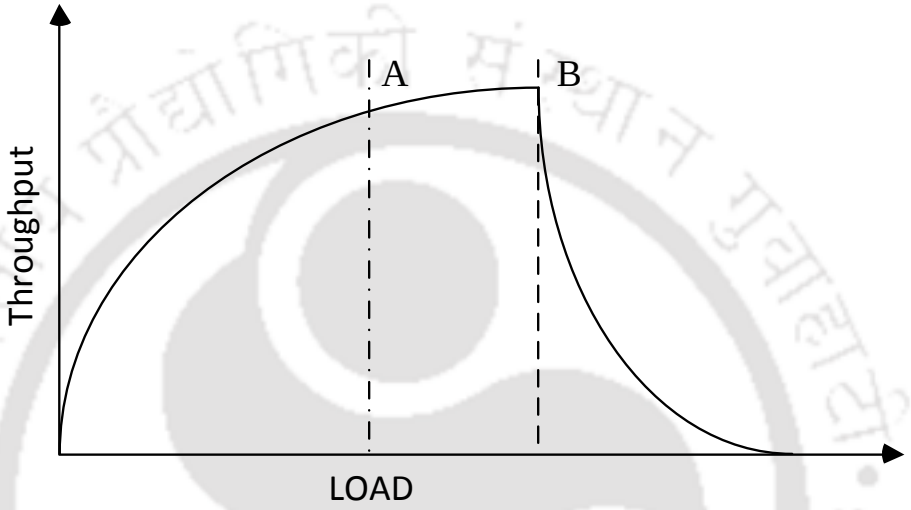


Figure 2.7: Throughput Vs Load for a network

Congestion control has been a primary focus area in packet networking. Various congestion control strategies like window based control, rate based control, slow start and congestion avoidance, scheduling and feedback based control have been proposed in the literature. A congestion control mechanism has two components - congestion avoidance and congestion recovery. Congestion avoidance is the strategy to keep the network at its optimal capacity so that it does not get into congestion. Congestion recovery is required to recover from the congestion in case it does arise in the network. Congestion recovery is required because transient changes in the traffic might drive the network into congestion even when congestion avoidance procedures are in use. Several congestion control algorithm have been proposed over time in literature. A broad classification [44] of various congestion control algorithms has been shown below in Figure 2.8 .

In Internet TCP and UDP are the most widely accepted transport protocols. UDP has no congestion control by design. However, TCP is a reliable transport protocol and several congestion

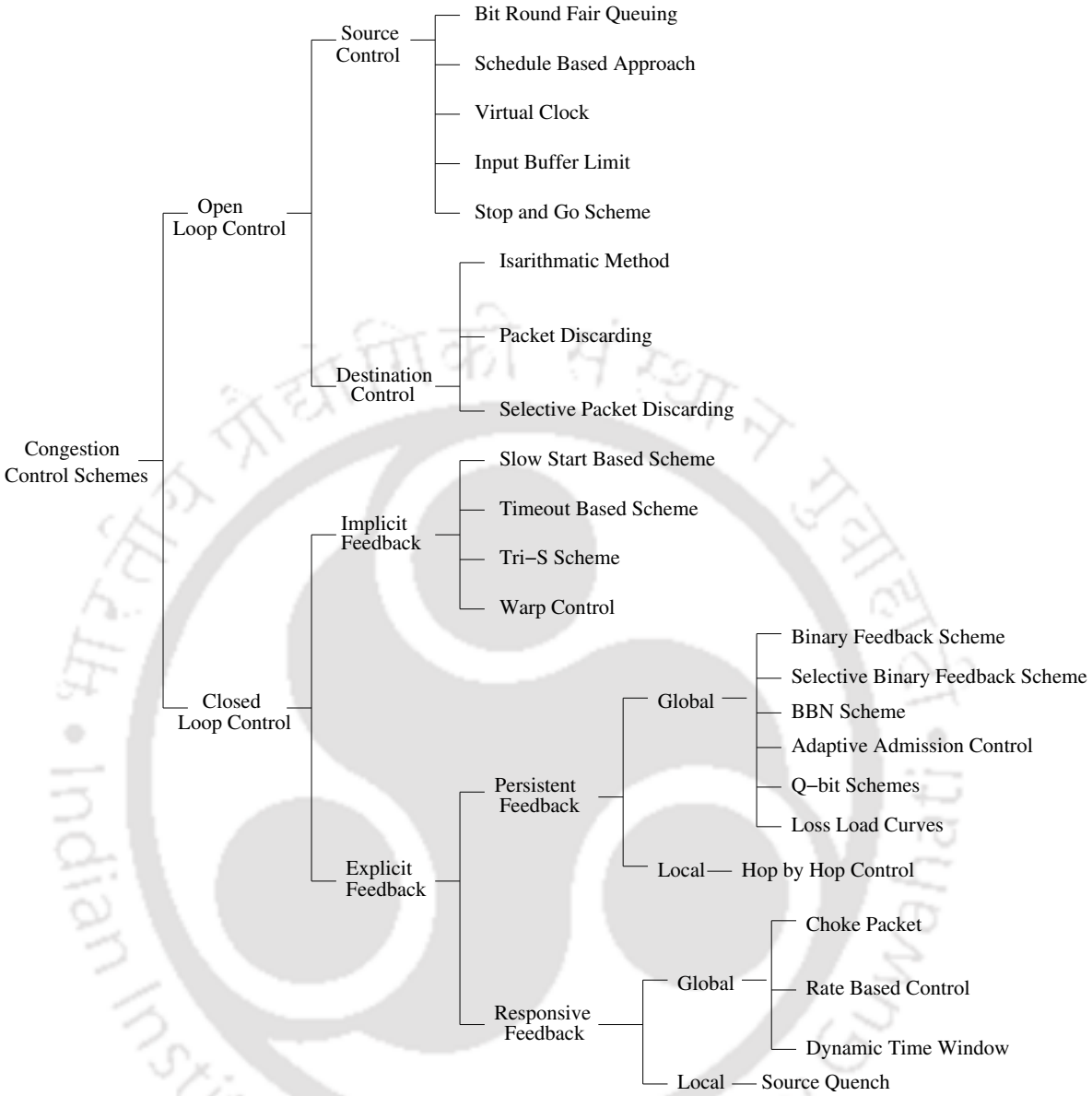


Figure 2.8: Classification of existing Congestion Control Algorithms [44–50]

control variants have been proposed for TCP. TCP uses a window based reactive approach to handle congestion control. IETF has standardized TCP congestion control in RFC 2581 [51]. Several improvements have also been proposed for TCP congestion control [52–54].

It may be noted that every TCP congestion control algorithm consists of three phases, namely Slow Start, Congestion Avoidance and Congestion Detection. In the slow start phase, TCP starts with sending one segment and increases congestion window by 1 for each Ack received i.e. it doubles its congestion window every RTT. Once this data rate reaches a pre-set threshold level (called slow

start threshold or *ssthresh*), TCP starts increasing its data rate linearly, i.e. increases window size by one every RTT, until it encounters a packet loss event. This is called the congestion avoidance phase. In Congestion Detection, whenever TCP detects any packet loss, it assumes that this loss is due to congestion and reduces its congestion window to half of the current value. This approach poses potential problems when links are not reliable, such as when wireless channels are being used. The wireless channel is inherently lossier than a wired network. Unlike a wired channel, a packet loss event in a wireless channel may be caused either by congestion or because of bit errors in a packet. Hence, reducing the congestion window on every packet loss in a wireless channel would probably be inefficient [54]. Hence, TCP for wireless networks should not halve the bandwidth on packet loss but should instead decrease it in the proportion to the available bandwidth on the channel.

Though SCTP was proposed with support for multihoming, it only uses one path at a time for data transfers. The data is originally transferred on the primary path. If the primary path fails, SCTP switches to any alternative secondary path. Once the primary path is restored, SCTP starts sending data on the primary path once again. SCTP congestion control is largely based on TCP congestion control.

2.3.1 Coupled Congestion Control

Recently IETF is working on Multipath TCP (MPTCP), a multipath variant of TCP. TCP like congestion control for multipath protocols exhibits the problem of unfairness on shared link and harms the fellow flows by unduly capturing more bandwidth on shared links. Raiciu et al. [37, 55] have proposed coupled congestion control which is based on a Resource Pooling Principle [56]. This algorithm assumes that all the resources available as a single pool of resource and it does not change the congestion window of each path independently. According to coupled congestion control, the congestion window of each path is modified as follows:

The algorithm says that on receipt of new ack on i^{th} subflow corresponding congestion window should be increased by

$$min(\alpha \times bytes\_ack \times MSS_i/tot\_cwnd, bytes\_ack \times MSS_i/cwnd_i) \tag{2.3}$$

where

α	= factor of aggressiveness,
tot_cwnd	= sum of ssthresh for all subflows,
MSS	= Maximum Segment Size,
$bytes\_ack$	= number of bytes acked in previous ACK,
$cwnd_i$	= congestion window of i^{th} path.

A fully coupled congestion control algorithm on shared path exhibits a problem that whenever the smaller window decreases, it is truncated all the way to "1", hence the algorithm spends most of the time using only one of the paths [37]. This is called "flappiness". The proposed increase of congestion window as in (2.3) helps in reducing the flappiness in the network but reduces the resource pooling as well. An alternative congestion control algorithm for MPTCP is still an open area of research.

In this chapter, we have discussed various literatures related to multipath protocols, and multipath congestion control. We have also introduced the concept of flow division at source and presented various works available in literature in these areas. Various multipath variants of SCTP have been discussed with their relative merits and demerits. Currently work is going on to enable TCP for multipath environment. A novel congestion control, Coupled Congestion Control, based on resource pooling principle has also been proposed to eliminate the unfairness of multipath protocols on shared links.

3

Optimization Models

Contents

This chapter explores various methods for optimization of flow division at the source. This is motivated by the fact that optimizing flow division at the source may significantly improve the overall utilization of network resources when multipath transport protocols are used. Various optimization models are suggested for this and their relative performances are compared and discussed. We also compare the performance of these optimizations for both end-to-end and link-by-link error recovery approaches.

3.1 Introduction

Stream Control Transmission Protocol (SCTP) [6,25,26] can be easily modified to support concurrent multipath multihomed connections between the end users. Multipath TCP (MPTCP) [36] has been proposed as a multipath variant of TCP which can also support concurrent usage of multiple paths between source and destination. A basic requirement of such a multihomed, multipath transmission would be to properly divide the data stream at the source into individual flows for the different paths which can then be properly recombined at the destination. To optimally utilize the network resources, it would then be essential to manage and adjust the flow division (possibly in a dynamic fashion) based on the performance of each component path. The performance of a path would in turn depend on the performance of the individual nodes and links on that path. The data should be divided so that the overall end-to-end performance is improved. This kind of usage will require optimized flow division among the candidate paths in order to utilize the network resources efficiently. Optimizing flow division at the source requires division of the original incoming flow between the available candidate paths so that some selected end-to-end quality of service (QoS) parameter is satisfied. The QoS parameter chosen for this may depend on the nature of the application since different applications may have different perceptions about quality.

In the subsequent sections of this chapter, we propose various optimization approaches for flow division where the objective is to improve the overall quality of service being provided to the user along with better utilization of the network resources. We analyze and compare three ways in which these optimizations can be done. The performance of these optimization algorithms will also depend upon the error recovery approaches. Traditional transport layers use end-to-end approach for flow control and error recovery. Performance of end-to-end error recovery is found to degrade for lossy channels. Link-by-link error recovery is reported to perform better under such conditions [57].

We also compare the performance of these flow division approaches for the multipath environment using both end-to-end and link-by-link error recovery schemes.

The choice of optimization approach also depends upon the nature of the application, i.e. whether the applications are delay sensitive or delay insensitive in nature, as they have inherently different QoS requirements [19]. These optimization approaches are easily extendable to systems running protocols like SCTP and MPTCP as their transport protocols, though changes in the standard protocol stack will be required to achieve this. We have considered optimized flow division for both delay sensitive and delay insensitive applications. For delay sensitive applications, not only is the end-to-end delay required to be minimized but buffering and proper replaying of the data at the destination should also be considered. However, a delay insensitive application is the one where the delay does not appear as a QoS parameter. For delay insensitive applications, flow division may not actually require minimization of the end-to-end delay. Instead, it would typically require the end-to-end delay to be kept under some predefined limit while maximizing the throughput. Flow division satisfying such QoS constraints (i.e. bounded end-to-end delay) has also been investigated.

We consider here the problem of optimally dividing the flow between multiple disjoint paths at the source. Since different applications may have different QoS requirements, we also consider the nature of application for the optimal flow division. Based upon the QoS requirements, applications can be divided into two categories namely *delay Sensitive* and *delay Insensitive* applications.

3.2 Delay Sensitive Applications

Interactive services and multimedia applications where user satisfaction strongly depends upon the delay in transferring data from the source to the destination fall in this category. These require the overall end-to-end delay to be as low as possible. When the overall flow is divided on multiple parallel paths, the delay on one path will degrade the quality of the overall application and may cause excessive receiver buffer blocking and higher buffer size requirement at the receiver. Asymmetrical (i.e. highly variable) delays on the paths will also lead to higher delay jitter. In this work, we have not explicitly considered receiver buffer blocking and delay jitter but these problems may be addressed by proper selection of the buffer size at the receiver [58]. Here, we consider two optimization approaches for delay sensitive applications:

3.2.1 Min-Max Approach – Minimizing the Maximum Average Path Delay

The basic motivation behind *Min-Max Approach* is to keep the worst case end-to-end delay on the candidate paths as small as possible. When a flow is subdivided into a number of sub-flows, the path with the maximum end-to-end delay will primarily dictate the overall performance of the system. This implies that the traffic division at the source should keep this worst case delay as small as possible [59][‡].

Let $W = \{w_1,w_2,.....,w_N\}$ be the set of end-to-end delays for N candidate disjoint paths where each candidate path carries a sub-flow of λ, i.e $\{\lambda_1, \lambda_2, ..., \lambda_N\}$ packets per second. The min-max optimization can be achieved as follows:

minimize:

$$max(w_i) \qquad \forall i = 1, 2, ..., N \tag{3.1}$$

subject to:

$$\sum_{i=1}^{N} \lambda_i = \lambda \tag{3.2}$$

$$\lambda_i \geq 0 \tag{3.3}$$

3.2.2 Minimizing the Weighted Average Delay

In this approach we minimize the mean weighted average delay of all the paths weighted by the fractional load that they carry. We assume $W = \{w_1,w_2,.....,w_N\}$ as the set of end-to-end delays for N candidate disjoint paths where each candidate path carries a sub-flow of λ, i.e $\{\lambda_1, \lambda_2, ..., \lambda_N\}$ packets per second. The flow division to achieve this can be represented as follows:

minimize:

$$\sum_{i=1}^{N} \left[\frac{\lambda_i}{\lambda} \cdot (w_i) \right] \tag{3.4}$$

subject to:

$$\sum_{i=1}^{N} \lambda_i = \lambda \tag{3.5}$$

$$\lambda_i \geq 0 \tag{3.6}$$

[‡]Author's joint publication.

3.3 Delay Insensitive Applications

Unlike delay sensitive applications, delay insensitive applications like FTP or HTTP transfers do not have strict delay requirements. However, practical considerations (i.e. reasonably satisfactory user experience) will still require the delivery of data within a maximum pre-agreed end-to-end delay while using the network resources optimally. With concurrent usage of multiple disjoint paths, it is required to carry as much data as possible subject to the agreed upon delay constraint for the application. This leads to the following optimization problem:

maximize:

$$\sum_{i=1}^{N} \lambda_i \tag{3.7}$$

subject to:

$$w_i \leq T_i \qquad \forall i = 1, 2, ..., N \tag{3.8}$$

$$\lambda_i \geq 0 \tag{3.9}$$

where,

T_i is the agreed upon end-to-end *Target Delay* for i^{th} path,

w_i is the end-to-end delay for i^{th} path,

λ_i is the traffic rate for i^{th} path,

N is the total number of candidate paths.

3.4 Network Model for Flow Optimization

A general network graph may be represented as *G(V, E)* where V is the set of network nodes and E is the set of directional links connecting the nodes. We assume that several disjoint paths exist between the source and the destination. The flow of interest is subdivided on these paths so as to use the network resources efficiently. Here, we study the performance of optimal flow division for end-to-end error recovery and compare this with the case where link-by-link error recovery is being done. The latter will be more efficient but will also increase the complexity of the system.

Consider a physical network as shown in Figure 3.1. This provides N disjoint paths which may be used by a particular target flow between Source (S) and Destination (D). Let there be L_i nodes

on the i^{th} path as shown in Figure 3.1. Note that S and D are common to all of these paths. The packets of the target flow between the source and the destination are assumed to arrive following a *Poisson* arrival process [60] at rate λ packets per second. Note that the *Poisson* assumption here helps significantly in analytical tractability. This flow is divided on N parallel paths where the sub-flow on the i^{th} path has rate λ_i packets per second. The packets of the flow are randomly divided so that each sub-flow has the indicated packet rate. Moreover, by the property of *Poisson* process, this random division implies that the packet arrival process of each sub-flow will also be *Poisson* in nature.

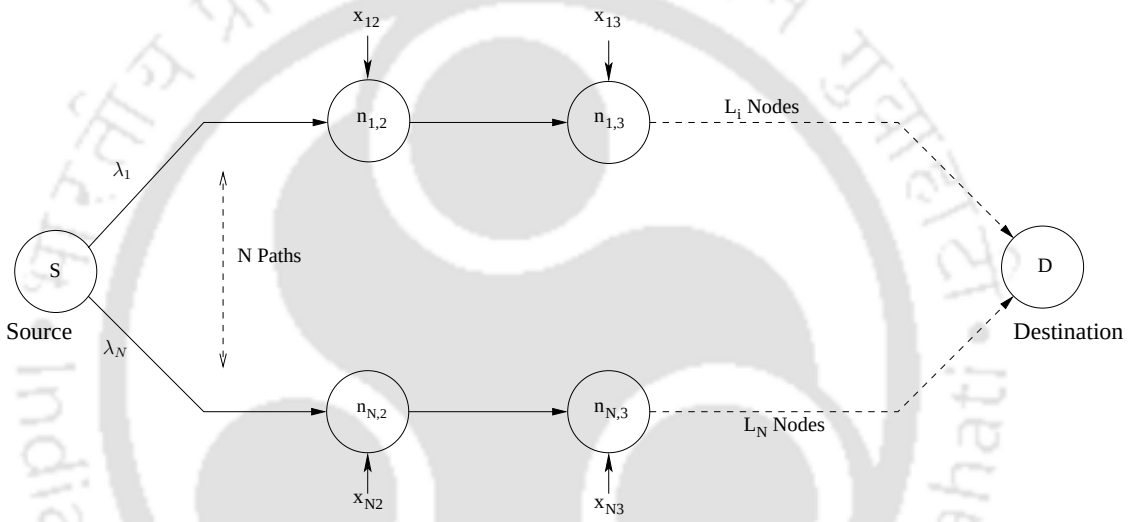


Figure 3.1: Network Graph for Flow Division between Source and Destination

Apart from the traffic of interest, intermediate node will also contribute other traffic for the subsequent link on the path. As shown in Figure 3.1, we assume $x_{i,j}$ to be the background traffic (in packets/seconds) entering at j^{th} node of i^{th} path ($n_{i,j}$) which is destined for the link (j, $j+1$). Let $p_{i,j}$ be the probability of packet error and $b_{i,j}$ (in bits/second) be the data rate for the link (j, $j+1$) of i^{th} path. We assume fixed size packets of s bits in our subsequent analysis though the approach can be extended for variable packet sizes as well.

3.4.1 End-to-End Error Recovery

In end-to-end error recovery, for every packet loss on any link on the path, the packet is retransmitted from the source. Let us consider a two node network as shown in Figure 3.2:

Let p_1 be the probability of packet error at the link between source and destination. For the

Figure 3.2: Two node network with a source and destination

successful delivery of a packet to the destination, the source will make $\frac{1}{(1-p_1)}$ transmission attempts on an average. Similarly, if we consider a three node network, the average number of transmissions from the source will be $\frac{1}{(1-p_1)(1-p_2)}$ where p_1 and p_2 are the probability of packet error for the two links.

For every packet, let n_i be the mean number of packet transmissions by the source on i^{th} path of the network.

$$n_i = \frac{1}{\prod_{k=1}^{L_i-1}(1-p_{i,k})} \tag{3.10}$$

The fraction of flow offered by source as in (3.10) will progressively decrease at intermediate nodes due to the errors. Hence the effective traffic at j^{th} node of the i^{th} path will be given by

$$\lambda_{i,j} = \frac{\lambda_i}{\prod_{k=j}^{L_i-1}(1-p_{i,k})} \tag{3.11}$$

Taking background traffic also into account, the total arrival rate of packets ($\Lambda_{i,j}$) to be carried between the j^{th} and the $(j+1)^{th}$ node of i^{th} path is

$$\Lambda_{i,j} = \lambda_{i,j} + x_{i,j} \tag{3.12}$$

The service rate $\mu_{i,j}$ for a packet on this link will be

$$\mu_{i,j} = \frac{b_{i,j}}{s} \tag{3.13}$$

With packets of fixed size, the corresponding average total delay [61] at the node feeding this link may be obtained from M/D/1 queuing results:

$$w_{i,j} = \frac{2\mu_{i,j} - \Lambda_{i,j}}{2\mu_{i,j}(\mu_{i,j} - \Lambda_{i,j})} \tag{3.14}$$

Hence, the mean end-to-end delay for a packet on the i^{th} path will be

$$w_i = n_i \cdot \sum_{j=1}^{L_i - 1} w_{i,j} \tag{3.15}$$

For the end-to-end delay given by (3.15), we have assumed that the delay seen by the acknowledgement traffic is much less than the queuing delays and can be neglected and that the propagation delays for the links are also negligible.

3.4.2 Link-by-Link Error Recovery

In link-by-link error recovery, every node keeps track of the error on the link and unlike end-to-end error recovery, retransmits the packet in case of any packet error.

Considering the network as in Figure 3.1, assuming perfect link-by-link error recovery, the corresponding node will offer an effective flow of $\lambda_{i,j}$ to be carried on the link.

$$\lambda_{i,j} = \frac{\lambda_i}{(1 - p_{i,j})} \tag{3.16}$$

Hence the mean arrival rate of packets to be carried between the j^{th} and the $(j+1)^{th}$ node of i^{th} path is

$$\Lambda_{i,j} = \lambda_{i,j} + x_{i,j} \tag{3.17}$$

The service rate $\mu_{i,j}$ for a packet on this link will be

$$\mu_{i,j} = \frac{b_{i,j}}{s} \tag{3.18}$$

Same as in (3.14) the average total delay will be:

$$w_{i,j} = \frac{2\mu_{i,j} - \Lambda_{i,j}}{2\mu_{i,j}(\mu_{i,j} - \Lambda_{i,j})} \tag{3.19}$$

Hence, the mean end-to-end delay for a packet on the i^{th} path will be

$$w_i = \sum_{j=1}^{L_i-1} w_{i,j} \tag{3.20}$$

3.5 Performance and Results

For performance studies, we consider a system where all the link parameters are available at the source. The optimizations are done with AMPL (A Modeling Language for Mathematical Programming) using Minos version 5.51 as the solver [62, 63].

We study a situation where the network provides three disjoint paths between Source (S) and Destination (D) with $L_i = 8$ nodes on each path. The target flow from S to D is assumed to be *Poisson* with a fixed packet size of 1000 bytes. The link rates for each link of paths 1, 2 and 3 are 5, 10 and 15 Mbps, respectively. The external background traffic added at each node of paths 1, 2 and 3 is also assumed to be *Poisson* with mean arrival rates of 50, 100 and 200 packets per second respectively. The packet error probability is assumed to be 10% for all the links.

We have observed how the flow is divided among the candidate paths for different rate of arrival traffic at the source and QoS requirements. We have also measured the end-to-end delay and the system throughput for different approaches of flow division.

3.5.1 Delay Sensitive Applications

3.5.1.1 Min-Max Approach

Figure 3.3 and Figure 3.4 show the percentage flow division on the three paths for both end-to-end and link-by-link error recovery approaches respectively.

From the system's point of view, given a value of the source traffic (and the other parameters mentioned earlier), the *Min-Max Approach* suggests a way in which the flow can be divided among the given paths. This also ensures that corresponding mean end-to-end delay is equal for all the candidate paths.

This flow division approach may not always use all the available disjoint paths. On the paths which are actually used, the flow division will be such that the delay along each path will be equal. If any path offers a lower (higher) delay than the others, then flow to that path should be incrementally increased (decreased) while flows to the others should be correspondingly decreased (increased) until all paths offer the same average delay. Operationally, when the offered traffic is

very low, only the lowest delay path will be used. Paths with progressively higher delays would be added as the offered traffic increases. Here, the *Min-Max Approach* ensures that the average delays on all the paths in use will be the same for each value of the offered traffic.

It is interesting to observe how the *Min-Max* operating point is iteratively reached. Figure 3.5 shows a sample trend of how the different path delays change for successive iterations of the optimization when the source traffic is kept at 1430 packets/s. As mentioned earlier, the delays on the different paths eventually converge to identical values as the final operating point (final percentage flow divisions) is reached.

Figure 3.6 shows the mean end-to-end delay for both end-to-end and link-by-link error recovery approaches. As expected, the link-by-link recovery scheme performs better than the end-to-end error recovery approach. It is also interesting to note that as the input rate increases the performance degrades for both schemes but the degradation is significantly worse for the end-to-end system. This can be easily explained with the help of (3.11). Unlike the link-by-link error recovery system, a packet loss in an end-to-end system has a cascading effect and increases the traffic on all links preceding the link where the packet is lost. This causes the rapid increase in the end-to-end delay experienced by a packet.

3.5.1.2 Weighted Average Minimization

For this, we consider the same system with three paths as before. Figure 3.7 and Figure 3.8 show the percentage flow division on the three paths both for end-to-end and link-by-link error recovery respectively when the *Weighted Average Minimization* approach is followed. Figure 3.9 gives the weighted average sum of the end-to-end packet delay experienced by three paths. The error-bars in the figure represent the minimum to maximum delay on three paths.

It is interesting to note that, given a certain volume of source traffic, the two approaches (Min-Max and weighted average) suggest different flow division between the paths. The average delay with the Weighted Average approach is lower than what can be achieved through *Min-Max Approach*. However, the path delays are no longer identical and the variation between them becomes progressively larger for increasing source traffic. This will lead to more difficult buffer management for reconstruction at the destination even though its average delay will be lower than that obtainable through the *Min-Max Approach*.

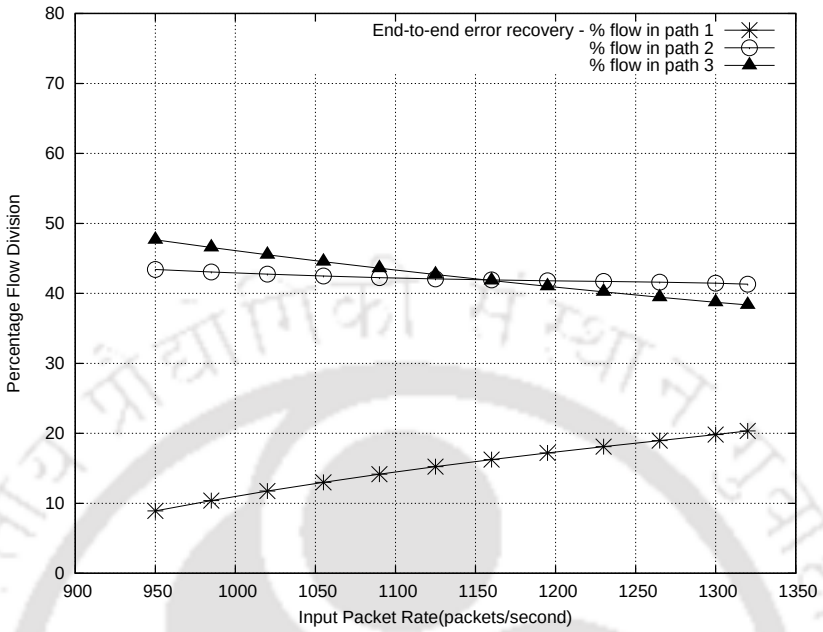


Figure 3.3: Percentage Flow Division for varying Input Packet Rate (S to D) with the Min-Max Approach for end-to-end error recovery

3.5.2 Delay Insensitive Applications

We consider the same system with three paths as before with a Target Delay set for each path, where for convenience, we choose the same target delays for each path in the example considered here. We evaluate the maximum overall throughput that can be delivered by the system and the percentage flow division on the three paths. The percentage flow divisions for end-to-end and link-by-link recovery approaches are shown in Figure 3.10 and Figure 3.11 respectively as a function of the Target Delay. Figure 3.12 shows the changes in overall throughput for the two approaches with varying Target Delay. Assuming the Target Delay to be the same for all the paths is a reasonable choice because for typical applications, when the transport layer divides the data of the same application on the multiple paths, the delay requirements will nominally be the same from all the paths for that application.

For this system, Figure 3.10 shows that for Target Delays less than approximately 12 ms, *path-1* does not carry any data as this is the delay that the path has because of the background traffic even when it does not carry any of the target flow. However, for the end-to-end recovery approach

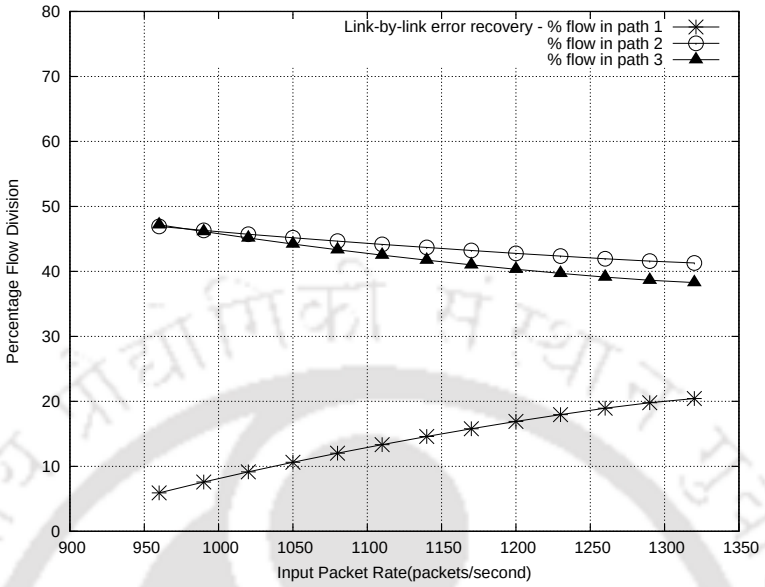


Figure 3.4: Percentage Flow Division for varying Input Packet Rate (S to D) with the Min-Max Approach for link-by-link error recovery

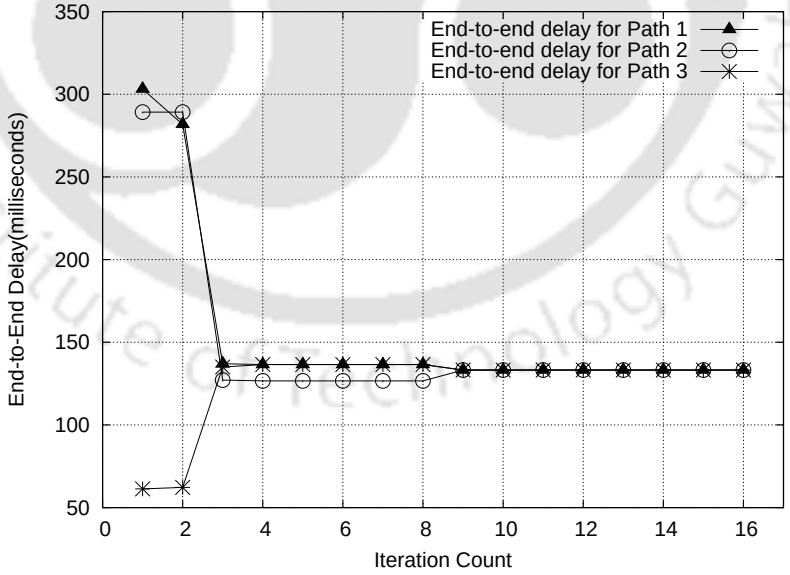


Figure 3.5: Average Path Delays for Successive Iterations of the Min-Max Approach (Input Traffic=1430 packets/second)

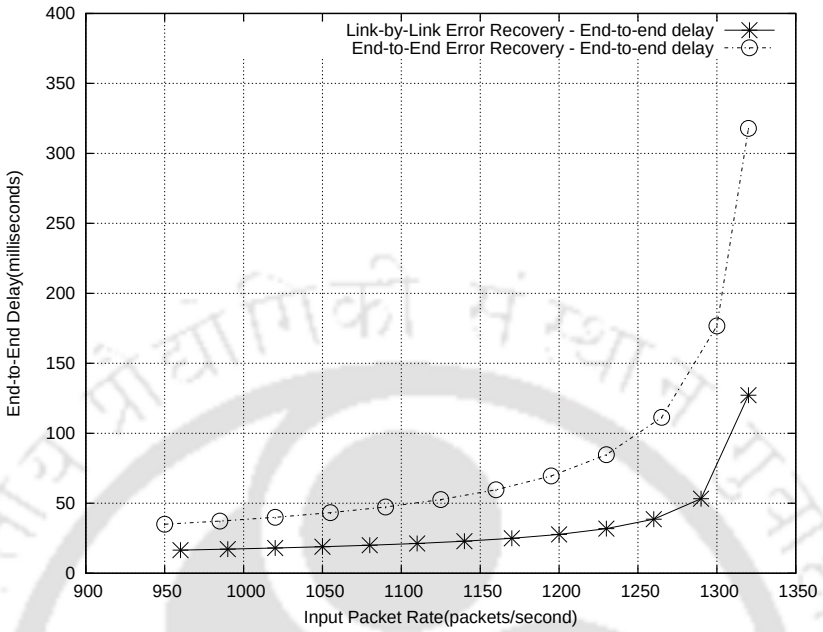


Figure 3.6: Average End-to-End Delay for varying Input Packet Rate (S to D) with the Min-Max Approach

(Figure 3.11) *path-1* is not capable of carrying any data for Target Delay values less than 27 ms. For Target Delays higher than this value, the fraction of the traffic carried by *path-1* increases while the fraction carried by the other two paths decrease.

It may be noted in Figure 3.12 that the overall throughput increases rapidly with increasing Target Delays until it reaches a value of about 1300 packets/second. Increasing the Target Delay further leads to only a marginal increase in the throughput. This threshold indicates that the ability of the network to carry the target flow exhibits a saturation beyond which it is not possible to get more throughput by allowing for higher target delays.

For this system, Figure 3.12 also shows that the end-to-end error recovery approach supports less data rate for the same network conditions. The data rate supported by the network for same target delay is also less for the end-to-end error recovery. Moreover, the maximum data rate supported by the network increases much more rapidly for link-by link error recovery with the increase in accepted Target Delay.

3.6 Conclusion

In this chapter we have considered flow division at source to improve network utilization when multiple disjoint paths are available between the source and the destination and the transport protocol allows them to be used in parallel. Various flow division approaches at source have been proposed and they are compared for two error recovery approaches: end-to-end error recovery and link-by-link error recovery. It has been observed that link-by-link error recovery performs better than the end-to-end error recovery. However, in case of link-by-link error recovery, the intermediate nodes have to perform extra work to recover from data loss at every link. Hence, link-by-link error recovery scheme requires more complex network design and increased capability in the intermediate nodes. It is also noted that for delay sensitive applications, the weighted average approach gives smaller average end-to-end delay but the delay variation among the paths tends to be larger. This introduces larger delay jitter in the packets and complicates the receiver buffer design.

Finally, it may be noted that the techniques given here require that the entire network statistics should be known to the sender so that it can use this information to decide how to subdivide the target flow into various component sub-flows. Unfortunately, in an Internet like scenario, this information may not always be available. We present practically implementable heuristics for some of these techniques in later chapters.

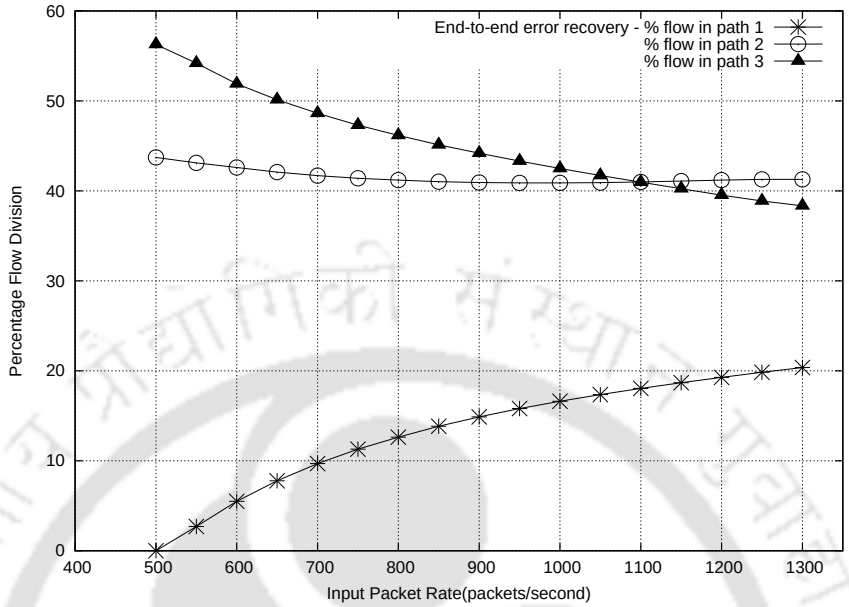


Figure 3.7: Percentage Flow Division for varying Input Packet Rate (S to D) with the Weighted Average Approach for end-to-end error recovery

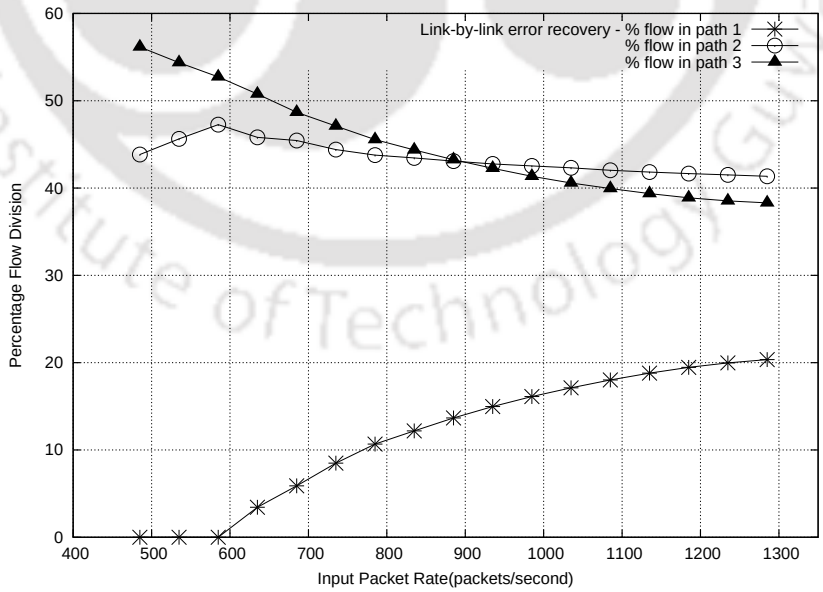


Figure 3.8: Percentage Flow Division for varying Input Packet Rate (S to D) with the Weighted Average Approach for link-by-link error recovery

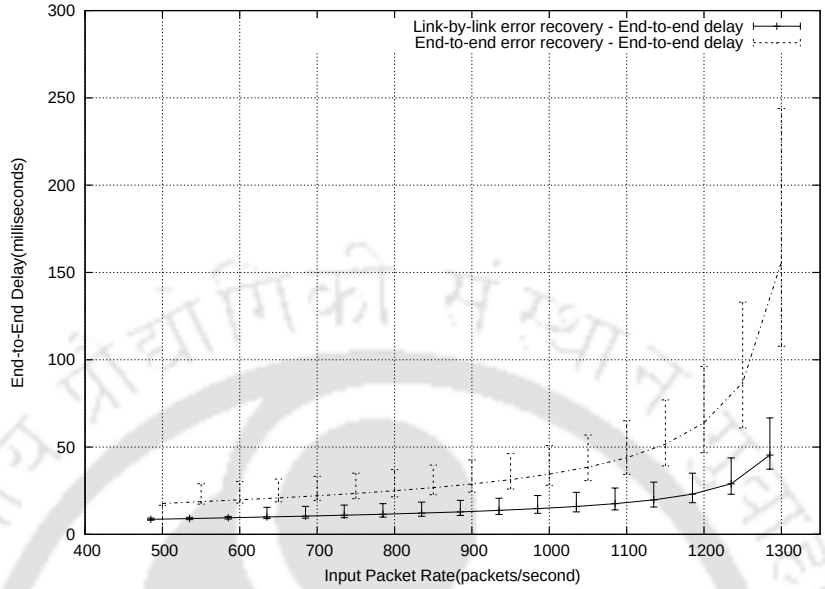


Figure 3.9: Average End-to-End Delay and Delay Variation for varying Input Packet Rate (S to D) with the Weighted Average Approach

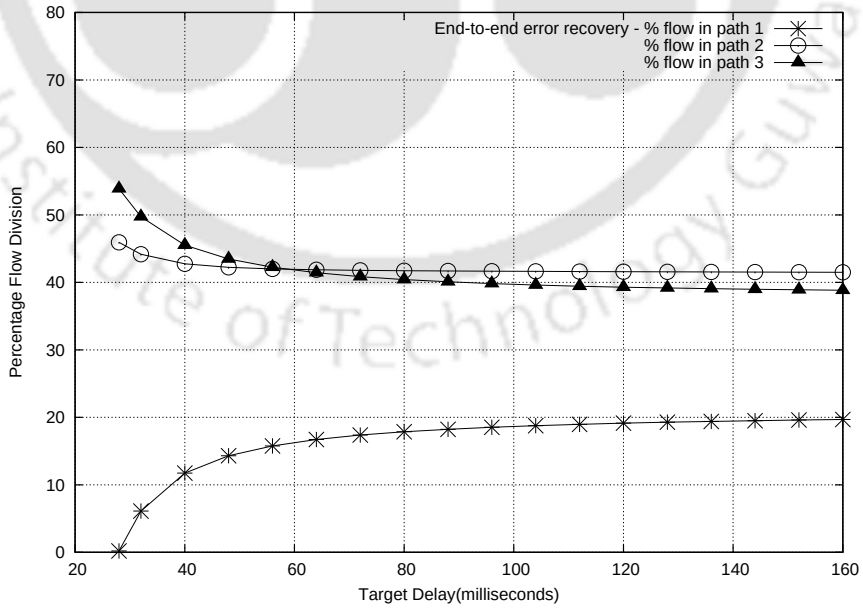


Figure 3.10: Percentage Flow Division for varying Target Delay (same for all paths) for end-to-end error recovery

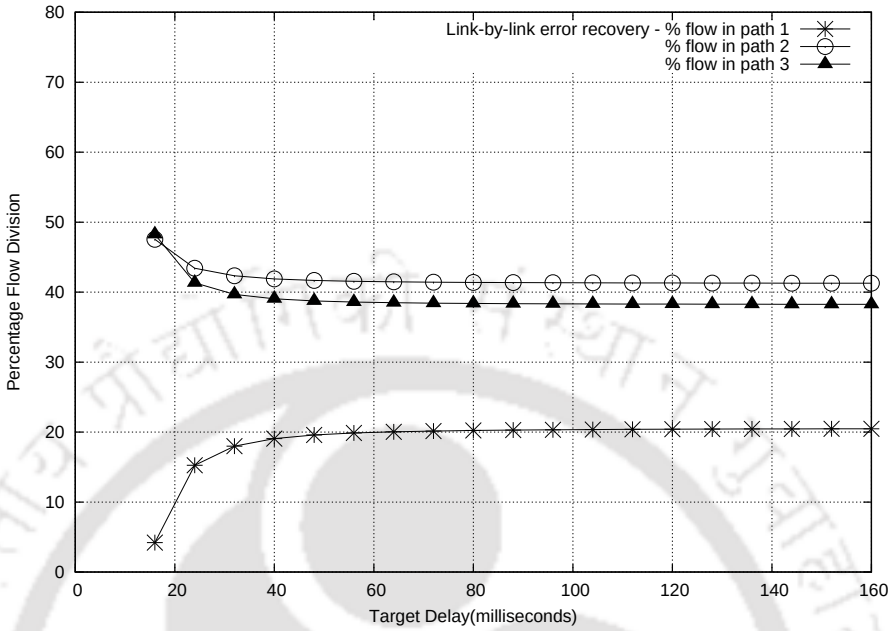


Figure 3.11: Percentage Flow Division for varying Target Delay (same for all paths) for link-by-link error recovery

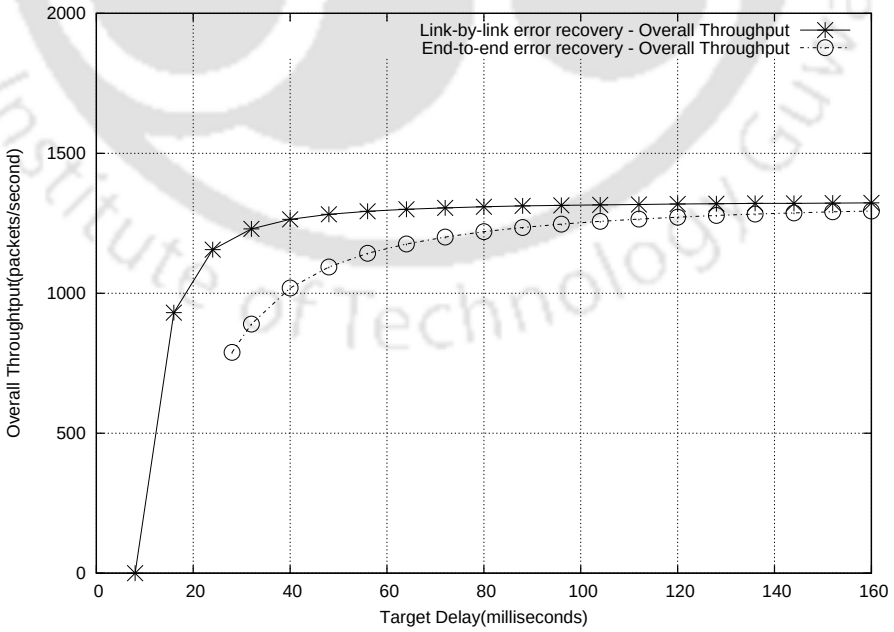


Figure 3.12: Overall Throughput (packets/second) for varying Target Delay (same for all paths)

4

Multipath Protocols and Congestion Control

Contents

This chapter introduces a modified Stream Control Transmission Protocol (SCTP) which enables it to use all the available paths concurrently. Concurrent usage of paths introduces the problem of false holes at the receiver that causes unnecessary retransmissions and increased acknowledgement traffic. Hence, we propose several modifications in the SCTP protocol with algorithms to mitigate such effects and compared the performance of our proposal with Concurrent Multipath Transfer (CMT) [17], a multipath variant of SCTP proposed earlier. This chapter also proposes a new congestion control algorithm based on Resource Pooling principle [56] and bandwidth estimation. It is suggested that this be incorporated as SCTP's original congestion control is based on the TCP congestion control approach which tends to behave unfairly on shared links when multiple paths are used concurrently.

4.1 Introduction

Transport Layer multihoming is a relatively old concept where multihomed nodes connect thorough multiple access technology and/or multiple end-to-end paths to increase their reliability and resilience to path failure [64–66]. Though SCTP was proposed with support for multihoming, it does not support simultaneous data transfer on multiple paths. For multi-homed nodes, augmenting SCTP further to handle simultaneous transfer of data on multiple paths will allow even better usage of network resources with greater reliability, especially in networks where link qualities change dynamically.

Packet reordering is a natural consequence of concurrent multipath usage. SCTP can be enabled to use multiple paths by simply modifying the sender to transmit to all the destinations simultaneously. However, if this is done, then due to packet reordering, the *Transmission Sequence Numbers* (TSN) on a path may not be in sequence and the receiver may falsely interpret them as lost TSNs. These missing TSNs are called *false holes* in the sequence. The receiver will then send the acknowledgements with missing reports for such TSNs causing unnecessary retransmissions from the sender side. The congestion window is also not updated due to these false holes, even though the packets may actually be received in the actual sequence of transmission. Typically to reduce acknowledgement traffic, a receiver would send delayed acknowledgements [51] if the packets are arriving in sequence. However, these false holes do not give the receiver an opportunity to delay acknowledgement of the received packets. This phenomenon also increases the acknowledgement

traffic in the network.

We propose Multipath SCTP (MPSCTP), another multipath variant of SCTP, to use available candidate paths in a concurrent fashion [67]‡. Our proposed approach to handle the above problems is to add another 32-bit sequence number in the chunk header to isolate packet loss from the packet reordering that may happen in a multipath transfer. Note that the introduction of this 32-bit sequence number implicitly takes care of false holes that may arise in the packet sequence due to reordering of packets on multiple paths. Since the acknowledgements are sent back based on these 32-bit sequence number values, this also obviates the problem of increased ACK traffic due to holes in the packet sequence. However, introduction of path sequence number requires changes both in the chunk header and the SACK structure. This also requires modification in certain algorithms as discussed in detail in the subsequent sections. Apart from reducing the implementation complexity, these changes are also expected to improve the system throughput with fewer overall retransmissions.

4.2 Proposed Changes in SCTP header

The smallest unit of data transfer in SCTP is called a *chunk*. Each chunk has a unique TSN to reassemble them at the destination. Our proposed MPSCTP protocol requires the following changes in the chunk header and SACK structure.

4.2.1 Changes in the Payload Chunk Header

To handle the reordering which may arise in multipath data transfer, we modify the SCTP chunk header by adding another 32-bit *Path Sequence Number* (PSN) as shown in Figure 4.1.

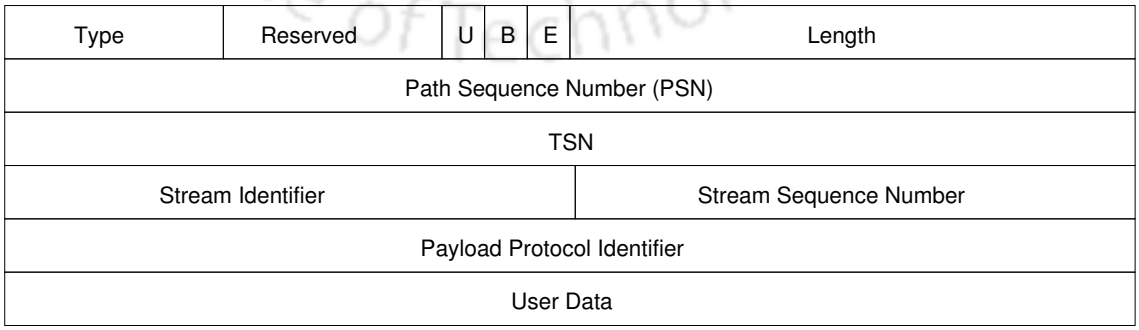


Figure 4.1: Payload Chunk Header for MPSCTP

‡Author's joint publication.

In our proposed MPSCTP, the PSN is a path specific sequence number to maintain the sequence of chunks on each candidate path. However, the TSN is still unique over all the paths and is used to maintain the sequence of data over the entire SCTP association so that it can be delivered to the upper layer of the destination in proper order. The PSN will always increase (mod-32 bit). We cannot retransmit the packet on path other the original path using the same PSN because PSNs are path dependent. Hence, a missing chunk is retransmitted using the same TSN as before but with a new PSN value to provide the flexibility of retransmitting the lost chunk on a path other than the one originally being used.

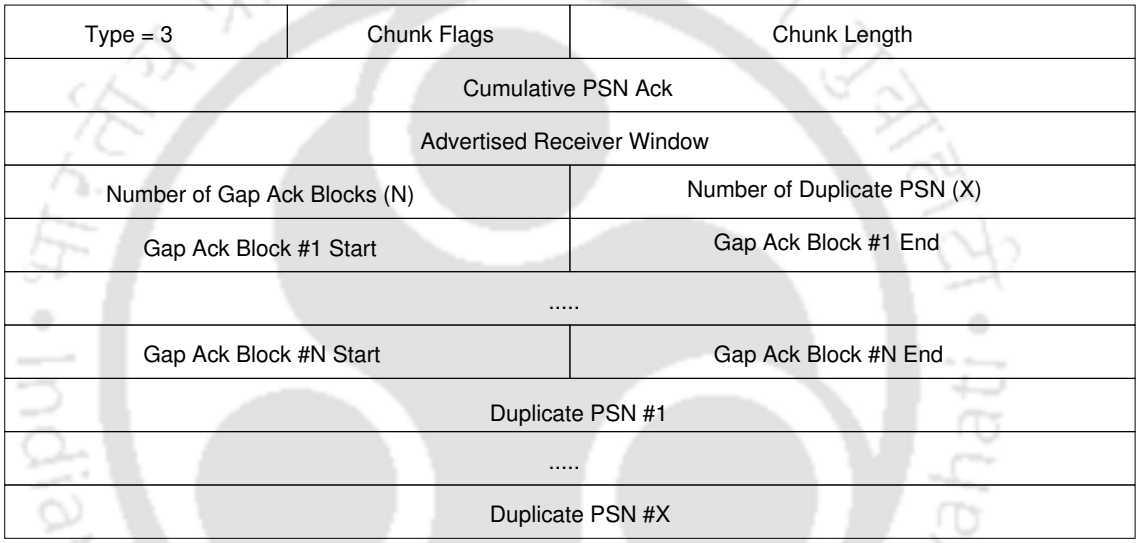


Figure 4.2: SACK Structure for MPSCTP

4.2.2 Changes in SACK Structure

In MPSCTP, each path independently acknowledges every chunk received on that path with a SACK transmitted back on the same path. This helps in proper RTT calculations for a path. In contrast with standard SCTP, here the SACK contains *Gap Ack* reports about the PSNs (as shown in Figure 4.2) for each corresponding path and the list of duplicate PSNs received on each path. The cumulative PSN Acknowledgment (*cumPSNAck*) indicates the highest PSN received per destination rather than the highest TSN received in order.

4.3 MPSCTP Algorithms

For successful data transmission and acknowledgement handling, the following algorithms of SCTP need to be modified in MPSCTP in accordance with the changes proposed in the previous section.

4.3.1 Handling Fast Retransmissions

Standard SCTP sends SACK immediately if there are gaps in the TSN and initiates *Fast Retransmission* if TSN is reported missing for four consecutive acknowledgements [6]. PSN removes the false gaps introduced by reordering due to multipath transfer. However, in case of packet loss, the packet is retransmitted with a new PSN value. This introduces holes in the packets received at the receiver. Moreover, since PSNs are path specific, the same PSN may be in use on different paths. Thus the standard SCTP fast retransmission algorithm will not faithfully indicate packet loss to initiate fast retransmission. In Figure 4.3, we present an algorithm to correctly count the missing reports for a TSN. At the sender side, this algorithm initializes a counter for each TSN. For every acknowledgement, the counter is increased for the TSN corresponding to the missing PSN. Once this count reaches 4, the corresponding chunk is retransmitted.

```
For every SACK received (at sender side):
    if (SACK contains GAP reports),
      For each destination (d_i):
      For each TSN T_a , set count C_a = 0;
      if (TSN for d_i.PSN == T_a), C_a++;
        if (C_a == 4),
          Initiate Fast Transmit;
```

Figure 4.3: Fast retransmission algorithm for gap reports in MPSCTP.

4.3.2 Cumulative Acknowledgement Update

In standard SCTP, the cumulative Acknowledgement (*cumAck*) value indicates that all the packets up to *cumAck* have been received. In MPSCTP, PSN is a path specific number. To allow retransmissions on an alternate path, the packet is required to assign a new PSN every time it is retransmitted. Hence, the *cumAck* value is replaced by the cumulative PSN Acknowledgement (*cumPSNAck*) value to indicate the highest PSN received at the destination. The algorithm to modify *cumPSNAck* value is given in Figure 4.4.

For every Chunk received (at receiver side):
For each destination (d_i):
if ($d_i.PSN_{new} > d_i.PSN_{old}$),
construct SACK with all missing PSN;
set $d_i.cumPSNAck_{new}$ = max($d_i.PSN$)

Figure 4.4: Algorithm to handle cumulative acknowledgement update.

4.3.3 Handling Missing PSNs at the receiver

Standard SCTP keeps sending the gap reports for a missing packet unless that chunk is received. However, in MPSCTP every time the packet is retransmitted the PSN is changed and the same PSN is not repeated. To handle this, we propose that the receiver sends only four gap reports (according to [6]) for a missing chunk as shown in Figure 4.5. On receiving these reports, the sender will retransmit the missing chunk with a new PSN but with the same TSN value as before.

For every chunk received (at the receiver side)
$\forall$ Missing PSN at the receiver, set count C_M = 0;
if ($PSN_{received}$!= $PSN_{missing}$),
send the gap report in the SACK;
C_M++;
if (C_M == 4),
stop sending the gap report;
set $d_i.cumPSNAck_{new}$ = max ($d_i.PSN$);

Figure 4.5: Algorithm to handle gap reports for missing PSNs.

4.4 Performance and Results

The addition of a new 32-bit field in the chunk header requires changes in the SCTP layer at the end-nodes but no changes are required in the lower layer protocols or at the intermediate nodes (if any). We tested the performance of MPSCTP through simulations using *ns-2* [68] by appropriately modifying the SCTP stack and compared its performance with CMT-SCTP. (For the CMT-SCTP protocol, we have used a patch available with the version-2.31 of *ns-2*.) In our tests, we assume that both MPSCTP and CMT use the *RTX-CWND* retransmission strategy as that is reported to be the best practically implementable strategy for CMT [17, 31]. In this strategy, retransmissions are sent to the destination with the largest congestion window. However, all other

CMT retransmission policies can be seamlessly applied to MPSCTP as well.

We consider a simple network with two disjoint direct links between dual-homed source and destination nodes. Total bandwidth of the network is assumed to be 4 Mbps. We consider two cases:

(a) Symmetric bandwidth distribution[‡], each path with 2 Mbps bandwidth and 50 ms end-to-end delay,

(b) Asymmetric bandwidth distribution, with path-1 of 1 Mbps and path-2 of 3 Mbps and 50 ms end-to-end delay for both paths.

Path-1 has a fixed packet loss probability of 1% but the probability of packet loss on path-2 is varied over 1% to 10%. We have varied the packet loss probability because packet loss has significant impact on the protocol performance and throughput. For our tests, we assume an *ftp* application transferring a 60 MB file using both MPSCTP and CMT with varying packet loss probabilities for path-2. Both CMT and MPSCTP transmits full *cwnd* data on one path before using another path. Figure 4.6 shows the average time taken for this transfer along with the 95% confidence intervals for both the symmetric and asymmetric cases using the *RTX-CWND* packet retransmission strategy. We note that in each case, MPSCTP performs better than CMT, especially when the packet loss probabilities are high. This improvement is obtained because the extra 32-bit PSN allows MPSCTP to separate packet loss from the packet reordering that may happen because of multi-path usage. Note that for both MPSCTP and CMT, the performance is better when the two paths have equal bandwidth as compared to the case where the bandwidth is unequally divided.

We have also examined the performance of our proposed protocol for different number of paths between source and the destination for different packet loss probabilities. The results in Figure 4.7 indicate that MPSCTP performs better in all the cases but that the performance degrades as the number of candidate paths between the source and the destination increases. This is due to the increased packet reordering when more paths are used. The average number of retransmission timeouts is also observed to be less for MPSCTP compared to CMT. Figure 4.8 shows that the average number of retransmission timeouts for MPSCTP is less than that of CMT for both the symmetric and asymmetric bandwidth distributions. Apart from directly improving the perfor-

[‡]Symmetric distribution implies equal distribution.

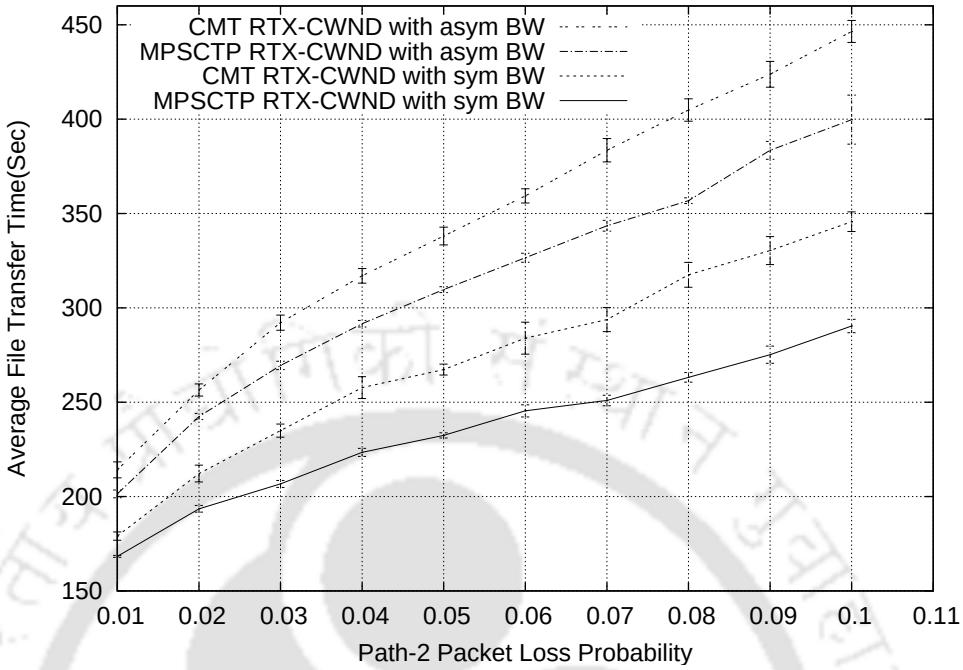


Figure 4.6: Transmission time for CMT and MPSCTP.

mance of MPSCTP, this is also indicative of the fact that MPSCTP will have lower algorithmic complexity than CMT.

We have also examined the performance of MPSCTP for multimedia data where Peak Signal to Noise Ratio (PSNR) is the QoS parameter used to measure the quality of video data transmission. We have used Evalvid-2.7 [69] for the measurement of the PSNR values. The results for the PSNR measurements for different retransmission policies [31] for both CMT and MPSCTP are summarized in Table 4.1. For these results we have considered asymmetric bandwidth distribution with 10% packet loss where any packet delayed by more than 100 ms is considered to be lost. We observe that *RTX-ASAP* [31] performs better than all other policies because it retransmits the data as soon as possible and hence fulfills the basic requirement of time sensitivity of video transmission.

Table 4.1: Average PSNR for CMT and MPSCTP for 10% packet loss

Retransmission Policies	Average PSNR	
	CMT	MPSCTP
RTX-ASAP	25.41	27.46
RTX-SAME	25.07	27.01
RTX-SSTHRESH	25.18	27.10
RTX-CWND	24.90	27.03

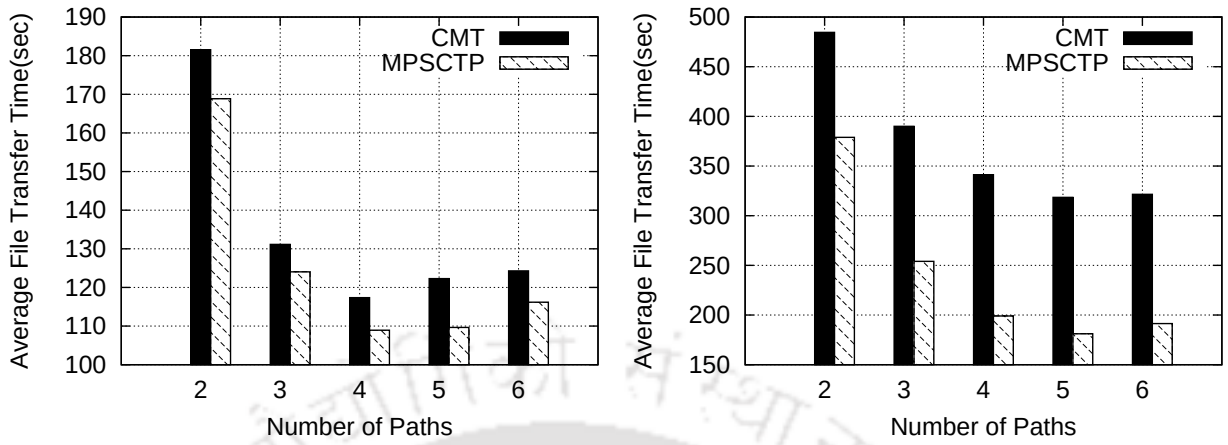


Figure 4.7: Average file transfer time for different number of paths between the source and the destination for 1% and 5% packet loss probability.

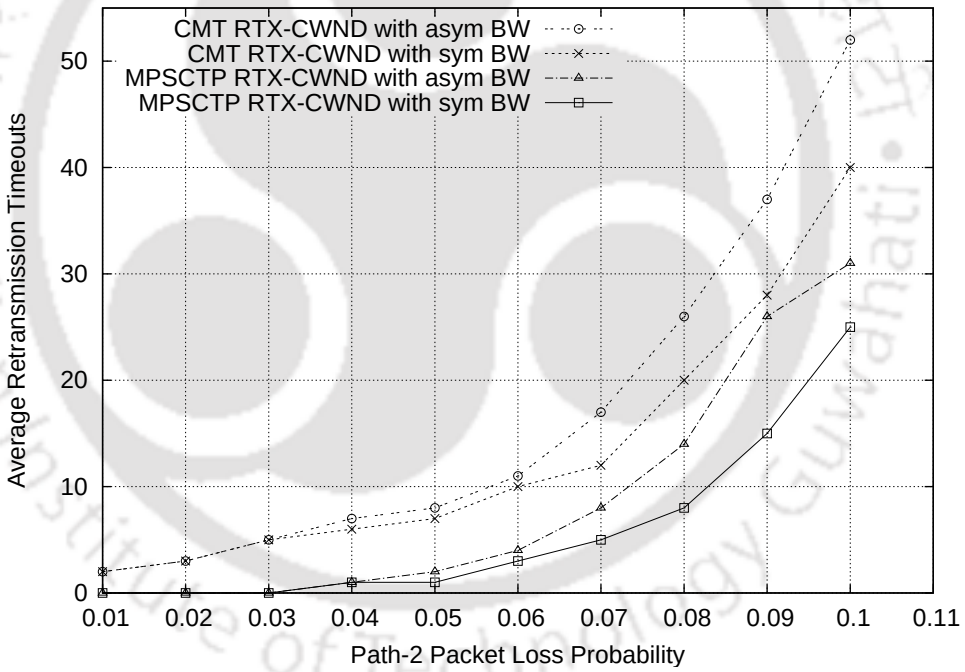


Figure 4.8: Number of retransmission timeouts for MPSCTP and CMT.

We have also measured the performance of the system for a random network of 30 nodes with various random links in between. We have run simulations for link bandwidths in the range of 1-2 Mbps with link delays of 50-90 ms for these links. This network consists of a dual homed source and destination. There are various other TCP sources contributing traffic in the network. The trends in the performance results were along the same lines as those for the direct paths illustrated

earlier with MPSCTP showing a 10-15% improvement over CMT.

4.5 Bandwidth Estimation Based Resource Pooling (BERP) Congestion Control

Considerable research work is currently underway to support multipath transport in networks so that resources can be efficiently utilized for data transfers. Implementation of proper Congestion Control algorithms is necessary for the operation of any transport protocol over a network and must be incorporated in any Internet based implementation. The congestion control mechanism proposed for multipath protocols like SCTP and CMT is largely based on the mechanism followed by traditional TCP and is not effective when multiple paths are concurrently used. While proposing and implementing new congestion control mechanisms for multipath protocols, a primary requirement would be that the new protocol implementation should not take undue advantage of the network and should not harm other fellow TCP and UDP flows with which it coexists in the network.

For providing better QoS to an end user, load balancing has become a de-facto implementation strategy for any service provider. The basic requirement for load balancing is to aggregate all the available resources. Thus load balancing is an example of *resource pooling*. The basic philosophy of resource pooling is to look at all the available resources as a single pool of resources. The available bandwidth of the individual paths is a typical example of the resource under consideration. Wischik et al. introduced the concept of resource pooling in [56]. According to the resource pooling principle, the total bandwidth of the set of available paths will be shared by all the flows using those paths. Currently IETF is working on MultiPath TCP, a TCP extension to support multiple path usage in parallel fashion. In [37], Raiciu et al. have proposed a coupled congestion control algorithm based upon this resource pooling principle. They have demonstrated that their algorithm exhibits better performance in terms of throughput and fairness on candidate paths and shared links, respectively. This algorithm tries to achieve the following three goals:

i) *Improved throughput* i.e. multipath flow should perform at least as good as the best of the path available in the pool,

ii) *Do not harm* i.e. other fellow flows on the shared link should not unduly suffer,

iii) *Balance Congestion* i.e. the congestion experienced by all the paths should be similar.

Both CMT-SCTP and Multipath TCP (MPTCP) have been proposed to use multiple parallel paths for data transfer. This should provide better resource utilization and increase the error resilience of the underlying network. The congestion control of SCTP is based on the TCP congestion control. TCP based congestion control works well with standard SCTP because in spite of being a multihomed protocol, it uses only one path at a time and switches to an alternate path only if the primary path degrades or fails. However, CMT-SCTP uses all the paths concurrently and each flow of CMT operates independent of each other. Hence it takes undue advantage of shared links on multiple paths which will create issue of fairness and TCP friendliness. In literature, it has been demonstrated that CMT flows unduly harm other network flows on the shared links and captures more bandwidth in the ratio of the number of concurrent paths being established across any shared bottleneck link en-route [70]. These problems motivated us to reexamine the congestion control algorithm of multipath protocols like CMT and MPSCTP and suggest an approach which will allow multiple paths to be used without incurring these difficulties.

A coupled congestion control algorithm which is based upon the resource pooling principle would be able to handle the shared link problem in a multipath scenario. During the congestion avoidance phase, it increases the congestion window in the inverse ratio of the total congestion window of all the candidate paths on every acknowledgement. However, during the congestion detection phase, coupled congestion control decreases the congestion window to half. Unlike wired links, losses in wireless links may happen because of reasons other than congestion (e.g. noise or link quality). Therefore, arbitrary reduction of the congestion window to half is not an optimum approach on heterogeneous networks with wireless links. Hence, we consider the resources to be a single pool of resources during the congestion detection phase as well. We have considered SCTP as the example protocol for this congestion control. However, the same can be adapted for any other multipath protocols as well. This algorithm applies the resource pooling principle based upon the bandwidth estimates obtained by observing the data flow on the paths. We have proposed a modified congestion control algorithm using **B**andwidth **E**stimation based **R**esource **P**ooling (**BERP**) to overcome the above mentioned shortcomings of traditional congestion control when applied to multipath protocols.

4.5.1 Proposed Changes in Congestion Control

Currently most of the congestion control algorithms are based upon the standard TCP congestion control algorithm with some minor modifications. SCTP congestion control is also based on TCP congestion control algorithm. Like TCP, SCTP congestion control algorithm also consists of three phases, i) Slow Start phase, ii) Congestion Avoidance phase and iii) Congestion detection phase. CMT and MPSCTP also inherit the same congestion control from SCTP e.g. CMT starts the connection with the slow-start phase. It exhibits exponential growth in the congestion window ($cwnd$) till the slow start threshold ($ssthresh$) is reached. Once the $cwnd$ value crosses $ssthresh$, it enters into the congestion avoidance phase. During congestion avoidance phase, CMT increases its $cwnd$ by one Maximum Transmission Unit (MTU) for every RTT. CMT decreases its $cwnd$ to half during congestion detection phase.

Whenever the sender experiences any loss in packet, traditional SCTP algorithm infers that the path is in congestion and to reduce the congestion on that path it reduces the congestion window by half. However, in wireless links this may not always be desirable as a wireless link may experience higher packet loss even when the link is not in congestion. We have avoided this arbitrary reduction of the congestion window in our approach by using bandwidth estimation based resource pooling during congestion detection for adjusting the congestion window. To apply resource pooling, we need to measure the available resources, i.e. bandwidth available to the flow. We use bandwidth estimation for measuring the link bandwidth available to the user. Several techniques for bandwidth estimation have been proposed in literature. We use the same approach for the bandwidth estimation as the one being used in TCP Westwood [71]. Using these, the congestion avoidance and congestion detection algorithms are modified as given next.

Congestion Avoidance Phase: Increase the value of $cwnd$ of i^{th} path by:

$$min(\alpha * P_a * MTU/c_T, P_a * MTU/c_i) \tag{4.1}$$

$$P_a = P_a - c_i \tag{4.2}$$

where,

P_a = Partial Bytes Acked

c_T = Total congestion window

c_i = Congestion window of i^{th} path

$$\alpha = \frac{2 * c_T * \max_i \left(\frac{\beta_i * c_i}{RTT_i^2}\right)}{\left(\sum_i \frac{c_i}{RTT_i}\right)^2} \tag{4.3}$$

$$\beta_i = \frac{BWE_i}{\sum_i BWE_i} \tag{4.4}$$

where,

BWE_i= Bandwidth estimate of i^{th} path

Note that MPTCP's coupled congestion control is also a special case of our algorithm with $\beta = 0.5$.

Congestion Detection Phase: *if (four duplicates are received)*

$$s_i = max(c_i - \beta_i * c_i, 4 * MTU) \tag{4.5}$$

$$c_i = s_i \tag{4.6}$$

if (Timeout Occurred)

$$s_i = max(c_i - \beta_i * c_i, 4 * MTU) \tag{4.7}$$

$$c_i = 1 * MTU \tag{4.8}$$

where,

s_i = slow start threshold of i^{th} path,

4.5.2 Calculation of α

The first part in (4.1) tells us how the congestion window should be increased in the congestion avoidance phase. However, the second part ensures that this increase is not more than the corresponding TCP increase on that path. The factor 'α' in the first term decides by how much the

congestion window should be increased and is called the *factor of aggressiveness*. The value of 'α' is chosen such that at equilibrium, the increase and decrease of congestion window are equal.

$$(1-p_r)\cdot \min(\frac{\alpha}{c_T},\frac{1}{c_i}) = p_r\cdot\beta_i\cdot c_i$$

$$\min(\frac{\alpha}{c_T},\frac{1}{c_i}) = \frac{p_r}{(1-p_r)}\cdot\beta_i\cdot c_i \tag{4.9}$$

If c_{TCP} is the equilibrium congestion window for TCP, then we can similarly write

$$(1-p_r)\cdot\frac{1}{c_{TCP}} = p_r.\frac{c_{TCP}}{2}$$

$$\Rightarrow c_{TCP} = \sqrt{\frac{2(1-p_r)}{p_r}} \tag{4.10}$$

From (4.9) and (4.10), for our algorithm to perform at least as good as TCP, we need:

$$c_{TCP}^2 = \frac{2\cdot\beta_i\cdot c_i}{\min(\frac{\alpha}{c_T},\frac{1}{c_i})}$$

Therefore,

$$c_{TCP}^2 = 2\cdot\beta_i\cdot c_i\max(\frac{c_T}{\alpha},c_i) \tag{4.11}$$

From the Goal (1), the throughput of the system should be at least as good as of the best TCP throughput on any of the candidate paths.

$$\sum_i \frac{c_i}{RTT_i} \geq \max_i\left(\frac{c_{TCP}}{RTT_i}\right) \tag{4.12}$$

From (4.11) and (4.12), we get

$$\left(\sum_i \frac{c_i}{RTT_i}\right)^2 = 2\max_i\left(\frac{\beta_i c_i\max(\frac{c_T}{\alpha},c_i)}{RTT_i^2}\right)$$

$$\left(\sum_i \frac{c_i}{RTT_i}\right)^2 \triangleq 2\max_i\left(\frac{\beta_i c_i c_T}{\alpha RTT_i^2}\right)$$

$$\alpha = \frac{2 * c_T * \max_i \left(\frac{\beta_i * c_i}{RTT_i^2}\right)}{\left(\sum_i \frac{c_i}{RTT_i}\right)^2}$$

4.5.3 Calculation of Bandwidth Estimate

Since multipath protocols use all the paths concurrently, we can use the SACKs on each path for measuring the bandwidth estimate on respective paths. If the k^{th} SACK is received at time instant t_k and acknowledges d_k new bytes then the current bandwidth sample B_k is given by

$$B_k = \frac{d_k}{(t_k - t_{k-1})}$$

In order to average the sampled bandwidth, a low pass filter is applied to these samples as in [71]. The filtered bandwidth estimate is given as

$$\hat{B}_k = p \cdot \hat{B}_{k-1} + (1-p) \cdot \left(\frac{B_k + B_{k-1}}{2}\right)$$

where p is a constant whose value is typically set to 0.9.

4.6 Performance Analysis of BERP congestion control

In this section we present the performance of BERP congestion control algorithm and compare it with TCP like congestion control. We have used CMT as the test protocol for this comparison.

4.6.1 Disjoint Paths with no congestion

We consider the network graph as shown in Figure 4.9. This network consists of dual homed source (S) and destination (D). Both the paths have several disjoints links as shown in the figure. The link bandwidths and the corresponding delays are as shown in figure. All the links in the networks have packet loss probability of 1% except the packet loss probability for the link between N_2 and N_4 is varied over 1-10%.

For our tests, we have assumed that the source is transferring a 60 MB file to the destination using ftp. In Figure 4.10, we show the time taken to transfer the file from the source to the destination when the packet loss probability of link is varied from 1% to 10% for both the CMT congestion control (CMT-CC) algorithm and our proposed algorithm (CMT-BERP). The results

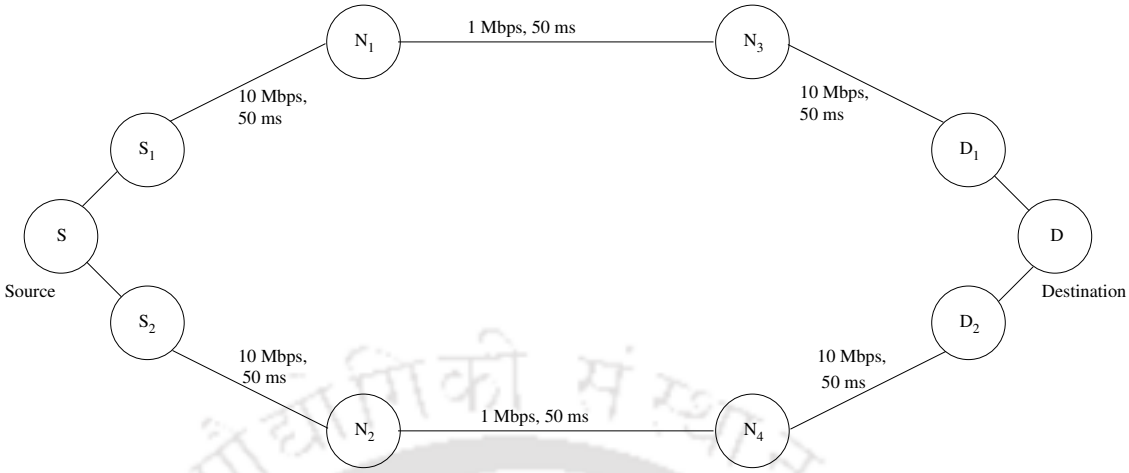


Figure 4.9: Network Graph for disjoint paths with similar characteristics.

indicate that when the losses are purely due to error and there is no congestion in the links, CMT-BERP algorithm performs better in comparison to the CMT-CC algorithm. This is because CMT-BERP algorithm does not abruptly decrease the congestion window on every loss by assuming that this is being caused by congestion. Instead, it estimates the bandwidth of the connection and decreases the bandwidth according to the resource pooling principle. In Figure 4.11, we show how the congestion window at the source evolves for the case when the packet loss probability is 10%. Note that there are couple of instances between **18 seconds** and **38 seconds** when the *cwnd* for CMT-BERP falls below the CMT-CC. We observed from the simulation traces that during this time CMT-BERP has experienced much higher losses in comparison to CMT-CC. However, we can note that mostly *cwnd* for CMT-BERP is much larger than the CMT-CC and hence the average transmission time for CMT-BERP is much less than that for CMT-CC.

4.6.2 Paths with Shared Bottleneck link

Figure 4.12 shows an example of a scenario with a shared bottleneck link. The link characteristics are same as in Section 4.6 except that the shared bottleneck link between N_1 and N_2 has 1 Mpbs bandwidth with a packet loss probability that is varied from 1% to 10%.

The test results for transferring a 60 MB file are shown in Figure 4.13. Figure 4.13 shows the average file transmission time for varying packet loss probability in the bottleneck link.

The results indicate that CMT-BERP algorithm performs better than the CMT-CC algorithm for this scenario. However, because of the shared bottleneck link, the average transmission time is

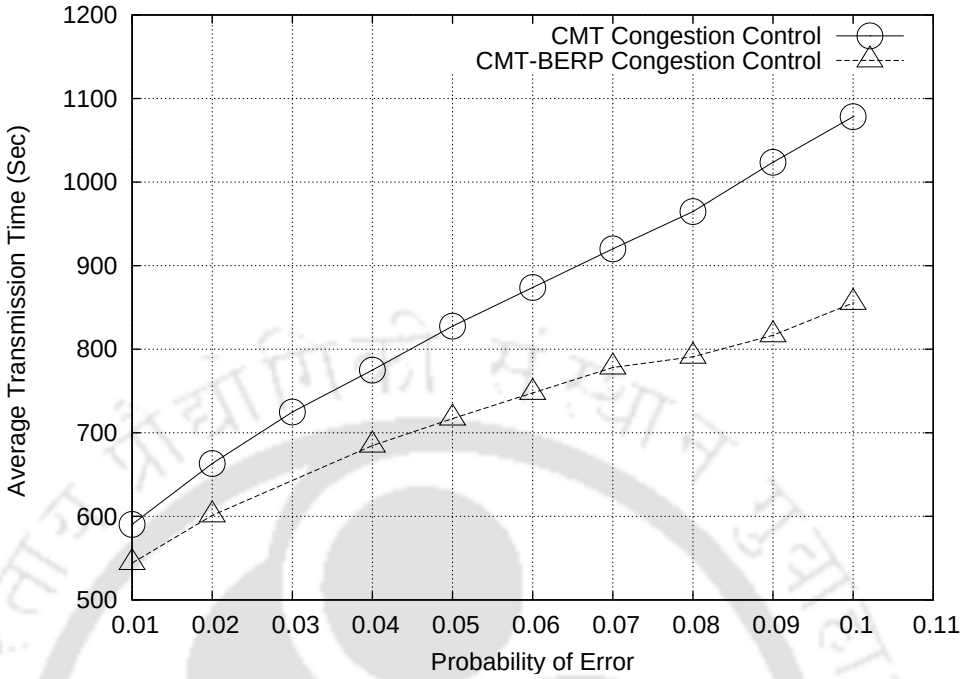


Figure 4.10: Average Transmission Time for disjoint paths.

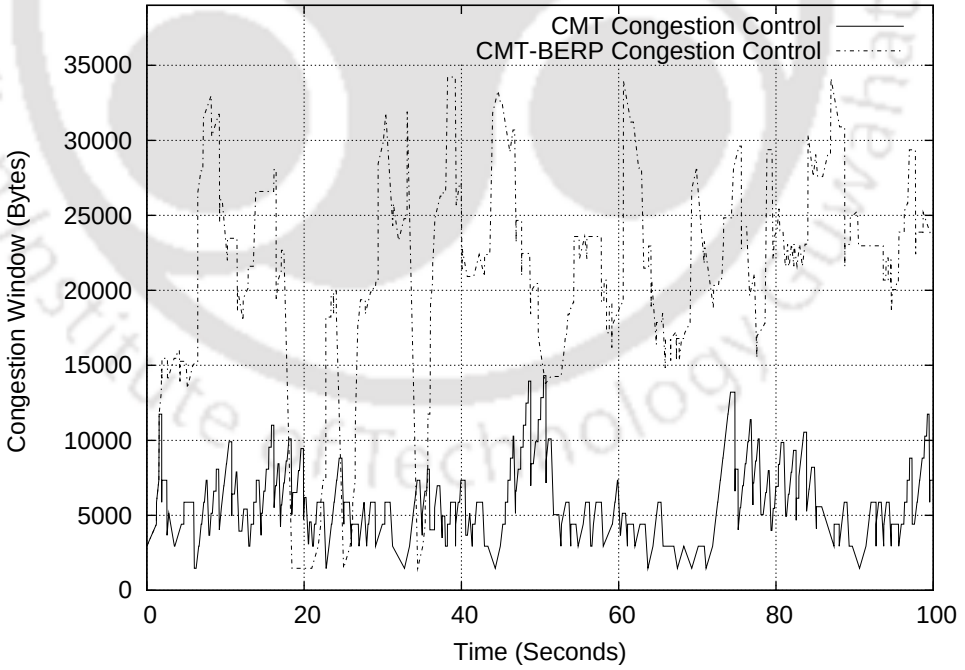


Figure 4.11: Congestion Window evolution for CMT congestion control and CMT-BERP algorithm for packet loss probability of 10% for disjoint paths.

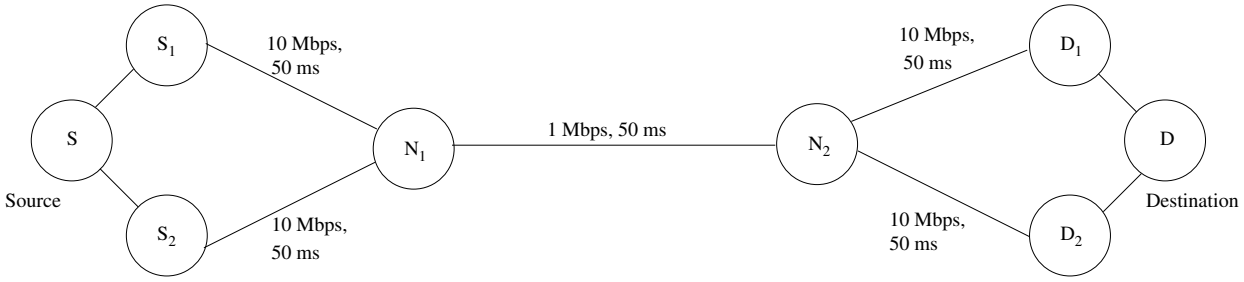


Figure 4.12: Network Graph for shared bottleneck.

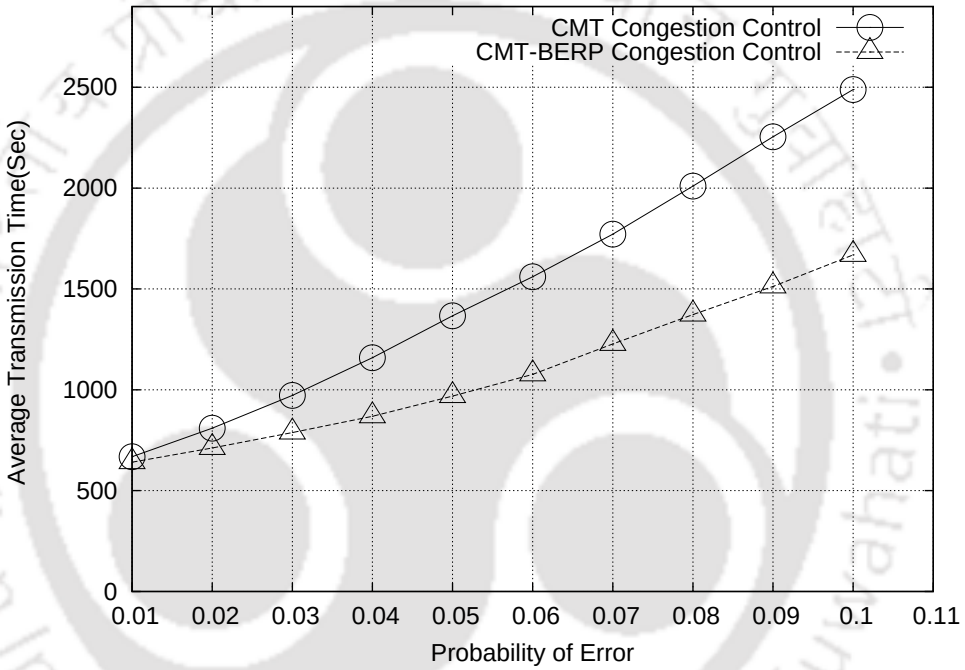


Figure 4.13: Average Transmission Time for Shared bottleneck link.

higher than the earlier case where disjoint paths were used. The way the congestion window evolves in this scenario has been shown in Figure 4.14 for both the congestion control schemes when the bottleneck link has 10% packet loss probability.

4.6.3 Disjoint paths with only congestion losses

For this scenario, as in Figure 4.15, we consider all the links to be error free. We assume that U_1 and U_2 are UDP sources generating CBR traffic with U_{11} and U_{12} as their respective sink nodes. These UDP sources are introduced to create congestion in the network between the source (S) and destination (D).

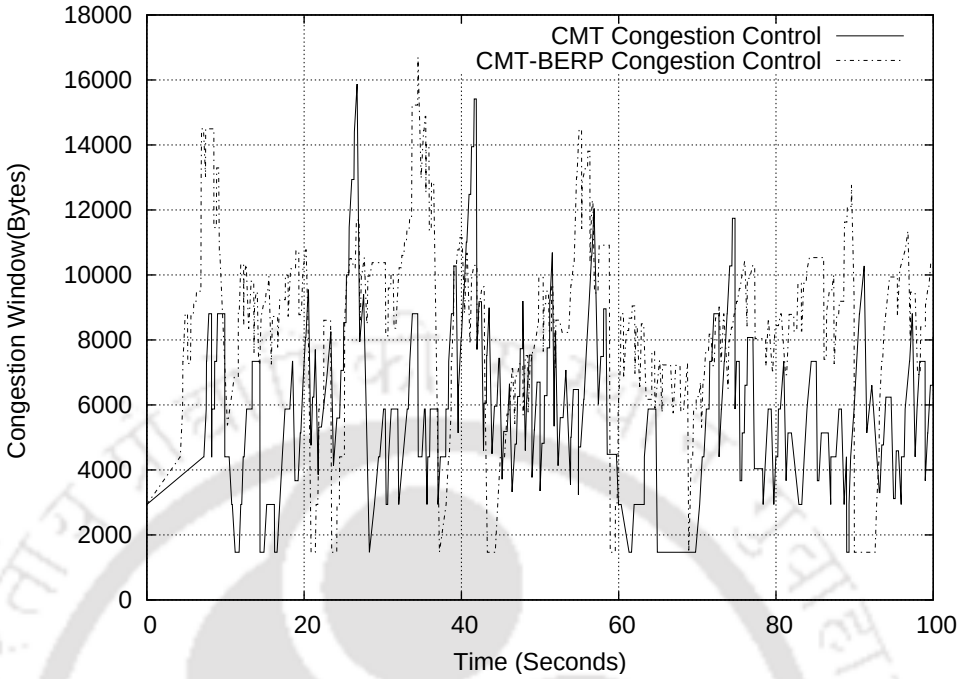


Figure 4.14: Congestion Window evolution for CMT congesion control and CMT-BERP algorithm for packet loss probability of 10% for shared bottleneck link.

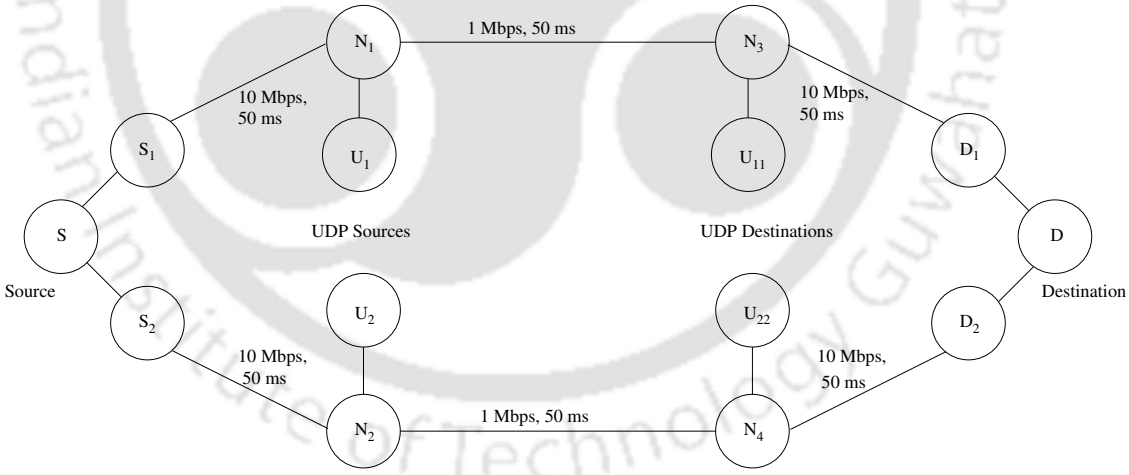


Figure 4.15: Network Graph for disjoint links with congestion losses.

The packet size assumed for these UDP sources is 512 bytes. In this case, we have obtained average throughput achieved for both CMT-CC algorithm and CMT-BERP algorithm. The results are shown in Figure 4.16.

We observe that in this scenario, both congestion control strategies show almost identical performances. This is expected, as in this system, virtually all the losses are caused by congestion.

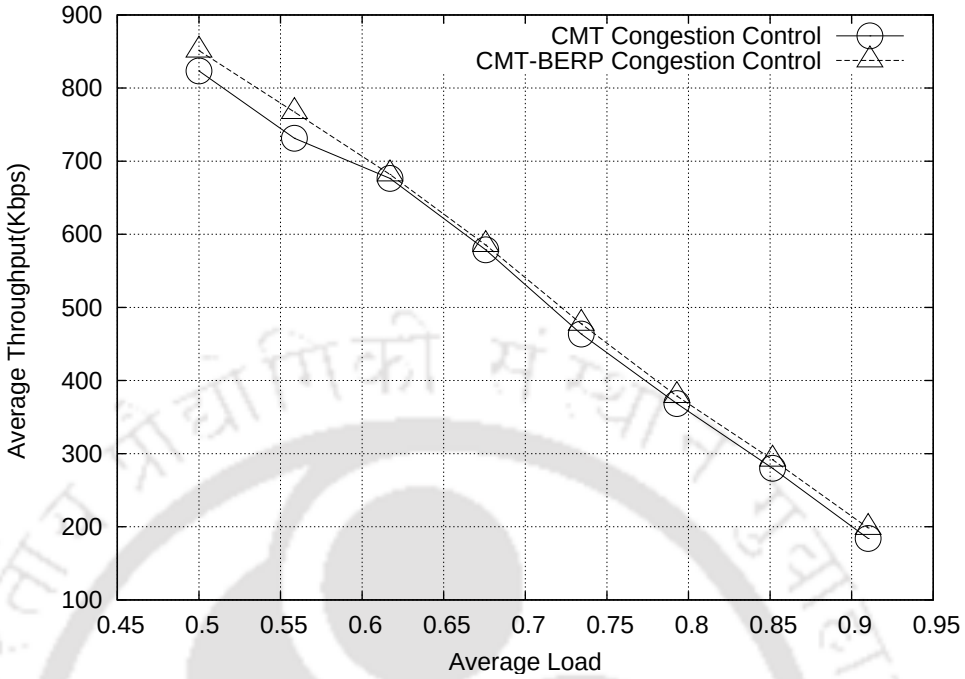


Figure 4.16: Average Throughput for different Average load of UDP traffic.

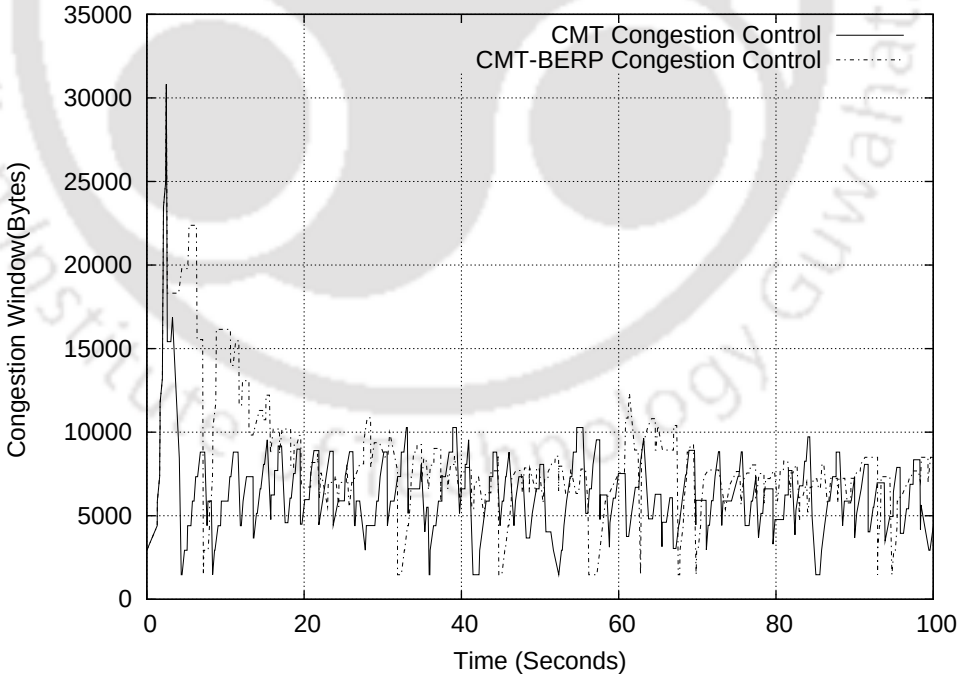


Figure 4.17: Congestion Window evolution for average UDP traffic load of 0.9.

The evolution of the congestion window for this scenario has also been shown in Figure 4.17 for average UDP traffic load of 0.9.

4.6.4 Disjoint paths with both congested and erroneous links

We have also considered a scenario as shown in Figure 4.18. As in previous scenario, U_1 and U_2 are UDP sources generating On-Off CBR traffic with U_{11} and U_{12} as their respective sink nodes. These UDP sources create congestion in the network between the source (S) and destination (D). The packet size of CBR traffic is assumed to be 512 bytes. Unlike the previous scenario, the links in this network also have losses due to errors.

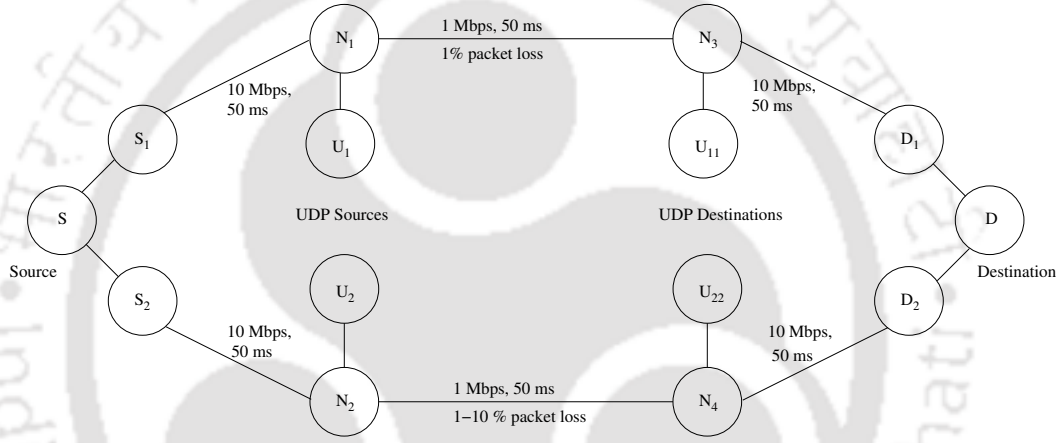


Figure 4.18: Network Graph for both congested and erroneous links.

All the links in the network have packet loss probability of 1% except the packet loss probability for the link between N_2 and N_4 is varied over 1-10%. Here, we have assumed that the source is transferring a 60 MB file to the destination using *ftp*.

Figure 4.19 shows the time taken to transfer the file from the source to the destination when the packet loss probability of link is varied from 1% to 10% for both CMT-CC and CMT-BERP algorithms. From Figure 4.19, it is clear when the probability of error is less, the network has predominantly congestion losses. Hence, the two algorithm have similar performances. However, at higher probabilities of error the CMT-BERP adapts better to the transmission losses. That is why the average time taken by CMT-BERP to trasfer the file is significantly less than CMT-CC algorithm. We have also plotted the congestion window evolution for the two algorithms in Figure 4.20. This also indicates that at higher probability of error CMT-BERP performs better than CMT-CC algorithm.

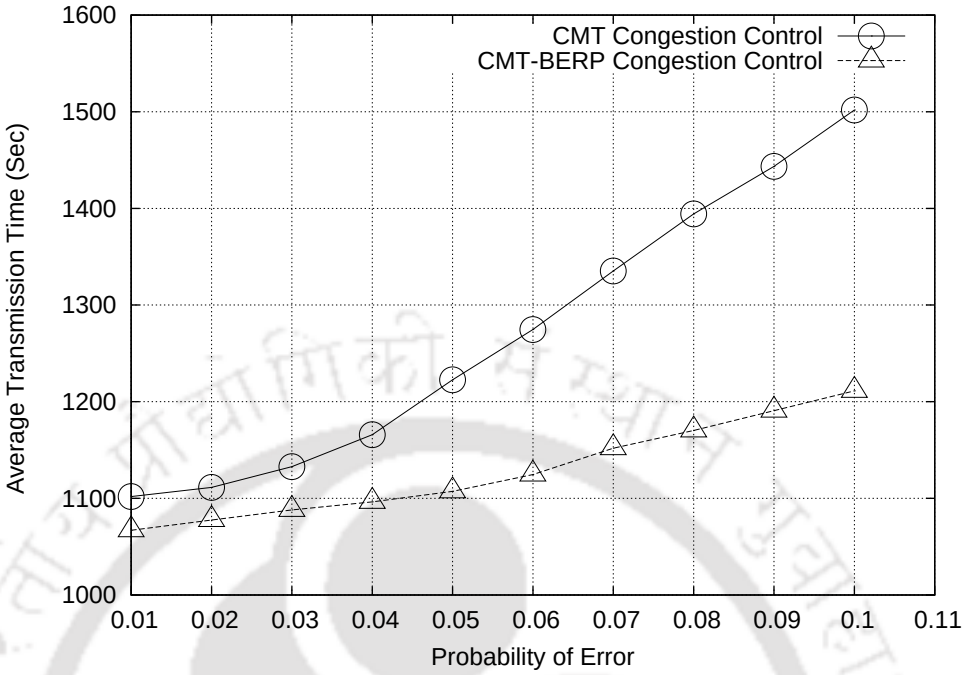


Figure 4.19: Average Transmission Time for both congested and erroneous links.

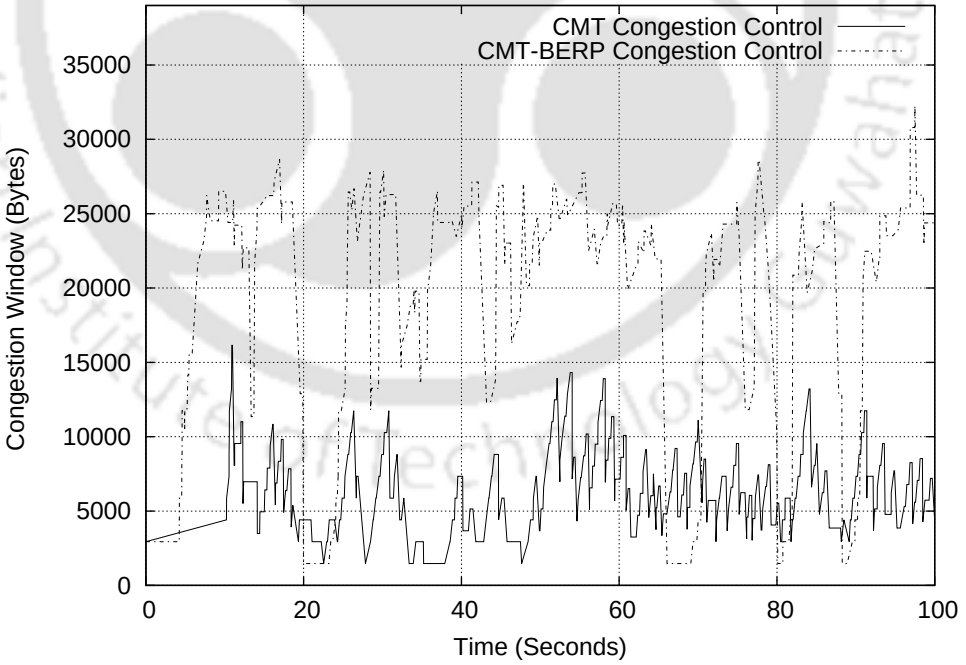


Figure 4.20: Congestion Window evolution for CMT congesion control and CMT-BERP algorithm for packet loss probability of 10% for both congested and erroneous links.

4.7 MPSCTP Fast Retransmission Algorithm

The algorithm in Section 4.3.1 implicitly assumes that there are no losses in acknowledgements. This is a relatively strong assumption for lossy and wireless networks and it would be important to study an implementation where this assumption is not made. For this, we have proposed a modified algorithm MPSCTP to handle Fast Retransmissions for mitigating the above limitation [72][‡].

4.7.1 Handling Fast Retransmission modified

In the original Fast Retransmission algorithm, the chunks are retransmitted with a modified PSN value. This gives the flexibility to retransmit the chunk on a path other than the original transmission. The receiver stops waiting for the corresponding PSN after transmitting four missing reports in SACK for the chunks are retransmitted with modified PSN value. If such a SACK is lost in the middle and another SACK with an updated *cumPSNAck* value is received, this might confuse the sender and the corresponding chunk may not be retransmitted. To mitigate this limitation, we have proposed a modification in the SACK structure as shown in Figure 4.21. The algorithm to handle Fast Retransmissions is also modified accordingly as shown in Figure 4.22.

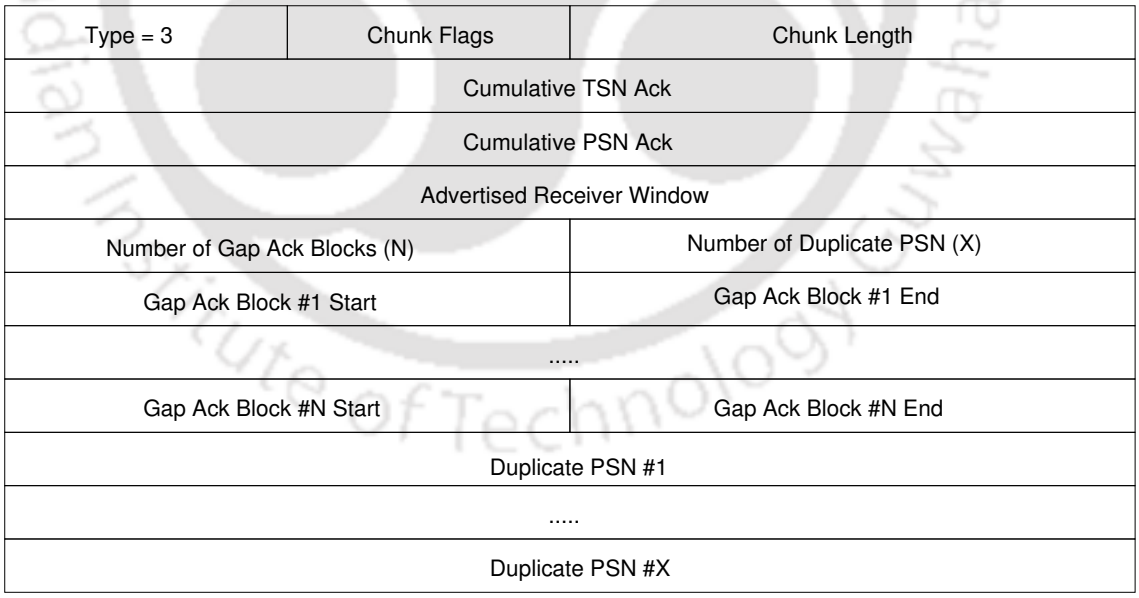


Figure 4.21: Modified SACK Structure for MPSCTP.

As shown in Figure4.21, the SACK contains the Cumulative TSN Ack (*CumTSNAck*). This number indicates the chunks received in sequence at the receiver across the entire association.

[‡]Author's joint publication.

For every SACK received (at sender side):
if (SACK contains GAP reports),
For each destination (d_i):
For each TSN T_a , set count $C_a = 0$;
if (TSN for d_i.PSN == T_a), C_a++;
if ($C_a \geq 3$),
Initiate Fast Transmit;

if (CumTSNAck $\geq$ TSN in send buffer),
Do not remove from SendBuffer;
For each destination (d_i):
if(timeout for d_i.PSN)
Initiate Fast Transmit;

Figure 4.22: Modified Fast retransmission algorithm for gap reports in MPSCTP.

However, the Cumulative PSN Ack (*CumPSNAck*) indicates the number of chunks received in order on that particular path. The receiver expects the lost TSN to be received with a new PSN value. The receiver will not update the cumulative TSN acknowledgement (*cumTSNAck*) number unless it receives the desired TSN. The sender will also not remove the corresponding chunk from its send buffer unless the cumTSNAck is updated and will retransmit the missing chunk if timeout occurs for that chunk.

4.7.2 Performance Analysis

In this section we study the performance of MPSCTP with the modified fast retransmission algorithm. We have compared the results for this with the CMT protocol. We consider dual homed source and destination nodes with two paths between them which are disjoint. The bandwidths of both the paths are assumed to be 2 Mbps and the propagation delays for the two paths are 50 ms. Path-1 has a fixed packet loss probability of 1% but the probability of packet loss on path-2 is varied over 1% to 10%. However, both paths have been assumed to have a probability of acknowledgement loss of 1%. For our tests, we assume an ftp application transferring a 60 MB file using both MPSCTP and CMT with varying packet loss probabilities for path-2.

Figure 4.23 shows the Average transmission time for the file using CMT and MPSCTP. We have also compared the results for two different retransmission strategies. We have considered *RTX-SAME* and *RTX-CWND* strategies for our comparison. We notice that MPSCTP with *RTX-CWND* strategy performs significantly better than CMT with *RTX-CWND*. However, MPSCTP

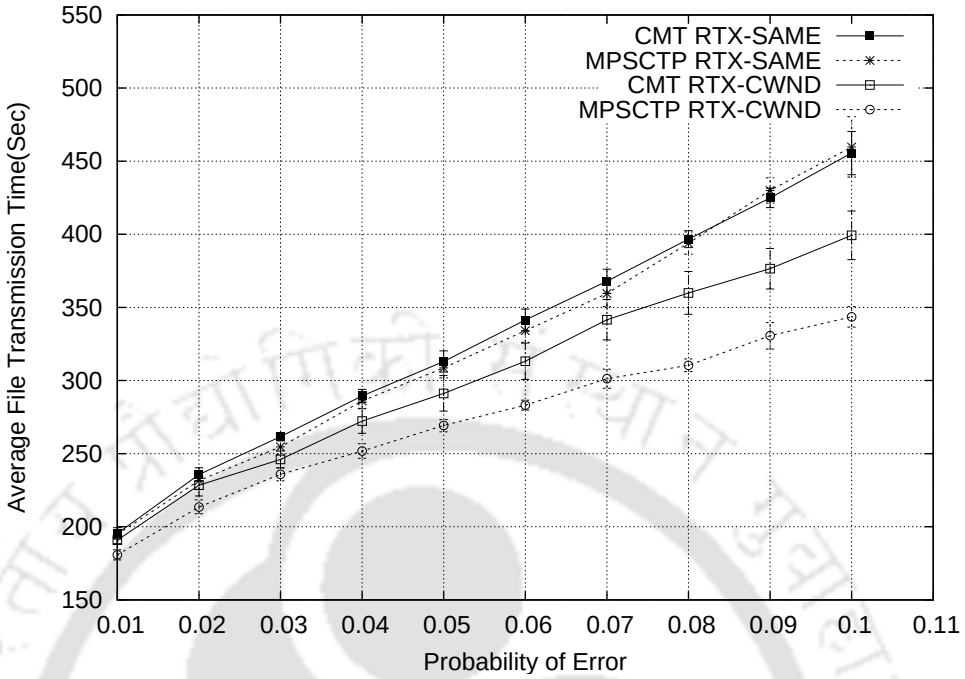


Figure 4.23: Average File Transmission Time for CMT and MPSCTP with RTX-SAME and RTX-CWND retransmission strategies.

with *RTX-SAME* strategy performs almost similar to CMT with *RTX-SAME* strategy. It can also be noted that the performance improvement of MPSCTP over CMT is less than that of in Section 4.4. The reason for this behaviour is MPSCTP is much more senstive to acknowledgement losses than CMT.

4.8 Comparative Study of CMT, MPSCTP and MPTCP

In this section we present comparative study of CMT, MPSCTP (with BERP congestion control) and MPTCP. For MPTCP we have used the patch distributed as part of the Google Code Project [73].

4.8.1 Performance Analysis

The comparative performance of the CMT, MPSCTP and MPTCP protocols has been considered for the following network scenarios.

4.8.1.1 Network with Disjoint Paths

Here, we have considered two network paths with no congestion as shown in Figure 4.24 where both the source and the destination are dual-homed in nature. There are several nodes between the source and the destination on both paths. The bit rates of each of the links and the link delays are as shown in Figure 4.24. On *path-1*, the probability of packet error is assumed to be 1% on the Source-N_1 and N_3-Destination links but is varied over 1-10% on the $N_1 - N_3$ link. All the links in *path-2* are assumed to have packet error probabilities of 1%.

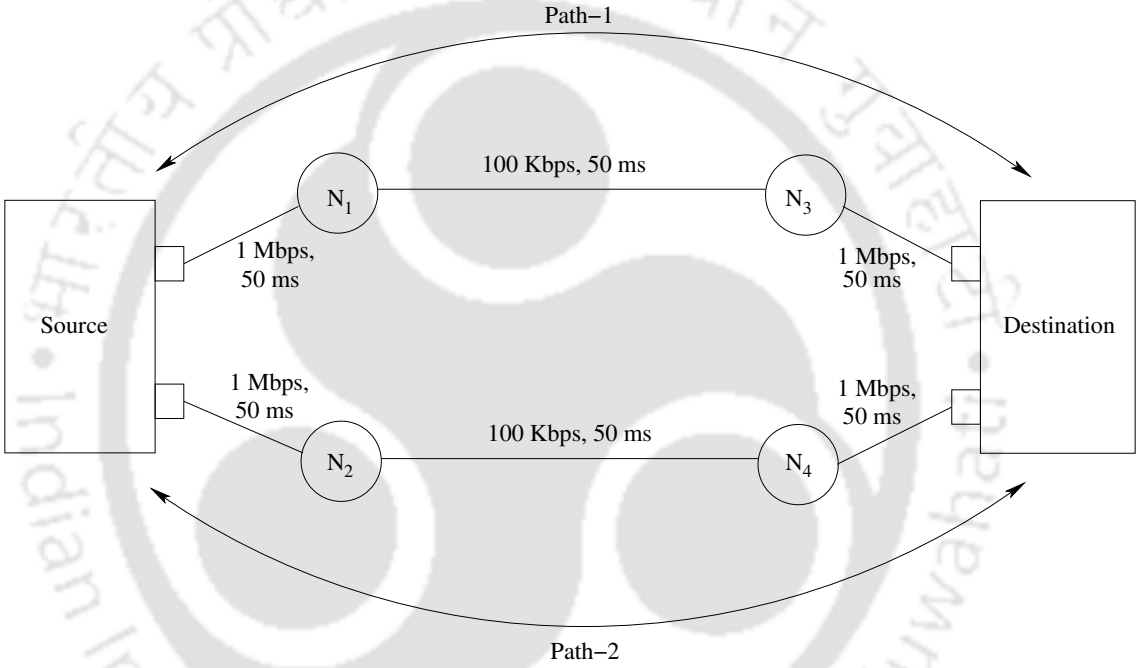


Figure 4.24: Network topology for disjoint paths.

For these tests we assume that the source transmits a 16 MB data file to the destination and measure the time taken for this transfer while varying the packet error probability of the $N_1 - N_3$ link from 1% to 10%. The packet size assumed for the data transfer is 536 bytes. 536 bytes is the default MSS for the TCP [74]. The MTU of the paths is set to be 568 bytes so that every packet for CMT & MPSCTP will consist of only one chunk. We have compared the time taken to transfer this file for CMT, MPSCTP and MPTCP in Figure 4.25.

From the results in Figure 4.25, it is clear that MPTCP takes the maximum time to transfer the file. The results for CMT and MPSCTP are quite close though MPSCTP does show a somewhat better performance for high packet error probabilities (i.e. beyond 4%). This is primarily due to

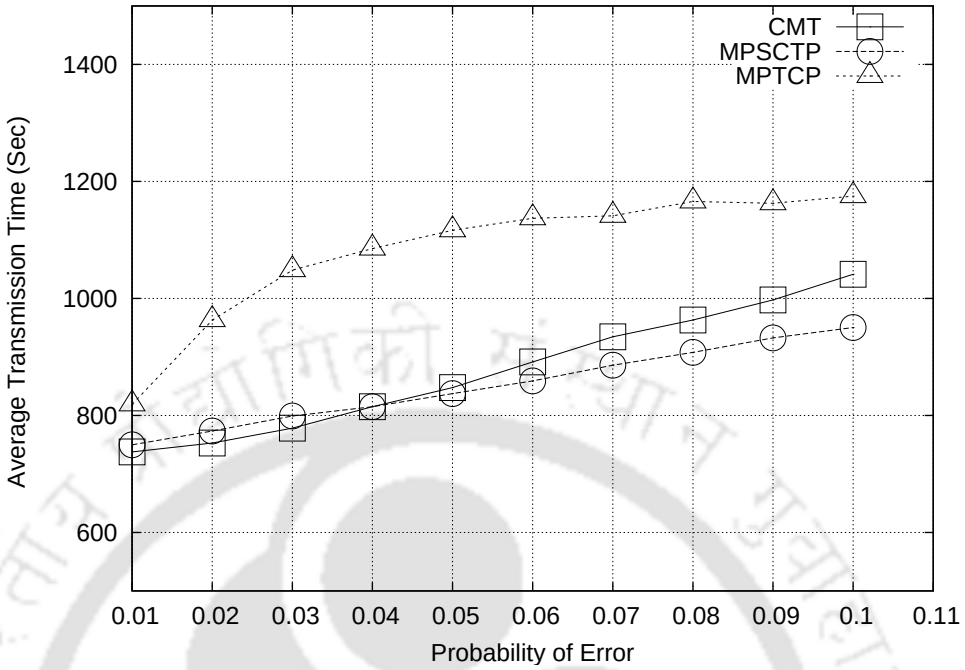


Figure 4.25: Average packet transmission time for disjoint paths.

the fact that there is no congestion loss in the network and the errors observed are mainly due to packet loss events. The BERP algorithm adapts better to this situation and reduces the congestion window in the ratio of the bandwidth available. (CMT or MPTCP would abruptly reduce the congestion window to half assuming every loss of a packet as a congestion loss.)

4.8.1.2 Paths with Shared Link

For this, we consider the network of Figure 4.26 with the bit rates and delays for each link are as shown in the figure. All the links other than $N_1 - N_2$ have packet error probabilities of 1%.The packet error probability for the $N_1 - N_2$ link is varied between 1% and 10% for our simulations.

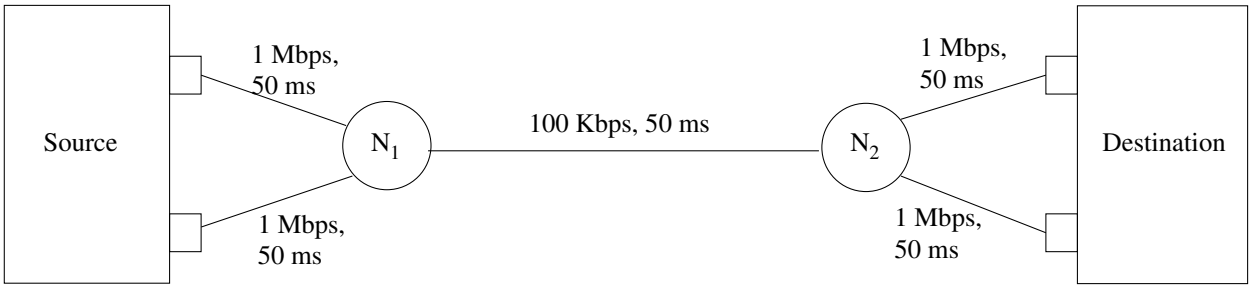


Figure 4.26: Network topology for paths with shared link.

As before, the source transmits a 16 MB file to the destination and we observe the total time taken for this transfer for varying values of the packet error probability for the $N_1 - N_2$ link. These results have been plotted in Figure 4.27 for the three multipath protocols, CMT, MPTCP and MPSCTP.

As expected, the presence of the shared link makes the average file transfer time higher for this case when compared with the disjoint path network considered earlier in Section 4.8.1.1. As in Section 4.8.1.1, MPTCP shows a poorer performance than both CMT and MPSCTP. It is interesting to note that in this network with a shared link, CMT performs somewhat better than MPSCTP for higher packet loss probabilities. This is due to the fact that MPSCTP increases the congestion window in accordance to the path characteristics like round trip time and probability of error and it also encounters higher retransmission timeouts in case of acknowledgement losses on shared link. This causes a slow increase in congestion window for both the paths because of the shared link (Since CMT does not do this it shows a better performance than MPSCTP). It may be noted that this is not really an advantage for CMT because the BERP approach makes MPSCTP more "friendly" towards other fellow flows on the shared link (i.e. it does not capture unduly higher bandwidth as CMT tends to do).

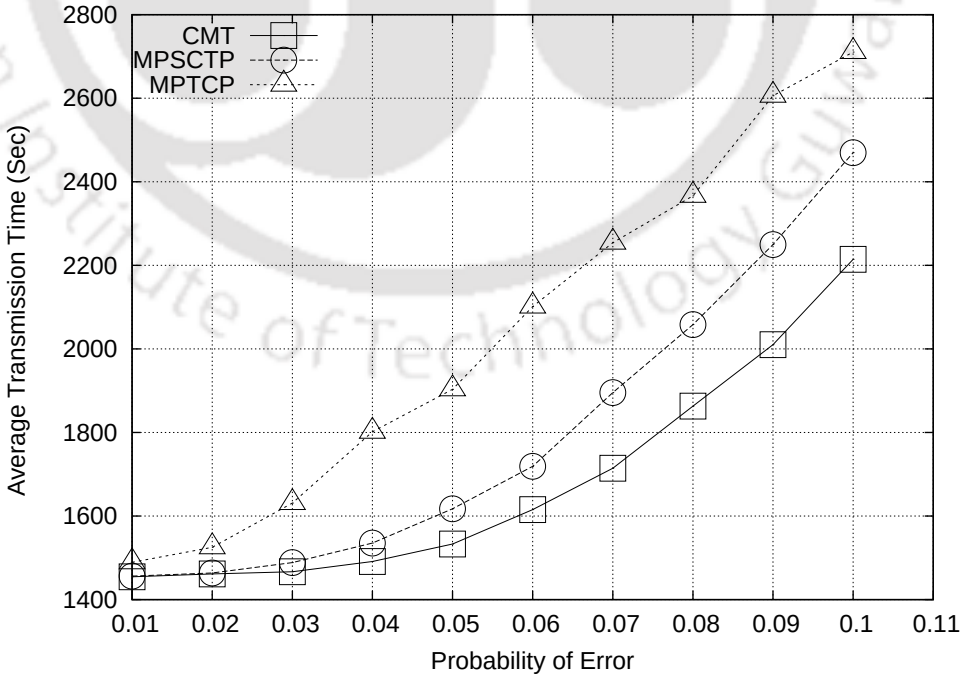


Figure 4.27: Average transmission time for paths with shared link.

We have used R. Jain [75] formula to measure the Fairness Index of CMT and MPSCTP protocols. Fairness Index is a value between 0 and 1. As the value approaches to unity, the fairness of the system increases where 1 indicates the equal allocation of the resources. If x_i is the throughput of the i^{th} flow, the *Fairness Index* is given by (4.13), where n is the total number of flows in the network.

$$F = \frac{(\sum_{i=1}^{n} x_i)^2}{n * (\sum_{i=1}^{n} x_i^2)} \tag{4.13}$$

For the measurement of TCP friendliness, we have considered the same topology as in Figure 4.26 with another TCP source transmitting data through the link $N_1 - N_2$. The Fairness Index for the scenario is given in Table 4.2.

Table 4.2: Fairness Index for CMT and MPSCTP

Fairness Index	
CMT	MPSCTP
0.9662	0.9923

These values indicate that MPSCTP is friendlier and does not unduly harm the fellow flows on the shared links.

4.8.1.3 Disjoint Paths with High Loss

To test the performance of the protocols, we have also considered the network with disjoint paths where the path losses are high. We consider paths with higher losses and significantly higher bandwidth than the values assumed for the tests in Section 4.8.1.1 and Section4.8.1.2. The bit rates and the link delays of various links in the network are as shown in Figure 4.28. The packet error probabilities for the links $N_1 - N_3$ and $N_2 - N_4$ are varied from 10% to 30%. All other links in the network have packet error probability of 1%.

We once again test this network with a 16 MB file transferred from source to destination over a large number of simulation runs. The mean times taken for the transfer by MPTCP, CMT and MPSCTP are plotted in Figure 4.29.

4.8.1.4 Paths with only Congestion Loss

We have also considered a scenario where paths have only congestion losses. For this we consider the scenario shown in Figure 4.30. We have considered all the links in the network to be error free.

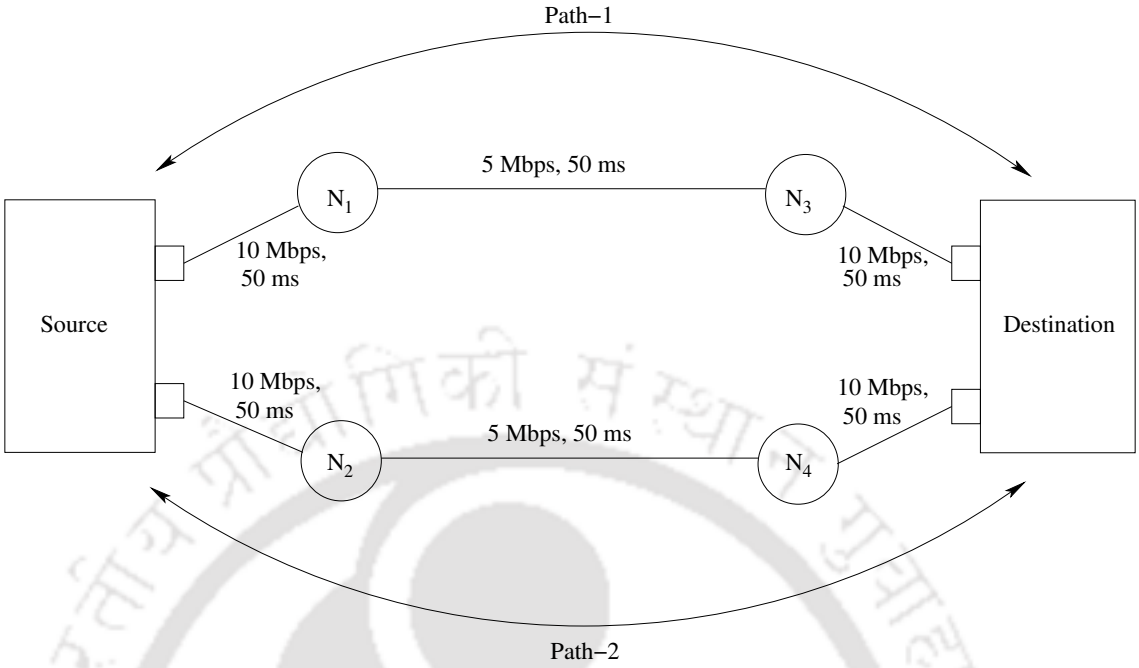


Figure 4.28: Network topology for disjoint paths with high losses.

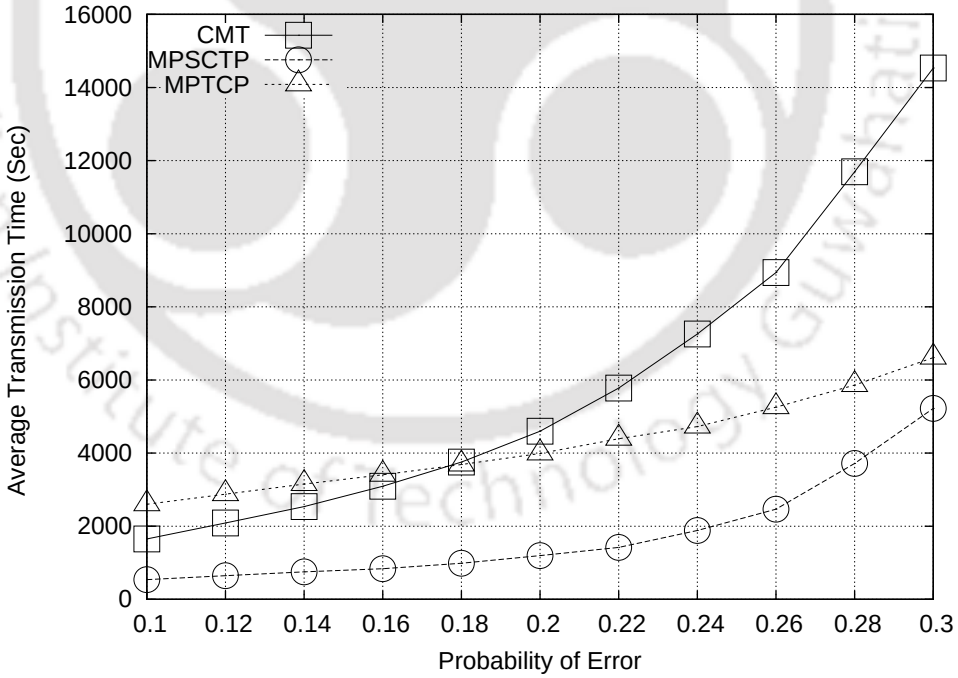


Figure 4.29: Average transmission time for lossy links.

All other properties of the links are same as in Section 4.8.1.1

There are two UDP sources attached at N_1 and N_2. These sources generate CBR traffic with

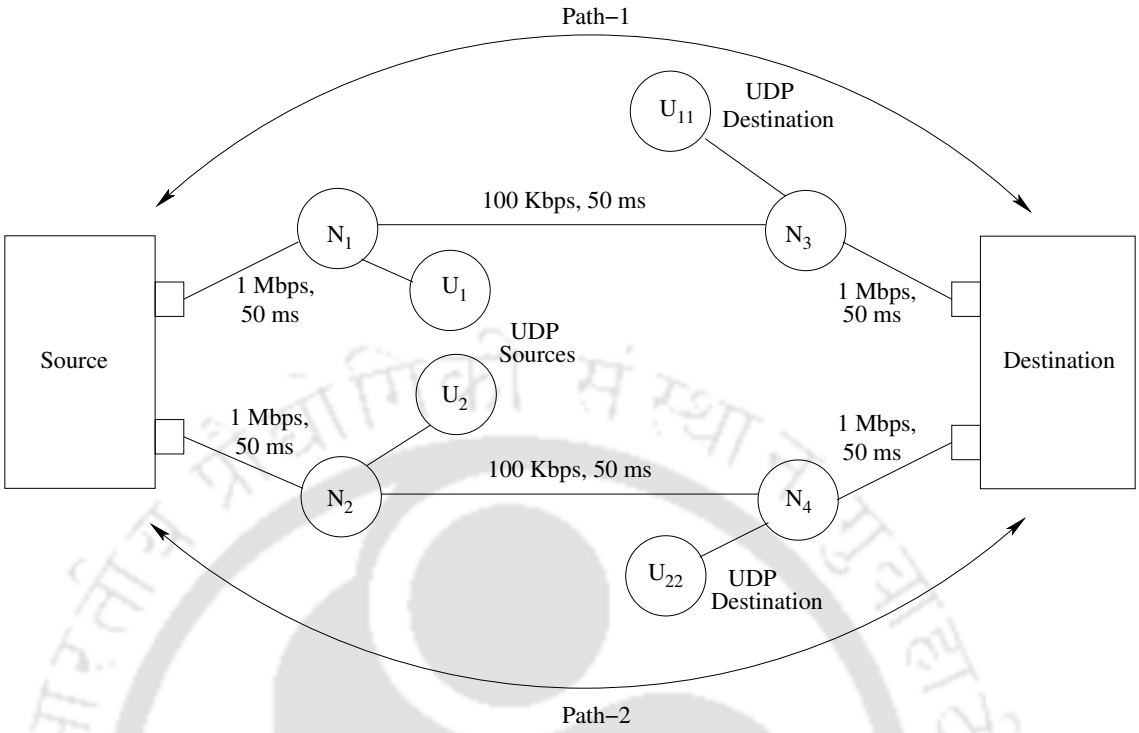


Figure 4.30: Network topology for paths with only congestion losses.

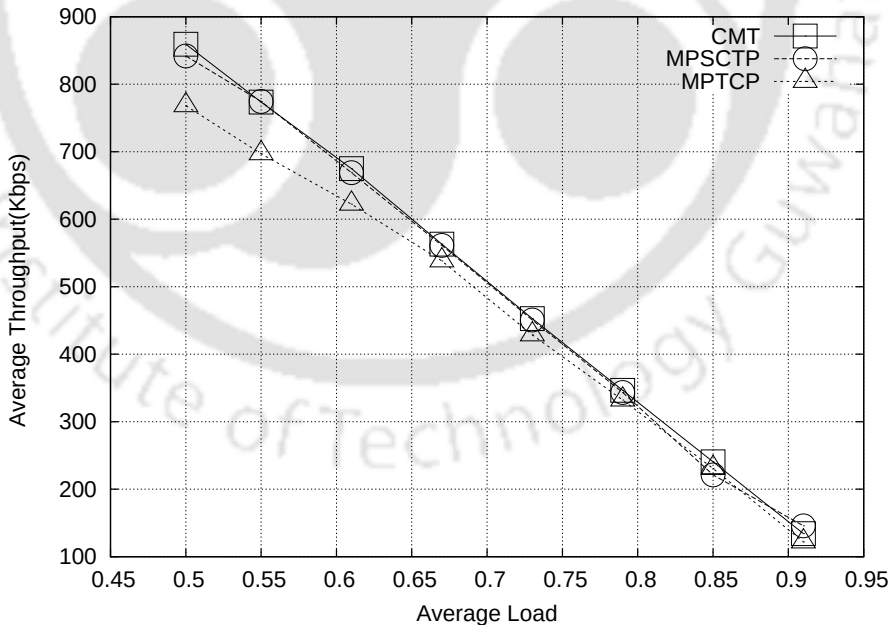


Figure 4.31: Average throughput for only congestion loss topology with different average load of UDP traffic.

packet size of 512 bytes. The links between the UDP sources and the corresponding nodes have bandwidth of 10 Mbps and delay of 50 ms.

For this scenario, we show in Figure 4.31, the average throughput achieved for all the protocols under consideration. As expected, all the three protocols exhibit almost identical performance. Since all the losses in this scenario are caused by congestion, all the three protocols will perform in almost similar fashion.

4.9 Conclusion

In this chapter, we proposed the MPSCTP protocol as an augmentation of SCTP. Though it would require a change in the header structure and some algorithmic changes at the transport layer of the end-nodes, MPSCTP is expected to use network resources better and will provide greater reliability with concurrent multi-path usage. It is simpler to implement than CMT and performs better with higher throughput and fewer retransmission attempts. We observe that paths with unequal bandwidth distribution perform relatively poorly in the examples shown. This may be tackled by optimum flow division based on the path characteristics and QoS requirements along with MPSCTP.

We have also proposed a modified congestion control algorithm for multipath protocols. We have compared the performance of BERP algorithm with TCP like congestion control. We have tested and compared it for CMT. Our proposed algorithm (BERP) uses resource pooling during congestion detection phase for better overall performance where the bandwidth resource available is obtained using a suitable bandwidth estimation approach. The results indicate that CMT-BERP algorithm adapts better to packet drops than CMT congestion control algorithm and therefore provides better resource utilization. We have also compared the performance of CMT, MPSCTP with modified congestion control (MPSCTP-BERP) and MPTCP. The results indicate that MPSCTP generally performs better than both MPTCP and CMT and that the improvement observed is high when the packet error probabilities are high. Even in the case of shared links where MPSCTP's performance is slightly inferior to that of CMT, the former would probably still be the preferred choice because it remains "friendly" to other flows that may also use the shared link. As far as MPTCP is concerned, one reason for its poorer performance compared to CMT and MPSCTP is the suboptimal retransmission policy that it adopts because of compatibility issues with traditional

TCP. Our results indicate that MPSCTP performs better than both CMT and MPTCP. We have presented BERP congestion control for SCTP variants only, however the same algorithm can be adapted for other transport protocols as well.

5

Heuristics for Implementation of Flow Division Optimizations

Contents

In previous chapters, we introduced different strategies for flow division at source. Flow division is essential in any multipath protocol for improved resource utilization. The choice of flow division mechanism also depends upon the nature of the particular application under consideration. The proposed flow division approaches can be implemented provided the complete knowledge of the path statistics is known at the source. However, in an Internet like scenario, it is difficult to obtain the complete path statistics. Hence, this chapter proposes two heuristics which approximate min-max and delay sensitive optimizations. We demonstrate that these heuristics are easily implementable in an Internet like scenario and closely approximate the original flow division approaches.

5.1 Introduction

Existing multipath protocols do not optimize the flow division on the available multiple paths. There have been various efforts in the past to divide data flow on multiple paths [4], [76]. Various optimized flow division approaches for dividing the data flow on multiple paths have been proposed in [77][‡]. In Chapter 3, we had considered two classes of applications, namely delay sensitive applications and delay insensitive applications. For these two classes of applications, we have proposed Min-Max optimization and Delay Insensitive Optimization. A true implementation of these optimizations would require complete network statistics to be available at the source. Since, this is difficult in an Internet-like scenario, suitable heuristics are required which can closely approximate these optimizations. In this chapter, we propose a heuristic for the Min-Max optimization and demonstrate that a practical implementation of this heuristic can be closely approximated by the Bandwidth Estimation based Resource Pooling (BERP) algorithm [78][‡]. We have also subsequently proposed a suitable heuristic to implement Delay Insensitive Optimization.

5.1.1 Min-Max and BERP Algorithms

In Chapter 3, we introduced Min-Max optimization for flow division of the data on available paths. Min-Max optimization minimizes the maximum mean end-to-end delay experienced by the packets on any of the candidate paths. The generalized version of the Min-Max optimization problem with due consideration of the propagation delays of the respective links is as follows:

[‡]Author's joint publication.

minimize:

$$\max_i \left(\sum_{j=1}^{L_i-1} (w_{ij} + d_{ij}) \right) \qquad \forall i = 1, 2, ..., N \tag{5.1}$$

subject to:

$$\sum_{i=1}^{N} \lambda_i = \lambda \tag{5.2}$$

$$\lambda_i \geq 0 \tag{5.3}$$

where,

$w_{i,j}$ is the average delay at the j^{th} router of the i^{th} path,

$d_{i,j}$ is the propagation delay at the j^{th} link of the i^{th} path,

λ_i is arrival rate of the subflow at ithpath,

λ is net rate contributed by the source,

L_i is the number of links on i^{th} path,

N is the total number of candidate paths.

Note that Min-Max optimization leads to the same average delay on all the available paths.

The Bandwidth Estimation based Resource Pooling (BERP) has been presented earlier in Chapter 4. BERP suggests changes in the Congestion Avoidance and Congestion Detection phases based on the bandwidth of the available paths. Some of these expressions are reproduced here once again for ease of reference and continuity.

Congestion Avoidance Phase: Increase the value of *cwnd* of i^{th} path by:

$$min(\alpha * P_a * MTU/c_T, P_a * MTU/c_i) \tag{5.4}$$

$$P_a = P_a - c_i \tag{5.5}$$

Congestion Detection Phase: *if (four duplicates are received)*

$$s_i = max(c_i - \beta_i * c_i, 4 * MTU) \tag{5.6}$$

$$c_i = s_i \tag{5.7}$$

if (Timeout Occurred)

$$s_i = max(c_i - \beta_i * c_i, 4 * MTU) \tag{5.8}$$

$$c_i = 1 * MTU \tag{5.9}$$

where,

s_i = slow start threshold of i^{th} path,

$$\beta_i = \frac{BWE_i}{\sum_i BWE_i} \tag{5.10}$$

where,

BWE_i= Bandwidth estimate of i^{th} paths

BERP computes the sample of available bandwidth every RTT and these samples are low pass filtered for the bandwidth estimates [78]‡.

5.2 Heuristic for Min-Max Optimization

The equations (5.1) - (5.3) can be solved using the Line Search Method [79]. Let $\lambda^t = \{\lambda^1, \lambda^2, ..., \lambda^N\}$ be a feasible solution. Then $\lambda^{t+1} = \lambda^t + \alpha d^t$, where α is a scalar, will be a better solution if

$$f(\lambda^{t+1}) \leq f(\lambda^t) + \alpha(\nabla f(\lambda^t).d^t) \tag{5.11}$$

where d^t is the direction of the change. Let us assume that the direction of change is always +1 if we have started from the initial condition. Hence, for a possible solution

$$f(\lambda^{t+1}) \leq f(\lambda^t) + \alpha(\Delta f(\lambda^t)) \tag{5.12}$$

where $\Delta f(\lambda^t)$ gives the instantaneous change in the direction of d^t, ensuring would suffice.

‡Author's joint publication.

From (3.14) it is clear that as the arrival rate at any link increases, the delay experienced by the packet on that link will also increase. The delay experienced by a packet is a monotonically increasing function of the arrival rate on that path. Hence, as shown in (5.13), the change in arrival rate (λ) is a function of the end-to-end delay (w) experienced on that path.

$$\delta(\lambda) \propto f(w) \tag{5.13}$$

According to the Min-Max algorithm, the two paths should have the same delay at equilibrium. The delay experienced on the two paths will be a function of the respective arrival rates on the paths. To make the delay on the two paths equal, the flow on the path with higher end-to-end delay should be increased by an amount less than the increase of flow on the path with the lower end-to-end delay. This indicates that the change of flow rate on the two paths is not independent and that the change will also be a function of the total arrival rate. According to the line search method, this approach will eventually lead to the same delay for both the paths [80][‡].

5.3 Implementation using BERP Algorithm

In this section, we show that heuristic proposed in previous section is closely approximated by the BERP congestion control algorithm.

Consider two paths between source and destination with respective flows λ_1 and λ_2 on path-1 and path-2. The respective delays of the paths are w_1 and w_1 where we assume that $w_1 > w_2$. As already mentioned, the delays of the two paths can be made equal by changing the flows such that the increase of flow on the path with larger end-to-end delay is less than the flow increase on the one with smaller end-to-end delay i.e. $\delta(\lambda_1) < \delta(\lambda_2)$. Note that the change in the flow will also be in proportion to the delays of the respective paths. Since the flow/delay changes for the two paths are not independent of each other, an intuitive approach will be to make the changes in weighted proportion of the delays of the respective paths so that the traffic on the path with smaller delay is increased by larger value. Hence, a possible increment rule can be as follows:

$$\delta(\lambda) \propto f(w_1, w_2)/(w_1 + w_2) \tag{5.14}$$

For any transport layer, the congestion window decides the amount of data that can exist on

[‡]Author's joint publication.

the path. Hence, *cwnd* (c) is in proportion to the arrival rate (λ on that path, i.e. $\lambda \propto c$. Since the end-to-end delay of any path is a function of the arrival rate on that path, this is also in proportion to the congestion window of the path, i.e. $w \propto c$. Hence, 5.14 can be rewritten as

$$\delta(c_i) \propto k.f(c_1, c_2)/(c_1 + c_2)$$

$$=> \delta(c_i) \propto a/c_T \tag{5.15}$$

where $c_T = (c_1 + c_2)$ and a is a factor of proportionality which depends on the network condition and the congestion window of the sender. It may be recalled that the standard TCP heuristic is to increase the congestion window by $1/c$ for every ack. Since we do not want the existing flow to be more aggressive than this and also want the increase rule to be "friendly" to TCP, the following increase rule is used

$$\delta(c_i) \propto min(a/c_T, 1/c_i) \tag{5.16}$$

Since SCTP counts the congestion window in bytes hence the (5.16) is modified as:

$$\delta(c_i) = min(a * P_a * MTU/c_T, P_a * MTU/c_i) \tag{5.17}$$

Where P_a is the partial bytes acknowledged and MTU is the Maximum Transmission Unit. Note that this is the same increase rule as that proposed for the BERP algorithm in (5.4). Therefore, the BERP algorithm closely approximates the Min-Max optimization approach for flow division. We have demonstrated this for two paths. However, the same logic can be extended for more paths between the source and the destination.

5.4 Simulation Results

We have implemented the BERP algorithm for MPSCTP protocol in *ns-2*. To verify that BERP indeed approximates the Min-Max optimization for a network, we consider the example network topology of Figure 5.1. This network consists of dual homed source (S) and destination (D) with two disjoint paths between them. Both paths consist of several intermediate nodes, UDP on-off sources (U_1 & U_2) and UDP destinations (U_{11} & U_{22}). The packet loss probability for all the links

on both the paths is assumed to be 2%. The link rate for path-1 is 1 Mbps while for path-2 it is 100 Kbps. The link delay for path-1 is 500 ms. For the links on path-2, delay varies over 10 - 900 ms. The values are chosen such that the Bandwidth Delay Product (BDP) differs significantly for the two paths. For our tests, we run each simulation for 500 seconds and measure the difference between the average packet delays for the two paths. The results obtained for various values of link delays for the path-1 for both CMT-SCTP and MPSCTP-BERP are shown in Figure 5.2.

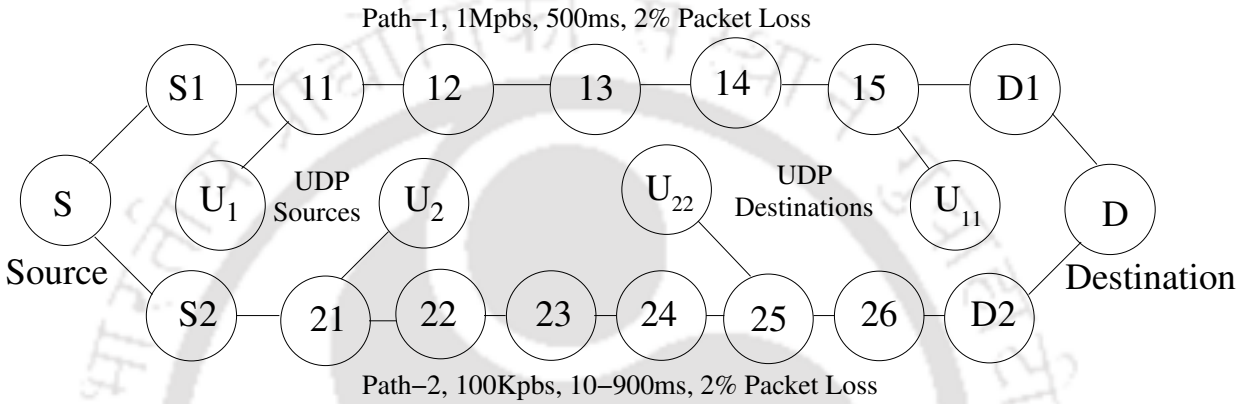


Figure 5.1: Network Topology.

The results in Figure 5.2 show that the difference in packet delay over the two paths for BERP is less than that for CMT-SCTP. BERP tends to make the delay of two paths closer to each other as desired by the Min-Max algorithm. We have also obtained the variance of packet delay for different link delays as shown in Figure 5.3. Figure 5.3 shows that BERP exhibits smaller variance for path delay difference than the traditional algorithm. This makes this algorithm more suitable for real time applications because less delay variance would simplify receiver buffer design.

When the delay for path-2 is around 400 ms, the packet delays between the two paths are similar and at their lowest values for both schemes. (The delay variance is also the least at this point.) This is expected since the two paths are then equal in terms of their path delays.

To test the performance of the proposed algorithm under dynamic conditions, we consider a network with two disjoint direct links between the dual-homed source and destination nodes. The bandwidth of path-1 is 1 Mbps with varying end-to-end delay and that of path-2 is 100 Kbps with end-to-end delay of 200 ms. The end-to-end delays of packets are plotted in Figure 5.4 and Figure 5.5 for MPSCTP-BERP and CMT-SCTP respectively. To study dynamic behavior, step increases are added to the delays of path-1 at 125 seconds and 225 seconds. The results show that when

the delay of path-1 changes (as shown in Figure 5.4), the BERP adjusts the flows (Figure 5.5) so that the average delay difference between the two paths is reduced, i.e. by an overall increasing trend in the delay over path-2 following those time instants. Similar step change in delay has been added for CMT-SCTP as well (Figure 5.6. Since CMT-SCTP does not use resource pooling and its congestion windows for the two paths vary independently, no adjustment of flow or delay can be discerned in the simulation results of path-2 (Figure 5.7).

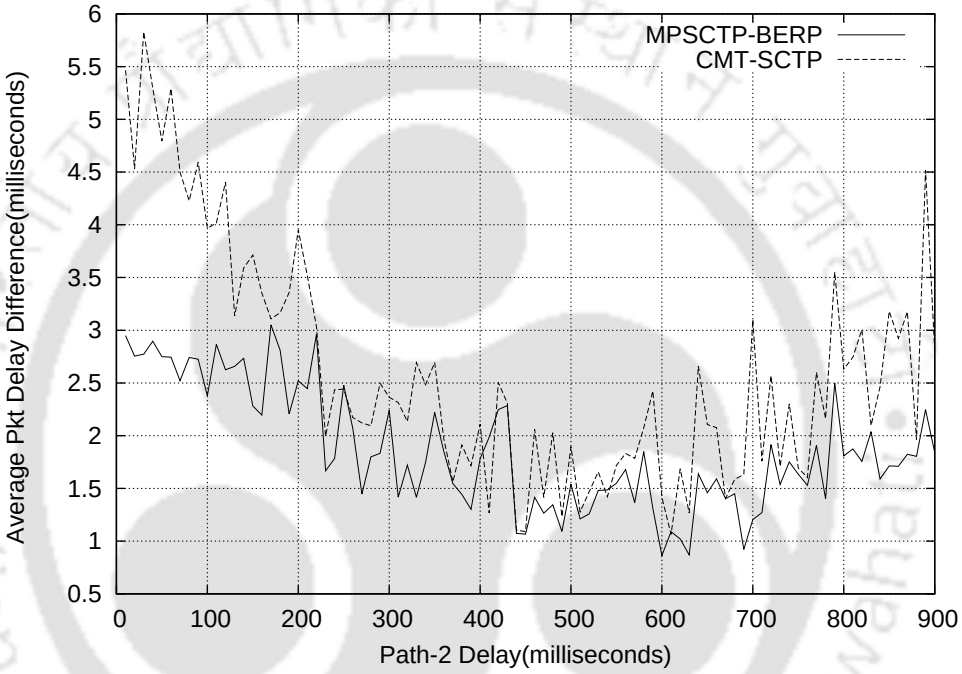


Figure 5.2: Average packet delay difference between path-1 and path-2 for various link delays of path-2.

5.5 Heuristic for Delay Insensitive Optimization

MPSCTP protocol in its original form does not deal with flow division on multiple paths. A proper flow division is essential to take complete advantage of any multipath protocol. The flow division on candidate paths depends upon the nature of the underlying application with varied QoS requirements. In this section, we present a heuristic approach for delay insensitive optimization that can be used to divide the flow on the candidate paths such that the agreed QoS parameter with the user is achieved. We have considered end-to-end delay as the QoS parameter for this optimization and MPSCTP is being used as the protocol to implement this optimization.

Delay Insensitive Optimization ensures that the delay of any of the candidate paths does not go

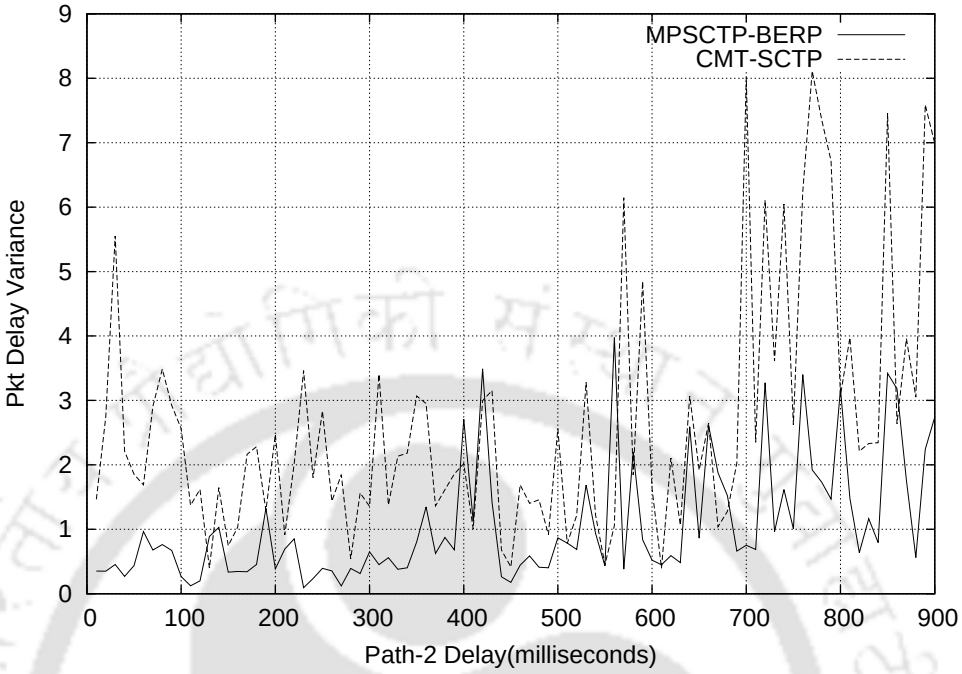


Figure 5.3: Packet delay variance for MPSCTP-BERP and CMT-SCTP for various link delays of path-2.

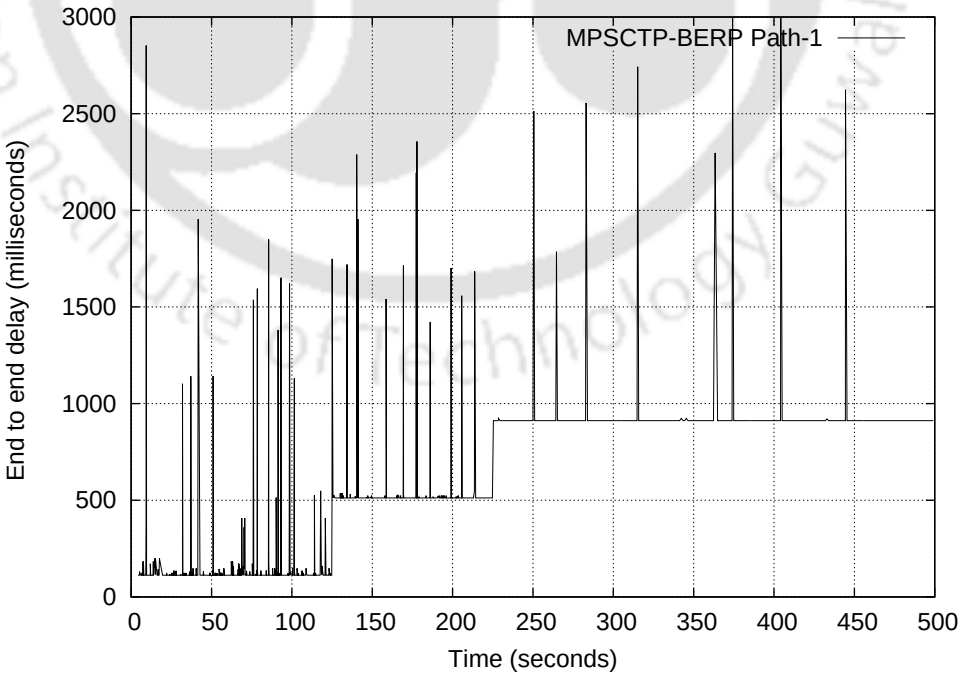


Figure 5.4: End-to-end packet delay for MPSCTP-BERP for Path-1.

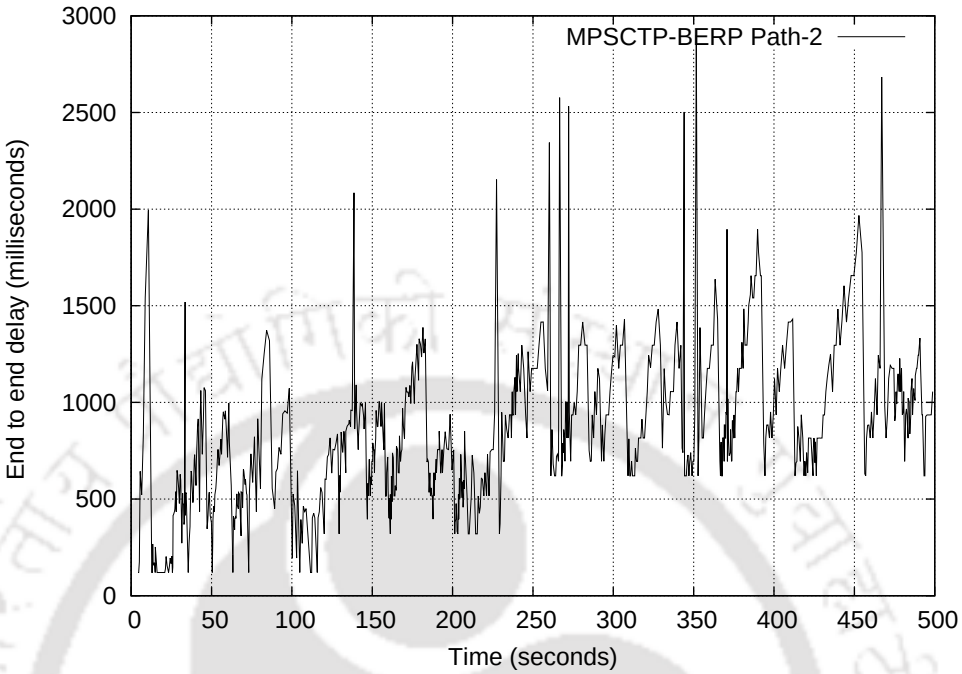


Figure 5.5: End-to-end packet delay for MPSCTP-BERP for Path-2.

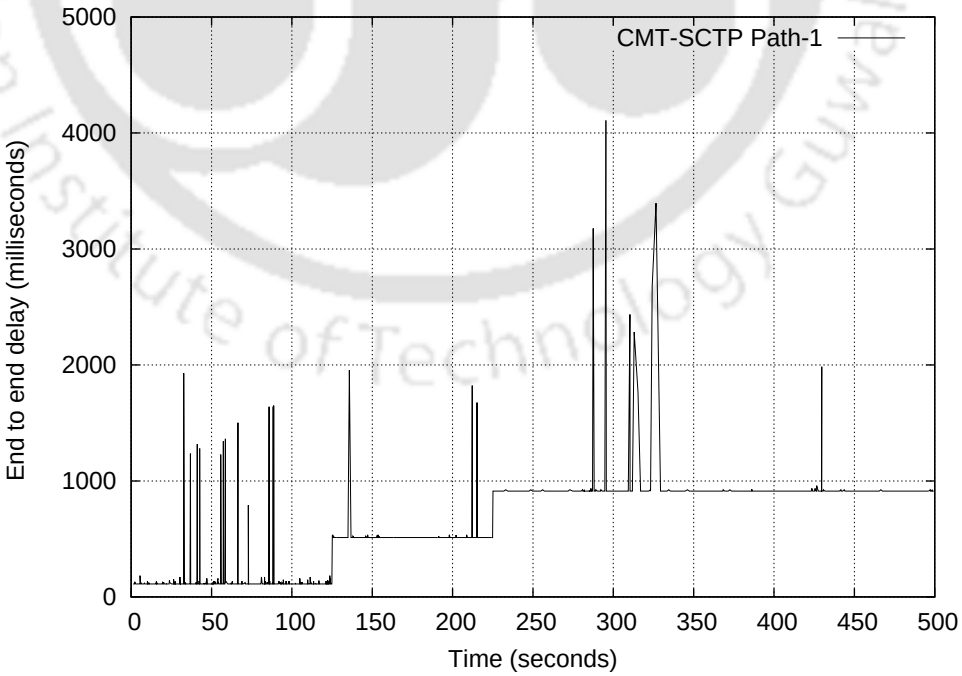


Figure 5.6: End-to-end packet delay for CMT-SCTP for Path-1.

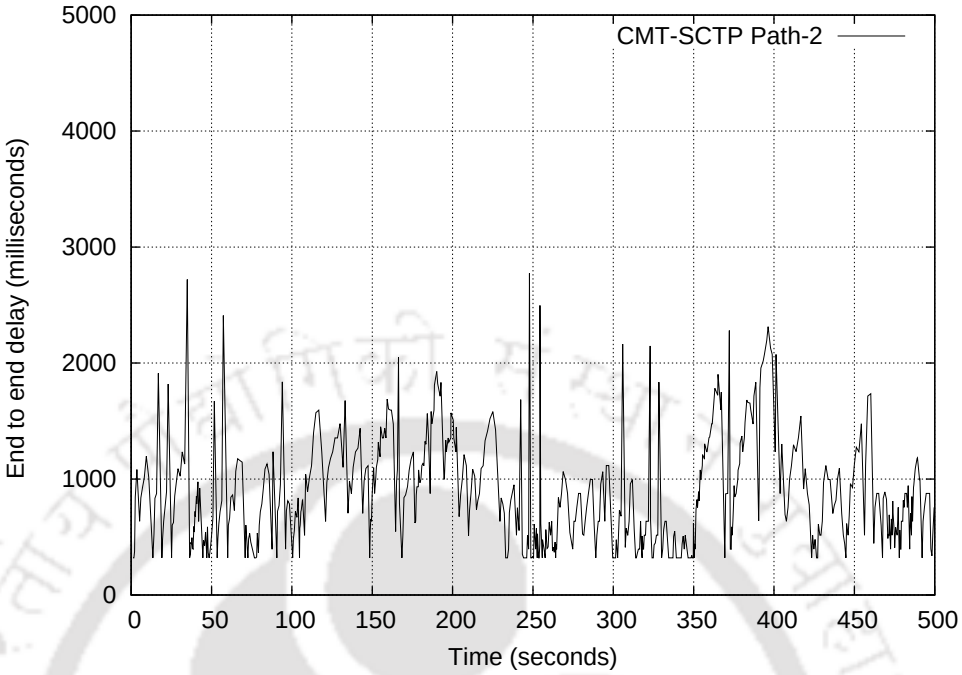


Figure 5.7: End-to-end packet delay for CMT-SCTP for Path-2.

beyond a pre-agreed value. This delay parameter is the QoS parameter provided by the optimization. This parameter depends upon the nature of application as well. We have discussed Delay Insensitive Optimization and its performance in Chapter 3. The expressions for this optimization are reproduced here with propagation delays also taken under consideration.

maximize:

$$\sum_{i=1}^{N} \lambda_i \tag{5.18}$$

subject to:

$$\sum_{j=1}^{L_i-1} (w_{ij} + d_{ij}) \leq T_i \qquad \forall i = 1, 2, ..., N \tag{5.19}$$

$$\lambda_i \geq 0 \tag{5.20}$$

where,

T_i is the agreed upon end-to-end *Target Delay* for i^{th} path,

d_{ij} is the proportion delay at the j^{th} link of i^{th} path,

w_{iJ} is the mean delay for the j^{th} link of i^{th} path,

λ_i is the traffic rate for i^{th} path,

L_i is the number of links on i^{th} path,

N is the total number of candidate paths.

The basic requirement to implement Delay Insensitive Optimization is that packets should not experience an end-to-end delay on any path which is more than the Target Delay (T_i). The end-to-end delay on each path is also a function of flow sent on that path. Hence, it is possible to adjust the end-to-end delay experienced by any packet by adjusting the flow on that path. The flow on any path should be adjusted such that the end-to-end delay experienced by a packet should not go beyond the agreed Target Delay (T_i). However, the path properties are dynamic in nature; hence an exact analysis of flow to be carried on a path is a costly operation both in terms of time and resources required. Hence, we use the round trip time measurement for the end-to-end delay calculations. It has also been demonstrated in earlier work [81] that RTT measurements in Internet scenario are good approximations and give close results to Auto Regressive models.

In a typical SCTP scenario, the congestion window is used to control the amount of data in the network in single RTT. At any point of time any path cannot have more than cwnd bytes outstanding on it. According to the modified congestion control for MPSCTP, its *cwnd* is increased for every SACK received. As the flow on any path is increased, the delay encountered by a packet is also increased. Hence, a change in *cwnd* based on the RTT measurements can be used to adjust the delay experienced by a packet in the network.

Hence, we propose to modify the congestion control algorithm as in Figure 5.8 to take care of the delay experienced by any packet on the network.

According to this algorithm, when the RTT approaches within a range (depends upon the parameter 'q' discussed later) of the target delay, the cwnd is not increased anymore. Since the network delays also depend upon other activities in the network, whenever the round trip time delay is more than a certain value of the target delay, the algorithm enters the congestion detection phase and reduces the congestion window to adjust the flow on the network. Note that the effect of any change in flow will be visible at the source only after at least one cwnd bytes are acknowledged; this is because there are already cwnd bytes of data outstanding in the network at the time of flow adjustment. Hence, we do not perform another adjustment unless one cwnd bytes are acknowledged

if ($RTT_i \leq$ *(1-q)* $* 2 * T_i$*)*
    if (cwnd $\leq$ ssthresh)
        Apply Slow Start.
    if (cwnd $>$ ssthresh)
        Apply Congestion Avoidance.
if (at least cwnd bytes are received since last cwnd adjustment)
    if ($RTT_i > (1-q) * 2 * T_i$ && $RTT_i \leq (1+q) * 2 * T_i$)
        Do Not Increase cwnd.
    if (cwnd $> (1+q) * 2 * T_i$)
        Decrease cwnd as per congestion detection algorithm.

Figure 5.8: Algorithm for Delay Insensitive Optimization

since the last adjustment. The upper and the lower bound of this limit are decided by the value of the factor 'q'. This factor 'q' is a function of variance of the delay of the network i.e. its value is decided by how much maximum delay jitter exists in the network. In this thesis we have assumed 'q' to be 0.1. That indicates that the maximum delay jitter in the network is not more than the 10% of the target delay. However, in the networks with different delay jitters the value of 'q' can be chosen accordingly.

Unlike standard congestion control algorithm, where the protocol keeps increasing the flow on the network until a loss is occurred, our proposed algorithm does not allow the traffic to increase on the network if the end-to-end delay starts approaching the target delay. This behavior requires a penalty to be paid in terms of the loss of throughput for the application and keeps the end-to-end delay close to the target delay. However, this would help in improving the throughput of other fellow flows on the network. Our proposed algorithm will not increase the flow on a path if this leads to a situation where the network would not be able to provide the delay performance targeted for the flow on that path. This gives an opportunity to other flows to use the network resources and in turn helps in running the network efficiently.

For a network we measure the Fairness Index using (4.13). We also measure the. Link Utilization for the shared bottleneck link in the network as in (5.21)

$$Link\,Utilization = \frac{\sum_{i=1}^{n} \lambda}{Bottleneck\,Link\,Bandwidth} \tag{5.21}$$

Here λ_i is the throughput of i^{th} flow entering on the bottleneck link, and n is the total number of flows entering that link. The purpose of this parameter is to measure the utilization of the bottleneck link. These two factors jointly indicate the reliability and the fairness of the proposed algorithm. By observing the two parameters, the fairness index and the link utilization together, we demonstrate that though the optimization reduces the throughput of the source in order to satisfy the QoS constraint, this may cause an increase in relative unfairness. However, this gives more opportunity to other fellow flows to transmit more data and the overall utilization of the network's resources does not decrease.

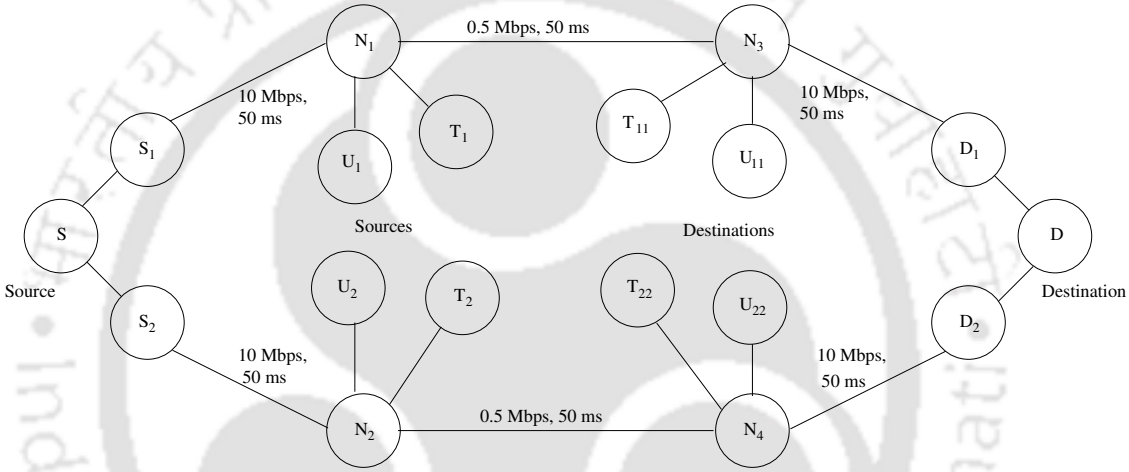


Figure 5.9: Network Graph for Delay Insensitive Optimization.

5.6 Simulation Results

Let us consider a network as shown in the Figure 5.9. The network consists of two UDP sources (U_1 & U_2), two TCP sources (T_1 & T_2), and one dual homed source S with MPSCTP source attached to it. There are ftp sources attached at T_1, T_2 and S while U_1 and U_2 have CBR sources attached to them. All the links are assumed to have packet loss probability of 10%. The links have bandwidth as shown in Figure 5.9. Remaining links have bandwidth of 2 Mpbs and propagation delay of 50 ms.

The average throughput achieved by the MPSCTP with the proposed optimization for various target delays (T_i) is presented in Figure 5.10. These results indicate then when the target delay is less than 160 ms, no data is carried on the network because other background traffic in the network does not allow any data to reach the destination within the agreed upon target delay. As the target

delay increases beyond 180 ms the throughput of network increases rapidly. However, there is no significant increase in the throughput if the target delay is increased beyond 260 ms. This is due to the fact that the network is running at its maximum capacity for this target delay.

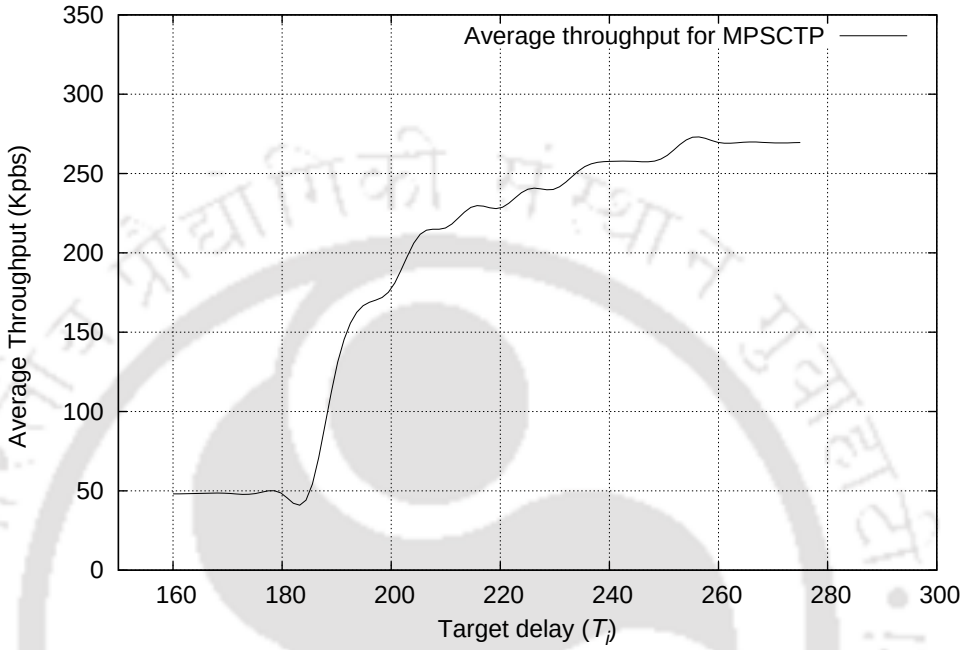


Figure 5.10: Average Throughput for MPSCTP for various Target Delays.

Figure 5.11 and Figure 5.12 present the respective packet delay dynamics with delay sensitive optimization and without it for a target delay of 210 ms. Figure 5.13 and Figure 5.14 shows the end-to-end packet delay dynamics for target delay of 270 ms with delay sensitive optimization and without the optimization respectively. The results show that with delay optimization, the variance of delay of the packets as well as the maximum delay experienced in the network is reduced. Note that Figure 5.10 gives an indication that there is a throughput loss when the delay insensitive optimization is used. We also notice (Table 5.1) that there is a decrease in the Fairness Index for target delays of 210 ms and 270 ms because the optimization reduces the data to be carried on the links. However, the link utilization is not effected; this indicates that other fellow flows are able to carry more data on that link. However, as we earlier asserted, this is still useful to provide the end user a better quality of service and also allow other fellow flows to transmit their data on this link without compromising the overall resource utilization. Table 5.1 also shows the percentage of packets which does not comply with the target delay. There is a significant reduction in this

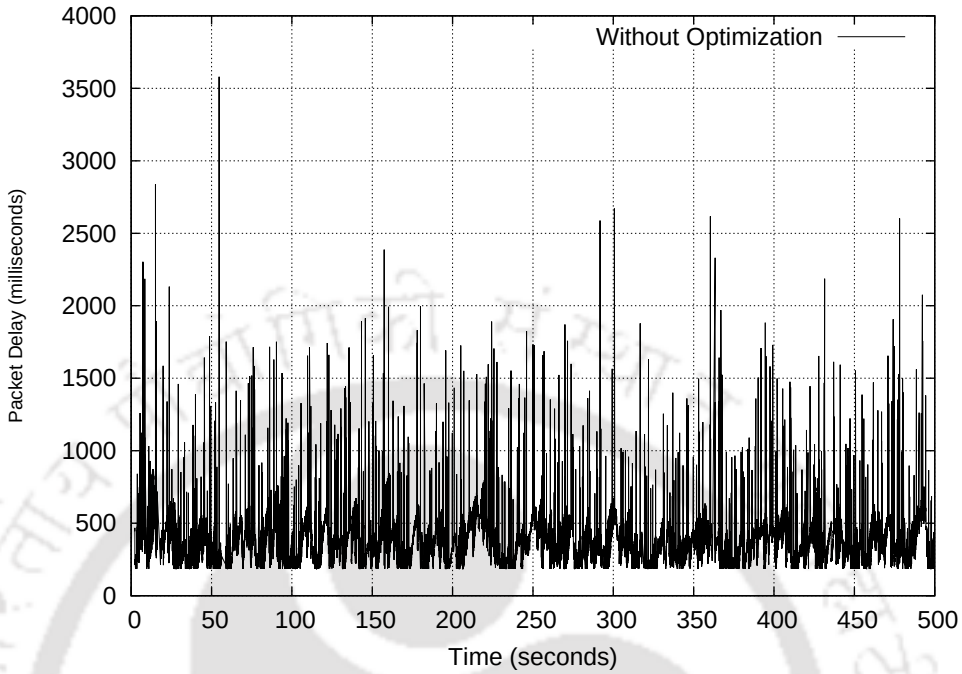


Figure 5.11: End-to-end packet delay for the target delay of 210ms without optimization.

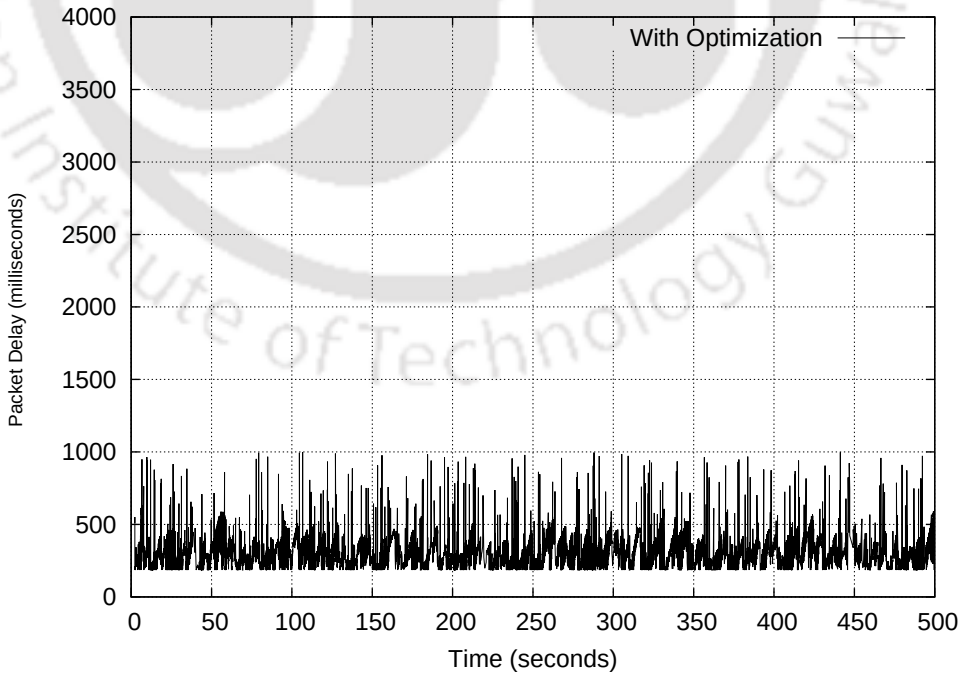


Figure 5.12: End-to-end packet delay for the target delay of 210ms with optimization.

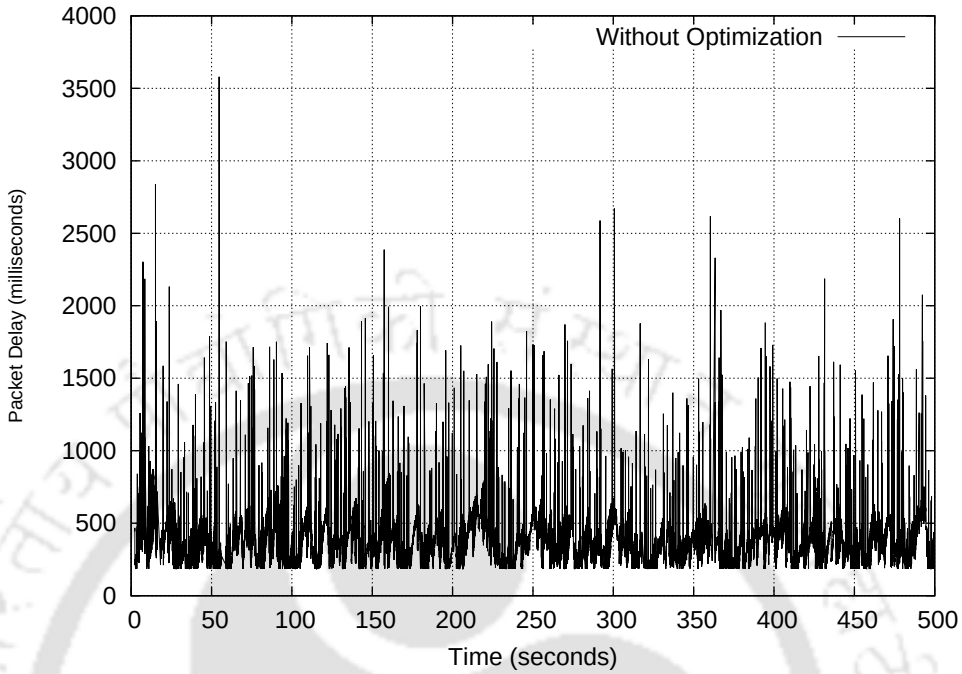


Figure 5.13: End-to-end packet delay for the target delay of 270ms without optimization.

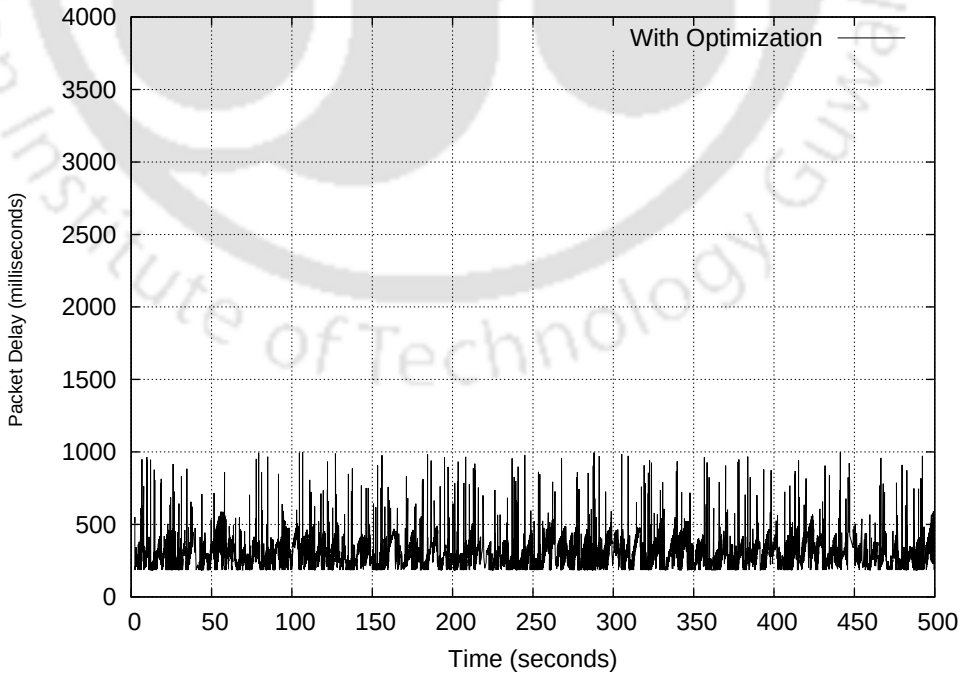


Figure 5.14: End-to-end packet delay for the target delay of 270ms with optimization.

percentage when the proposed optimization is used. As expected, the results also show that when the agreed target delay value is higher, significantly fewer packets will encounter target delays more than their respective target delays. However, it should be noted that still significant number of packets violate the target delay. The reason is that the proposed heuristic is a reactive one and takes only reactive measures. At the same time, due to various transient activities in the network, it is very difficult to control the RTT of every packet. This further indicates that it is possible to design another strategy to achieve optimal solution.

Table 5.1: Fairness Index and Link Utilization for link $N_1 - N_3$

Policies	Fairness Index		Link Utilization		% Packet Delayed	
	210 ms	270 ms	210 ms	270 ms	210 ms	270 ms
With Optimization	0.9787	0.9802	0.9359	0.9357	46.79	27.79
Without Optimization	0.9893	0.9892	0.9361	0.9364	81.31	80.54

5.7 Conclusions

We have proposed a practical implementation of the Min-Max optimization based upon a congestion control algorithm which uses Bandwidth Estimation based Resource Pooling. We have implemented and tested this algorithm for Multipath SCTP, an extension to SCTP proposed by us earlier. We have compared our results of MPSCTP-BERP with CMT-SCTP. The results indicate that BERP not only reduces the difference in average packet delay over the different paths but also reduces the variance of the delay experienced over these paths. This suggests that this algorithm is a good choice for delay sensitive applications because of simpler receiver buffer management.

We have also suggested a heuristic mechanism to provide QoS based service for delay insensitive applications using MPSCTP. This heuristic approach uses maximum target end-to-end delay requirement as the QoS parameter to be considered. Based upon the choice of the end-to-end delay requirement, the throughput for the end-user will be affected. However, we suggest that this heuristic adopts a non greedy approach while transmitting data on the network and hence helps in improving the overall user experience.

These heuristics approximate the original optimization approaches and provide easily implementable schemes which can be put in place in actual networks.

6

MPSCTP Implementation in Linux Kernel

Contents

In this chapter, we present an implementation of MPSCTP protocol in Linux kernel. We have incorporated our changes in Linux kernel version 2.6.37 distributed with openSUSE11.4. Linux kernel–2.6.37 is distributed bundled with a loadable module for SCTP. We have augmented this module with changes required for MPSCTP. In this chapter, we discuss the changes which have been incorporated to enable MPSCTP module in the Linux kernel. We also present the performance results for this, both in an emulated network as well as in the LAN environment of IIT Guwahati.

6.1 Introduction

In Section 4.2, we presented the MPSCTP protocol, an augmentation of the SCTP protocol, to support simultaneous transfer on multiple candidate paths between a source and a destination. We also presented in that chapter the simulation results to support the performance of MPSCTP and measured the throughput of the protocol using *ns-2* based simulations. In this chapter, we implement the basic MPSCTP algorithms in the Linux kernel to demonstrate the correctness of the MPSCTP protocol and provide a measure of its performance in a real network. Currently, the Linux kernel supports the SCTP as a loadable module. However, this module has no support for concurrent multipath usage for multihomed end-hosts. We have modified the SCTP module to incorporate concurrent multipath usage. This implementation is tested in an emulated network using *dummynet* [82]. We have also run this protocol on hosts connected through the IIT Guwahati LAN to measure the effect of intermediate switches and background traffic.

6.2 Implementation Details

MPSCTP is developed using the Linux kernel version 2.6.37. All Linux kernels are achieved and maintained at [83]. Though we have tested primarily on openSUSE 11.4, our implementation is expected to work on other Linux variants as well. We changed chunk Header and the SACK structure as discussed in Section 4.2. The corresponding algorithms to handle Fast Retransmission (Section 4.3.1), cumulative acknowledgement updates (Section 4.3.2) and missing PSNs (Section 4.3.3) have been implemented in the kernel. We have proposed a *Tx-CWND* flow division and transmission strategy, where we choose the path with largest congestion window among the available candidate paths for packet transmission and retransmit the lost packets on the path of original transmission. This ensures that we are sending the data on the path which has maximum network

resources available. We have demonstrated in subsequent sections that this strategy performs almost similar to the *RTX-CWND* strategy which has been suggested as the best implementable retransmission strategy [31]. Implementation of the *RTX-CWND* protocol was not possible because of practical difficulties in implementation.

6.2.1 Changes in the kernel source

The following changes have been made in the kernel source:

Multipath concurrent transfer has been enabled for MPSCTP primarily in *outqueue.c:sctp_outq_flush()*. Whenever the packets are flushed out of the queue, the transport on which this packet needs to be transmitted is being set. The function *protocol.c:sctp_v4_get_dst()* is the interface function between IP and MPSCTP layer. This function defines the structures which hold the details about the corresponding source and destination IP addresses.

To assign PSN values we have declared another function sm_make_chunk.c:sctp_chunk_asign_psn() as shown below. This is a helper function called by *output.c:sctp_packet_append_chunk()* to assign PSNs. It maintains the PSN count for each path in the association.

```
....
if (!chunk− > has_psn) {
        chunk− > subh.data_hdr− > psn =
        htonl (asoc− > peer.active_path− > next_psn) ;
        chunk− > has_psn = 1;
}
....
```

Note that the PSN values are monotonically increasing numbers.

Fast Retransmission algorithm is implemented in outqueue.c:sctp_outq_sack() function. This function currently counts the missing count for the chunks and calls the retransmit routine accordingly as shown in the code below.

```
....
if(chunk− > fast_retransmit == SCTP_CAN_FRTX&&!chunk− >
tsn_gap_acked&&PSN_lt (psn, highest_new_psn_in_sack)){
        if (!transport||!sctp_cacc_skip (primary, transport, count_of_newacks, tsn)) {
            chunk− > missing_report + +;
            if (chunk− > missing_report >= 3) {
            chunk− > fast_retransmit = SCTP_NEED_FRTX;
            do_fast_retransmit = 1;
            }
        }
}
....
```

The handling of cumulative acknowledgements is performed by sm_make_chunk.c:sctp_make_sack() function. It internally calls the sctp_psnmap_get_cpsn() helper function to get the cumulative acknowledgement value. We have defined another layer of abstraction find_psnmap_number_for_sack(). This helps in getting the value of cumulative psn acknowledgement for the corresponding path.

```
....
_u32   sctp_psnmap_get_cpsn(const   struct   sctp_tsnmap ∗ map,
                        struct   sctp_transport ∗ last_data_from){
    int   psn_map_number = find_psnmap_number_for_sack(map, last_data_from);
    return   map− > psnmap [psn_map_number] .cumulative_psn_ack_point;
}
....
```

The corresponding updates at the sender side are taken care of by the outqueue.c:sctp_outq_sack() method.

Missing PSNs are handled using the psnmap structure at the receiver. The psnmap structure is defined in tsnmap.h and associated functions to handle the missing reports are defined in tsnmap.c.

While delivering the packets to the upper layer, there has to be a mapping between the TSNs and the PSNs received from different paths. We have created a binary map for each path at the receiver, which stores the one to one mapping between PSN and the TSN. This also helps in constructing the SACK structure and correctly increases the missing count at the receiver.

The path selection to transmit or retransmit data is being implemented in outqueue.c:SelectTransport(). This function returns the path identifier depending upon the strategy implemented.

```
....
else  if (iSelect == 2) //Tx − CWND
{
      list_for_each_entry(my_transport, my_transport_list, transports){
          addr = &my_transport− > ipaddr;
          dst_port = ntohs (addr− > v4.sin_port) ;
          if((my_transport! = NULL) &&
          (my_transport! = asoc− > peer.last_sent_to) &&
          (my_transport− > cwnd >= asoc− > peer.last_sent_to− > cwnd)){
              return  my_transport;
          }
}
....
```

The above code snippet shows the sample implementation of *Tx-CWND* strategy which selects the path based on the congestion window.

MPSCTP related variables are declared in sctp.h and linux/sctp.h. Flow charts for the implementation of MPSCTP algorithms are given at the end of this chapter (Figures 12-14).

6.3 Heuristic for Path Selection Policy

While transmitting data on multiple paths, path selection is a major problem. MPSCTP transmits the acknowledgement on the same path as the one on which the data is being received. This reduces the complexity at the receiver side because it does not have to know about the state of the path to which it is transmitting the acknowledgement and also ensures proper RTT calcula-

tions in case of dissimilar path statistics. It has been demonstrated in literature that *RTX-CWND* retransmission strategy performs best among all the retransmission strategies [31]. However, retransmitting packets other than on its original path requires higher processing at the sender side. To reduce this increased processing we propose and implement *Tx-CWND* strategy. According to *Tx-CWND*, we always transmit the packet on the path with the largest *cwnd*. However, we do not change the path for retransmission and the packet is retransmitted on the same path as the one on which it was originally transmitted. We show that this approach performs very close to the *RTX-CWND* with greatly reduced implementation complexity provided the path statistics do not change within one Retransmission Time Out (RTO) interval. We have measured the performance of our heuristic using *ns-2* based simulations. For our tests we have considered the network scenario as shown in Figure 6.1. The packet error probability of path–1 has been fixed at 1% while that of path–2 is varied from 1% to 10%. The bandwidths of both the paths are as shown in Figure 6.1. The source (S) runs an *ftp* application and transmits a 60 MB file to the destination.

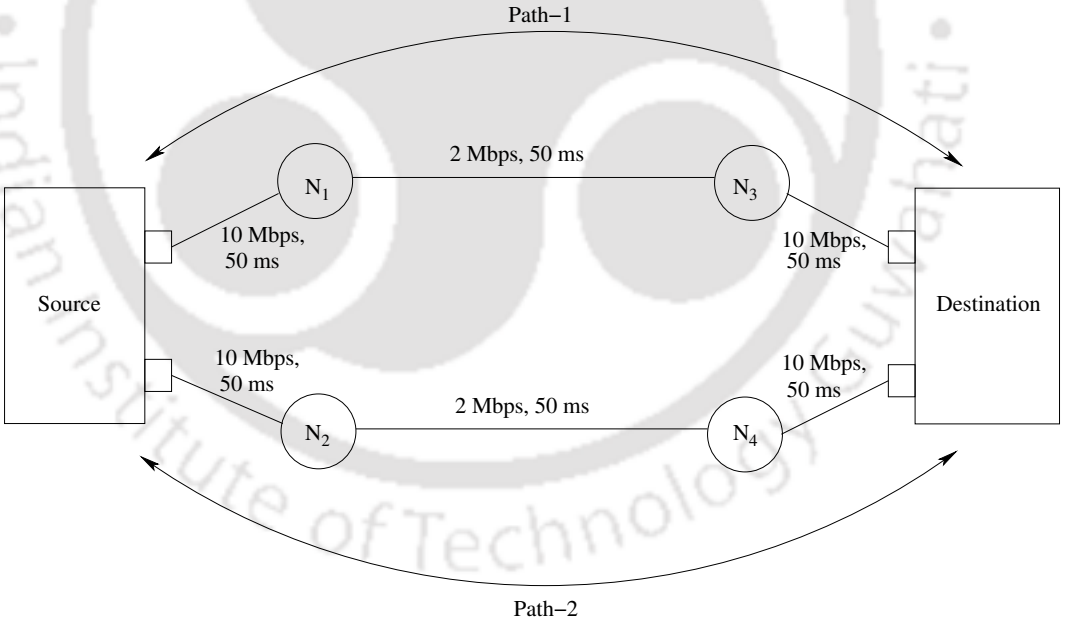


Figure 6.1: Network Layout

The average time to transmit this file is shown in Figure 6.2. The results are presented for both the *RTX-CWND* and *Tx-CWND* strategies. For smaller probabilities of error, *Tx-CWND* experiences much more packet reordering than the *RTX-CWND* strategy. This is because, while *RTX-CWND* keeps transmitting data on a path until its cwnd is completely exhausted, *Tx-CWND*

always transmits packet on the path with the higher congestion window (i.e. more network resources). It may also be noted that at higher probabilities of error *Tx-CWND* performs almost as good as *RTX-CWND* because *Tx-CWND* always transmits packets on the paths with the higher congestion window. Since *cwnd* is inversely proportional to the probability of error, the *Tx-CWND* strategy results in more packets being transmitted on paths with lower probabilities of error. This leads to relatively better performance when the probability of error is high. This can be seen in the normalized file transfer time results shown in Figure 6.3. Here, the file transfer times are normalized with respect to the file transfer time when probability of error is zero. The exact values for the file transfer time and the normalized file transfer time are also shown in Table 6.1. We notice that at higher probability of error *RTX-CWND* has higher normalized transfer time than *Tx-CWND* strategy (Figure 6.3). This indicates that the performance of *RTX-CWND* degrades much more rapidly than the *Tx-CWND* strategy as the probability of error increases.

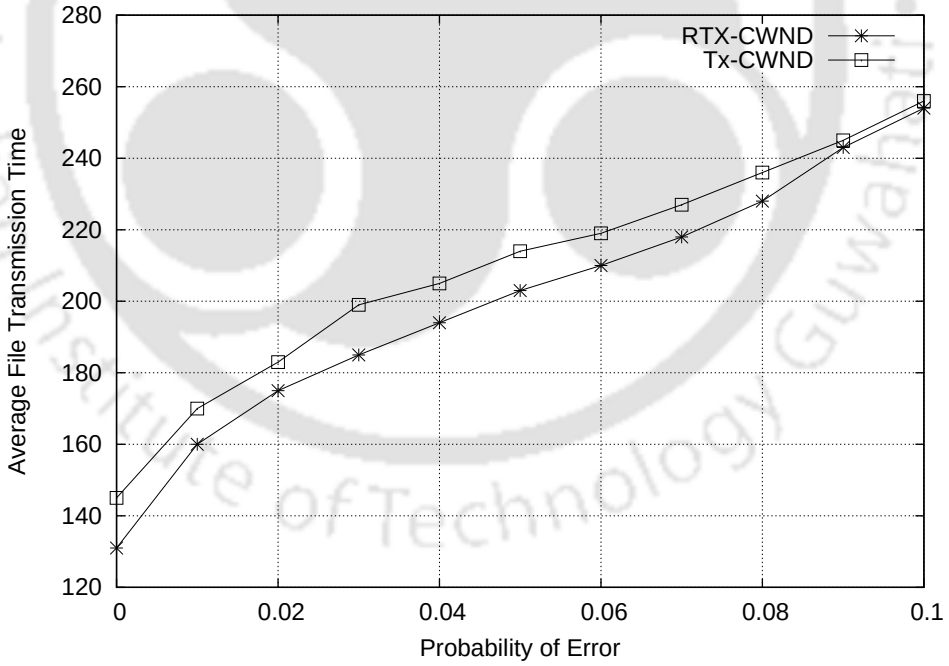


Figure 6.2: Average file transmission time for *Tx-CWND* and *RTX-CWND*

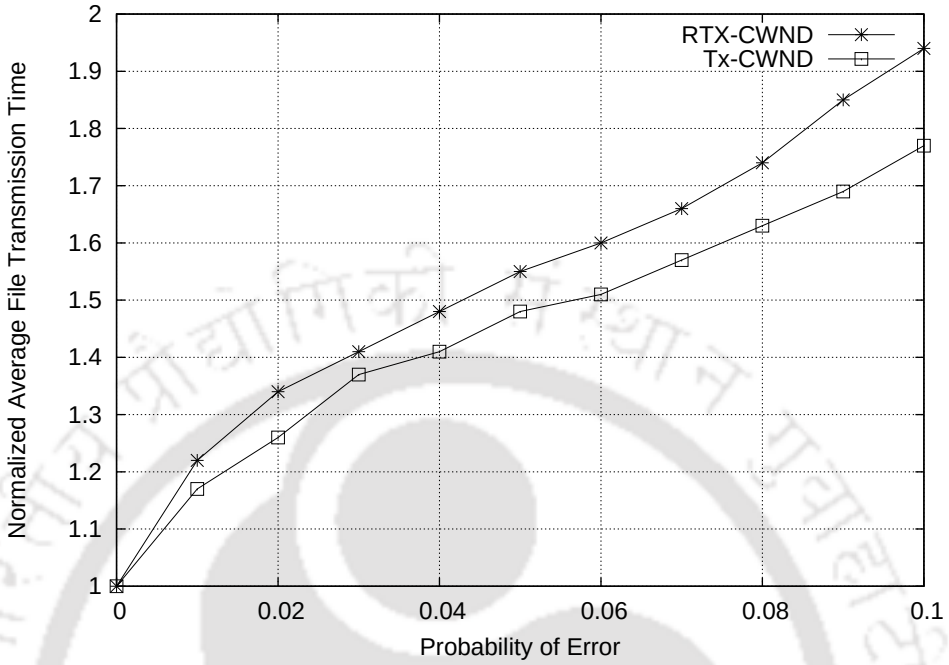


Figure 6.3: Normalized Average file transmission time for *Tx-CWND* and *RTX-CWND*

Table 6.1: Average file transfer time for *Tx-CWND* and *RTX-CWND* strategies

Probability of Error	RTX-CWND		Tx-CWND	
	Average File Transfer Time	Normalized File Transfer Time	Average File Transfer Time	Normalized File Transfer Time
0.00	131	1.00	145	1.00
0.01	160	1.22	170	1.17
0.02	175	1.34	183	1.26
0.03	185	1.41	199	1.37
0.04	194	1.48	205	1.41
0.05	203	1.55	214	1.48
0.06	210	1.60	219	1.51
0.07	218	1.66	227	1.57
0.08	228	1.74	236	1.63
0.09	243	1.85	245	1.69
0.10	254	1.94	256	1.77

6.4 Issues related to SACK Reordering

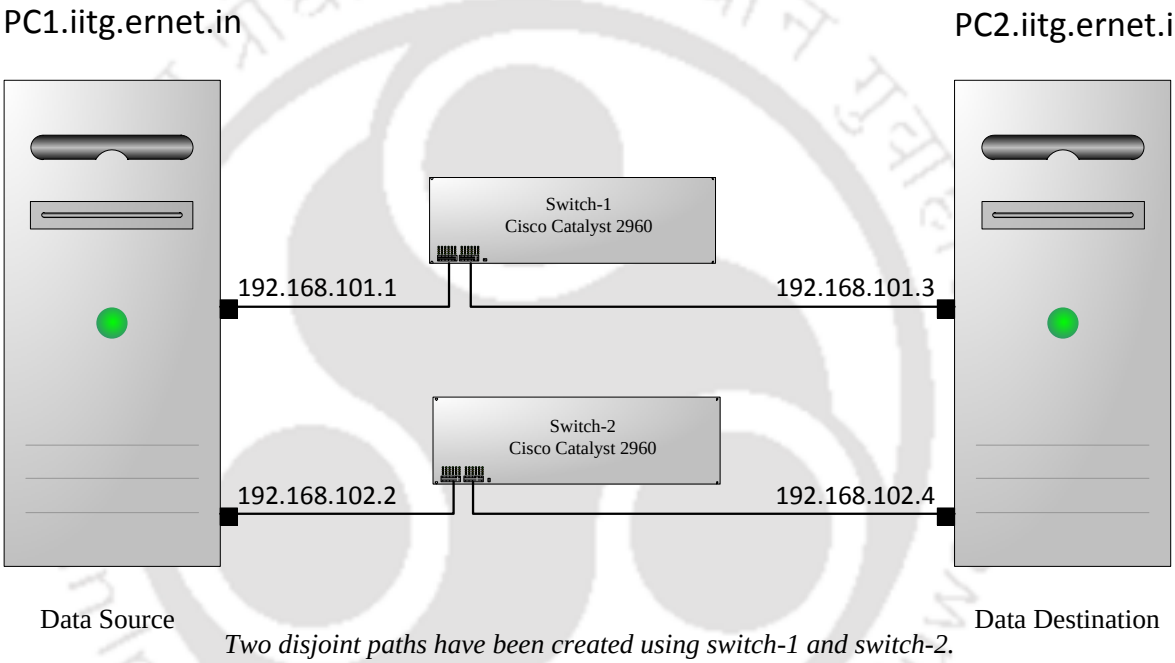
Like other transport protocols, MPSCTP may also exhibit the problem of SACK reordering. Reordered SACK causes data reneging at the sender side. The SACK reordering can happen due to differences in the path's RTT. Moreover, the network caused reordering [84] is also not uncommon and can cause SACK reordering. If a reneged chunk is acknowledged again it can increase the missing count for a missing chunk. This can lead to improper missing count at the sender and causes fast retransmission. However, the above problem is resolved by not considering the re-acknowledged chunk as a suitable candidate for increasing the missing count. Since elaborate study of network caused reordering is not the main focus of this thesis, this is not pursued further.

6.5 Performance Evaluation

We have tested the performance of our MPSCTP implementation in an emulated network using *dummynet* as well as in a real network i.e. the Local Area Network (LAN) of IIT Guwahati.

6.5.1 Results Using Dummynet:

We have considered two dual homed machines $PC1$ and $PC2$ connected as shown in Figure 6.4. The interfaces of these machines are connected through two *CISCO Catalyst 2960* switches to create two disjoint paths between the source and the destination.The delay and loss rate of each path is controlled using *dummynet*. The delay for each path between the two machines is set to 50 ms. The packet drop probability of one path is fixed at 1% while the packet drop probability on the other path is varied from 1% to 10%. The bandwidth of the two paths is also controlled using *dummynet* and each of them in set to 2 Mbps (Figure 6.4).



Two disjoint paths have been created using switch-1 and switch-2. Packet Error, bandwidth and Delays of both the paths are controlled using dummynet between PC1 and PC2.

Figure 6.4: Network Diagram for MPSCTP implementation in Linux kernel

We transmit a 60 MB file using MPSCTP. The path selection policy used by MPSCTP is *Tx-CWND* with retransmissions made on the same path as the original path of transmission. We use a receiver buffer of 64 KB, as this is the most commonly used value in most practical scenarios.

Table 6.2: Average file transfer time in Emulated Network using *dummynet* and Simulated Network using *ns-2*

Probability of Error	dummynet		*ns-2*	
	Average File Transfer Time	Normalized File Transfer Time	Average File Transfer Time	Normalized File Transfer Time
0.00	160	1.00	145	1.00
0.01	181	1.13	170	1.17
0.02	193	1.21	183	1.26
0.03	213	1.33	199	1.37
0.04	225	1.41	205	1.41
0.05	232	1.45	214	1.48
0.06	242	1.51	219	1.51
0.07	255	1.59	227	1.57
0.08	266	1.66	236	1.63
0.09	279	1.74	245	1.69
0.10	285	1.78	256	1.77

Figure 6.5 shows the normalized file transfer time for transferring 60 MB of file using MPSCTP. The corresponding average file transfer time values are plotted in Figure 6.6. The exact values for the transfer time and the normalized transfer time are shown in Table 6.2. We have averaged out each curve for 11 runs. The results in Figure 6.5 show that delay trends for MPSCTP implementation is similar to that of the simulation using *ns-2*. However, on comparing the absolute values in Table 6.2 we find that the delay values with kernel implementation are higher than the simulation values. This difference is primarily due to the deterministic nature of *dummynet*. A 10% packet drop probability drops exactly 10% of the packets in *dummynet* unlike in *ns-2* which drops the packet statistically. We have also plotted a sample evolution of *cwnd* in the emulated network with 10% packet error probability for path–2 in Figure 6.7.

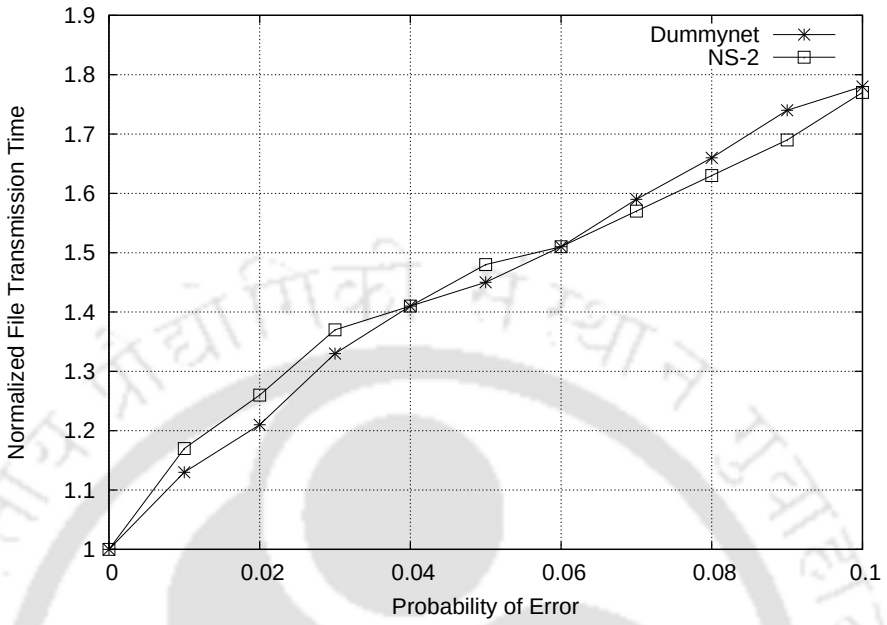


Figure 6.5: Probability of Error vs Normalized File Transfer Time

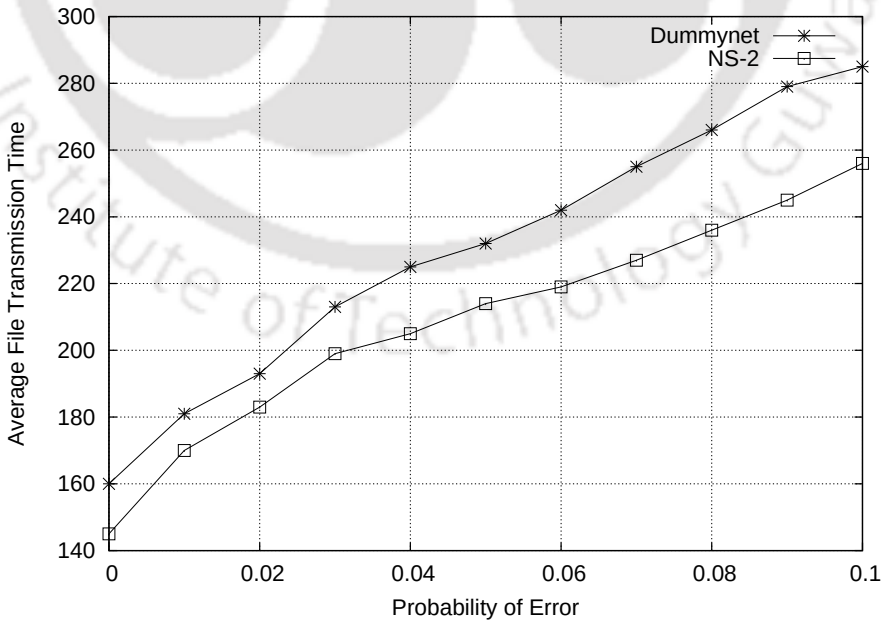


Figure 6.6: Probability of Error vs Average File Transfer Time

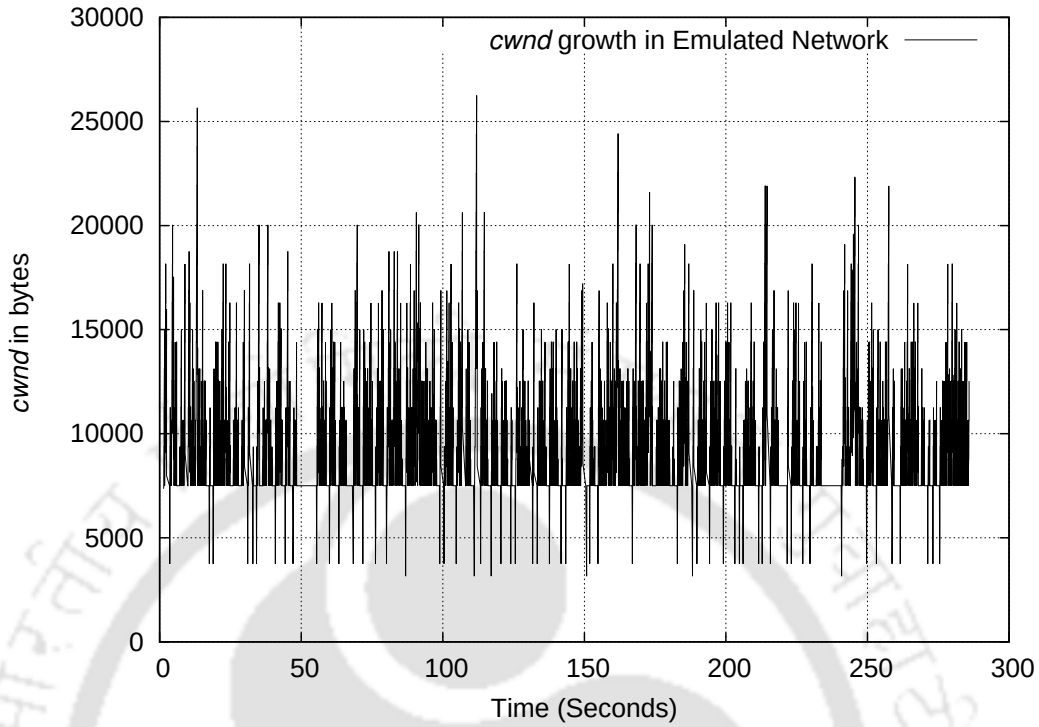


Figure 6.7: *cwnd* evolution of path–2 in emulated network for 10% probability of error

6.5.2 Results in IIT Guwahati LAN

We have measured the performance of MPSCTP implementation in the IIT Guwahati LAN environment. Two dual homed machines *HFPC* and *SSPC* are used with one placed in the High Frequency Lab and the other in the System Simulation Lab of the EEE Department in IIT Guwahati (Figure 6.8). Using *dummynet*, we have limited the bandwidth of each path to the maximum value of 2 Mbps, minimum value of delay is set to 50 ms, and minimum packet loss probability is set to 1% for path–1 and varied from 1% to 10% for path–2. Note that here the two paths are not truly disjoint as there may be some intermediate shared links on each path.

Table 6.3 gives the absolute values of delay and the corresponding normalized values of delay with respect to the delay at zero probability of error for the above setup. We ran our tests at four different times of the day to average out the effect of background traffic. In each case, the same test is run 11 times and the average results are tabulated in Table 6.3.

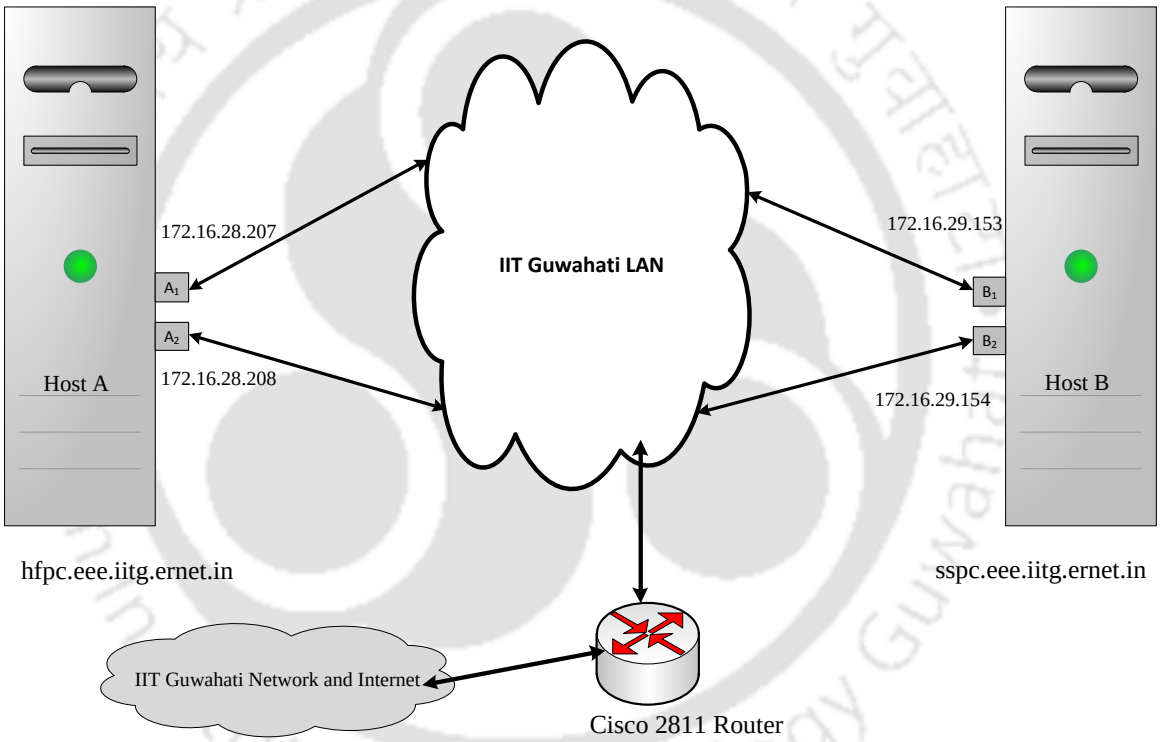


Figure 6.8: Network Layout of IIT Guwahati LAN environment

As the results show, the transfer time values in IIT Guwahati LAN are significantly higher than those in an emulated network. This is due to the various other background traffic present in the LAN. The average transfer times for different probabilities of packet error are shown in Figure 6.9. The corresponding normalized values are also shown in Figure 6.10. We observe that the transfer time increases rapidly with increasing probabilities of error. This is due to the various other background traffic in the network. Hence, the per packet recovery time also increases at higher rate in this case than in case of an emulated network. This phenomenon also explains the crossover observed in the Figure 6.10.

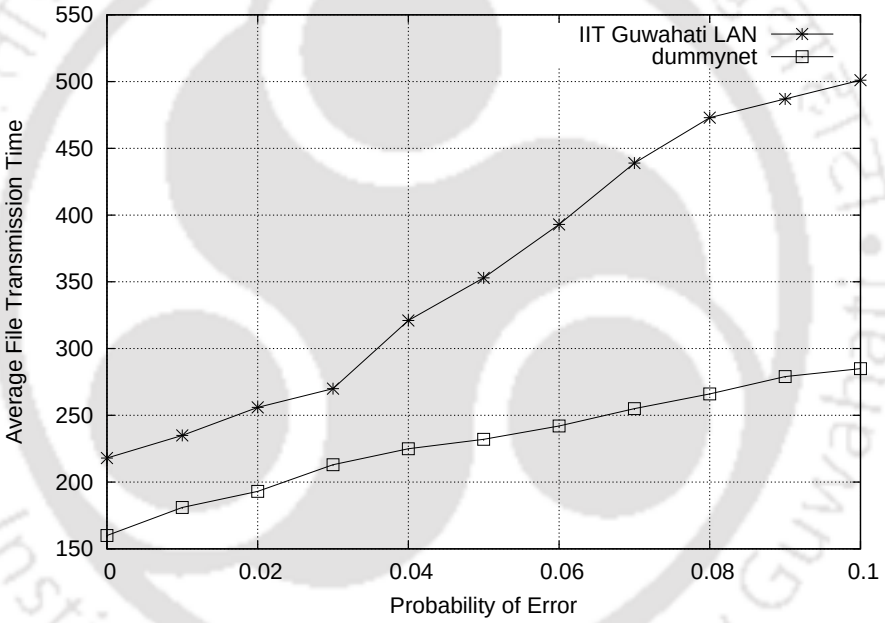


Figure 6.9: Probability of Error vs Average File Transfer Time in IIT Guwahati LAN

A sample evolution of cwnd growth in IIT Guwahati LAN is shown in Figure 6.11. This is plotted for the minimum packet error probability of 10%. It is clear from the *cwnd* plot in Figure 6.11 that it encounters more packet losses than in the scenario where an emulated network is used (Figure 6.7). Moreover, the *cwnd* grows to much higher values than in an emulated network. This also contributes to the higher transmission time observed in the LAN environment as compared to the results for the emulated network.

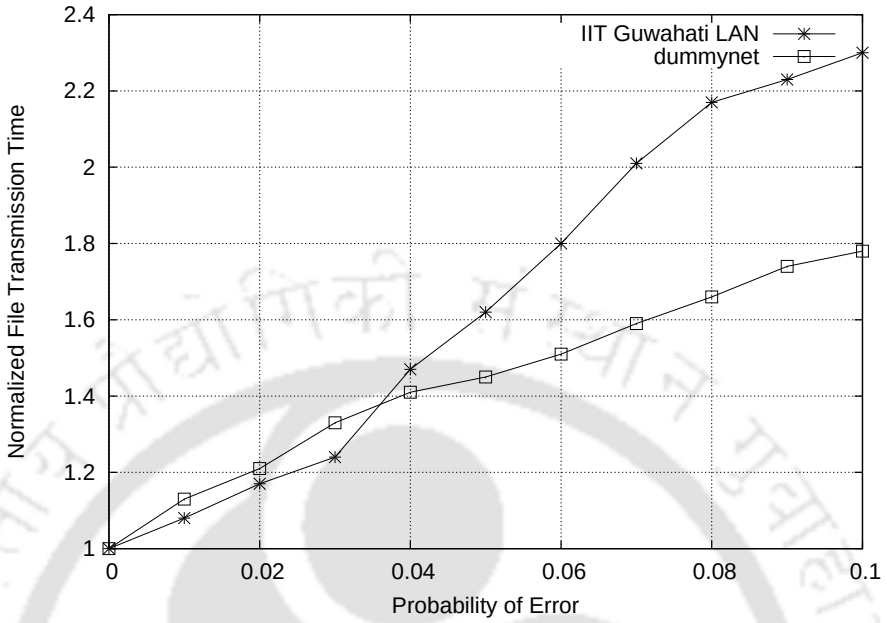


Figure 6.10: Probability of Error vs Normalized File Transfer Time in IIT Guwahati LAN

Table 6.3: Average file transfer time in Emulated Network using *dummynet* and IIT Guwahati LAN

Probability of Error	IIT Guwahati LAN		dummynet	
	Average File Transfer Time	Normalized File Transfer Time	Average File Transfer Time	Normalized File Transfer Time
0.00	218	1.00	160	1.00
0.01	235	1.08	181	1.13
0.02	256	1.17	193	1.21
0.03	270	1.24	213	1.33
0.04	321	1.47	225	1.41
0.05	353	1.62	232	1.45
0.06	393	1.80	242	1.51
0.07	439	2.01	255	1.59
0.08	473	2.17	266	1.66
0.09	487	2.23	279	1.74
0.10	501	2.30	285	1.78

6.6 Conclusion

In this chapter we have shown the results for a sample implementation of MPSCTP in the Linux kernel. We have compared those results with that of the simulation results with *ns-2* in Chapter

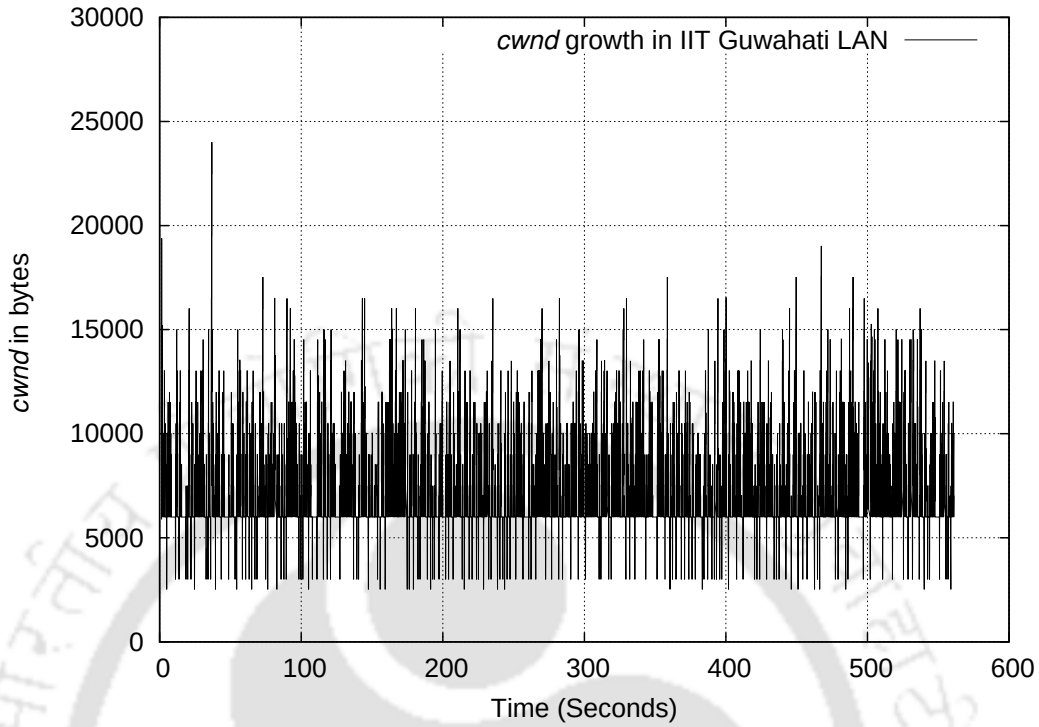
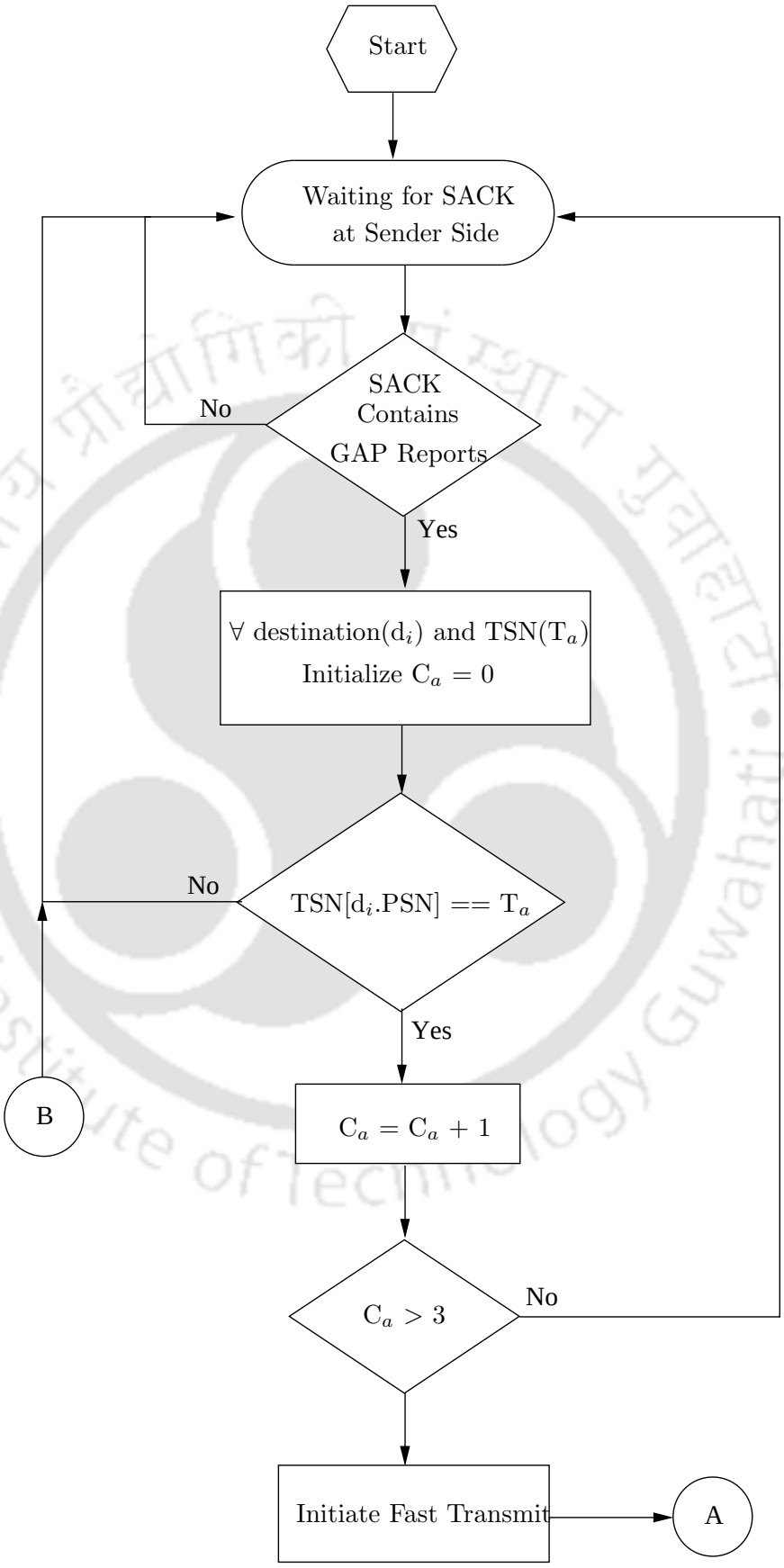


Figure 6.11: *cwnd* evolution of path–2 in IIT Guwahati LAN for 10% probability of error

4. We found that the results in this chapter are consistent with the results of MPSCTP in Chapter 4. Though we choose to implement MPSCTP in Linux, we believe that the same protocol can be implemented in other operating systems as well.

We have proposed a *Tx-CWND* strategy and compared the results with the *RTX-CWND* strategy proposed in the literature. We found that at lower probabilities of error, the average file transfer time with the *RTX-CWND* strategy is slightly lower than that of the *Tx-CWND* strategy. This is due to the greater reordering caused by the *Tx-CWND* strategy. However, at higher probabilities of error, both have almost similar file transmission time. This indicates the performance of the *RTX-CWND* degrades faster than the *Tx-CWND* strategy with increasing packet error probabilities.

The original SCTP congestion control algorithm is based upon the TCP congestion control approach. Current implementation of MPSCTP also inherits the same congestion control from the SCTP module. This implementation can be further extended in future to support flow division as well as BERP congestion control. The implementation of flow division and BERP algorithms is expected to further improve the performance of the MPSCTP protocol.


Start
Waiting for SACK at Sender Side
SACK Contains GAP Reports
No
Yes
∀ destination(d_i) and TSN(T_a) Initialize $C_a = 0$
TSN[d_i.PSN] == T_a
No
Yes
B
$C_a = C_a + 1$
$C_a > 3$
No
Initiate Fast Transmit
A

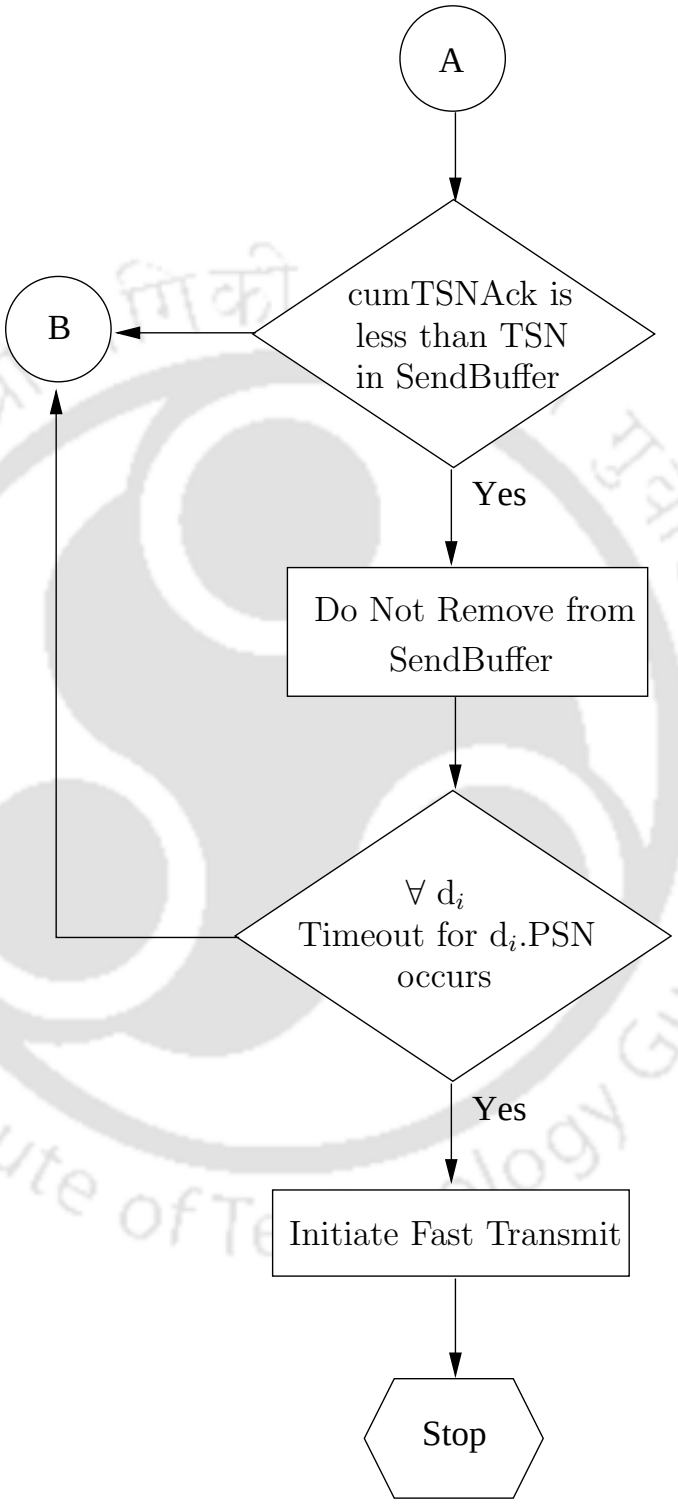


Figure 6.12: Flow Chart to Handle Fast Retransmission

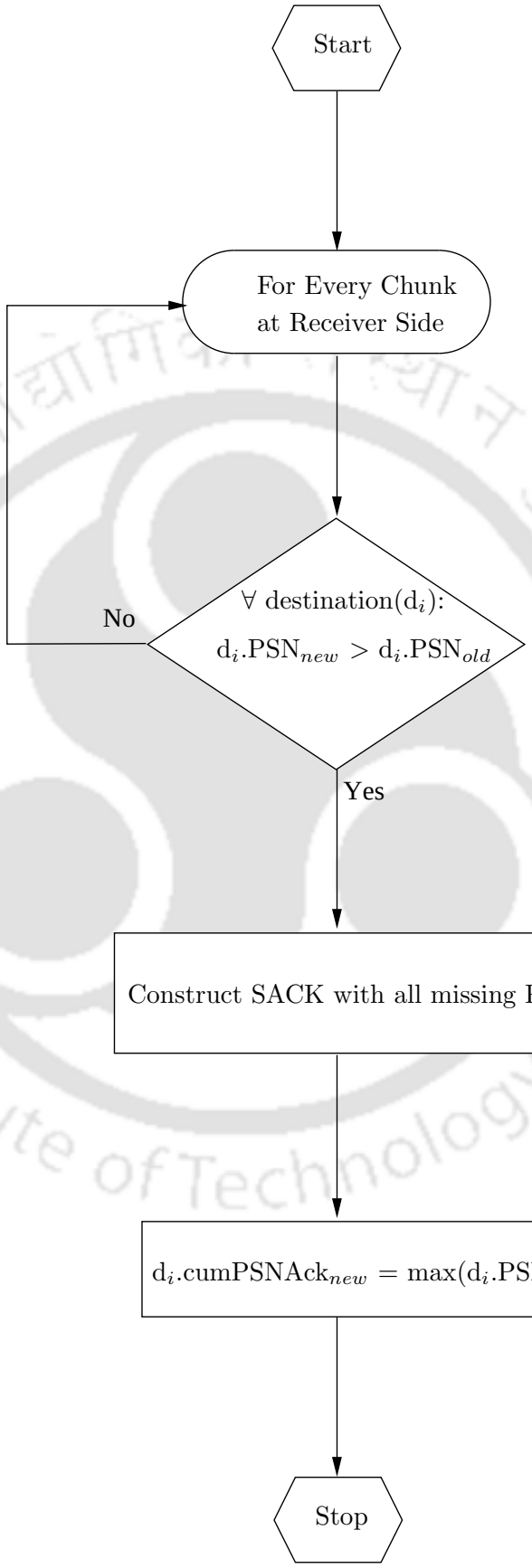


Figure 6.13: Flow Chart to handle cumulative acknowledgement update

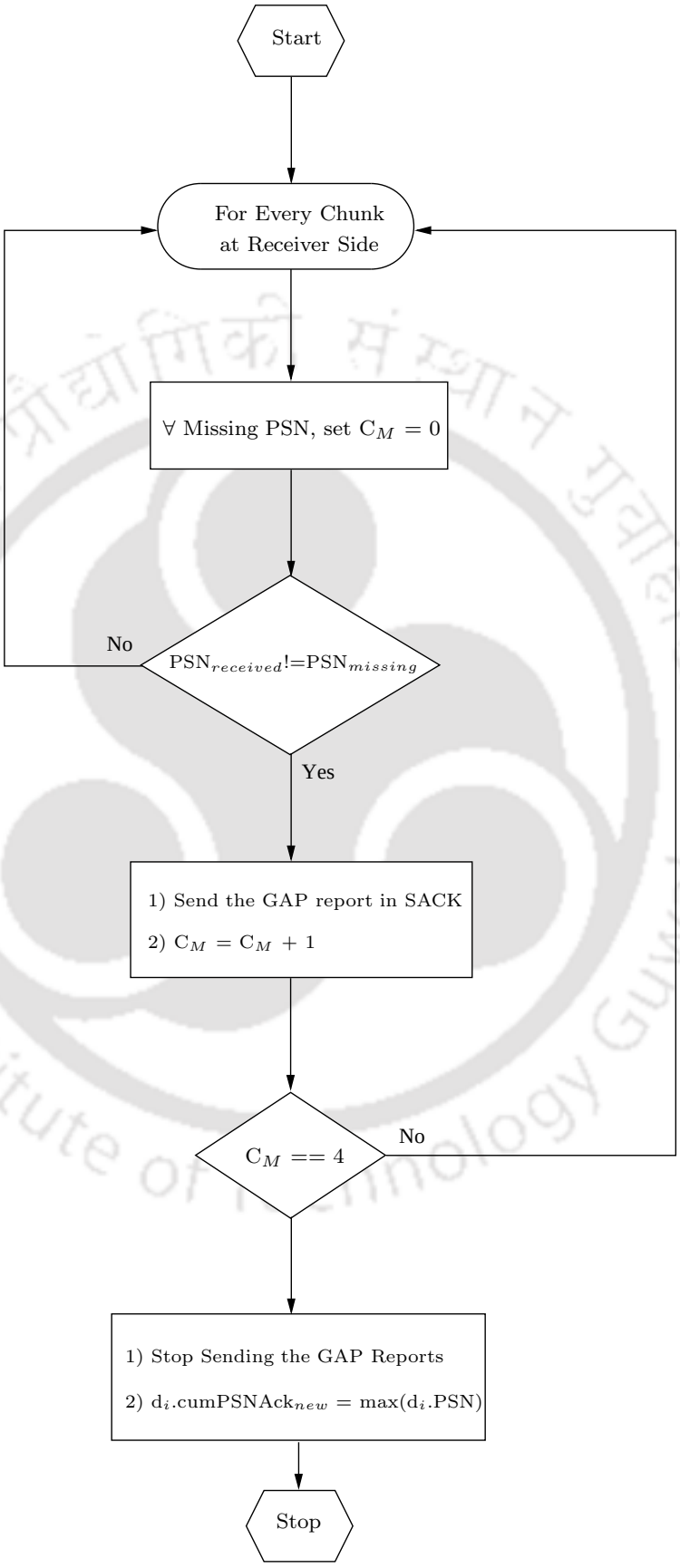


Figure 6.14: Flow Chart to handle Gap Reports for missing PSNs

7

Discussion and Future Work

Contents

In this thesis, we propose MPSCTP, a multipath version of SCTP and studied its performance. This thesis also discusses flow division at the source for multipath usage in the network. This chapter contains the summary of the thesis and also discusses some of the possible future directions of research. The main contributions of this thesis are summarized in Section 7.1 and some ideas for future work are outlined in Section 7.2.

7.1 Summary of Contributions and Discussions

In this section, we briefly summarize and discuss our contributions and findings during this thesis work.

A. Flow Division at Source:

In this part of work, we investigated flow division at source. Flow division at source is an interesting optimization problem which addresses the issue of how the flow should be divided on multiple paths such that some end-to-end objective is achieved. The optimum flow division must consider the nature of end applications as well. Three different optimizations have been proposed for two different classes of applications. These optimizations help in finding possible guidelines on how the flow should be divided on multiple candidate paths. Their performances have also been compared for end-to-end and link-by-link error recovery mechanisms and the latter is found to perform better.

B. MPSCTP Protocol

In this thesis we proposed Multipath SCTP (MPSCTP) protocol, a novel multipath variant of SCTP protocol. The proposed protocol uses two levels of sequence numbering to isolate the reordering caused by multipath usage from the packet loss in the network. This design gives better performance than Concurrent Multipath Transfer (CMT) protocol. We have also observed that the proposed protocol also generally performs better than the CMT protocol for real time data transmission when operating under similar conditions.

C. BERP Congestion Control

MPSCTP in its original form assumes that the paths between source and destination are disjoint. All multipath variants of SCTP like MPSCTP and CMT inherits their congestion control algorithm from SCTP and SCTP inherits its congestion control from TCP. On shared links,

such algorithms do not behave fairly with other fellow flows. This issue has been addressed through Bandwidth Estimation based Resource Pooling (BERP) congestion control. The results obtained indicate that this approach improves the performance of multipath protocols on shared link as well as in high loss scenarios. BERP reduces the congestion window based upon the bandwidth estimates of the path and takes care of shared paths while improving the overall performance of the protocol, especially in high loss scenarios. This algorithm is independent of the underlying protocol and can be applied suitably to any multipath transport layer protocol.

D. Heuristics for Flow Division Optimization

In this thesis we have proposed several heuristics for flow division optimizations in an Internet like environment. We have proposed implementations for Min-Max Optimization and Delay Insensitive Optimization. The exact implementation of these algorithms requires complete network statistics to be known at the source. This is quite time consuming as well as computationally involved. Hence, we have proposed suitable heuristics which approximates these results based upon end-to-end measurements. The Min-Max optimization is demonstrated to be approximated by BERP algorithm. We have proposed to implement Delay Insensitive Optimization by suitably modifying the congestion control algorithm. This modification can be seamlessly applied to any TCP like congestion control algorithm as well.

E. MPSCTP implementation in Linux kernel

We have implemented MPSCTP in the Linux kernel version 2.6.37 bundled with openSUSE 11.4. Though we have considered Linux for our implementation, this can be seamlessly implemented on other operating systems as well. This Linux kernel is distributed with *lksctp*, a loadable module for SCTP. We have implemented MPSCTP in *lksctp* and the results are compared with the simulation results. The implementation results are in close agreement with the simulations results. We have tested MPSCTP implementation in an emulated network as well as in the IITG LAN environment. We have also proposed the *Tx-CWND* flow division strategy. This strategy along with retransmissions on the same path performs very much like the *RTX-CWND* retransmission strategy and is much easier to implement in the Linux kernel.

7.2 Suggestions for Future Work

In this thesis we have addressed some of the basic issues related to multipath transport and flow division for multihomed hosts. For practical implementation of such schemes in Internet like environments, the works reported in this thesis can be extended in several directions. Some of them are listed below.

i) In this work, we have not tested the applicability of the proposed optimization algorithms in real networks. These algorithms can be implemented and tested in the Internet like environments for different applications like Skype and YouTube and their performance can be investigated.

ii) We proposed heuristics only for the Min-Max Optimization and Delay Insensitive Optimization. A suitable heuristic for the Weighted Average Optimization can also be developed.

iii) An implementation of these heuristics over the real Internet can be done to study their performance.

iv) Development of improved heuristics for closer implementation of the proposed optimization may be an interesting extension of this work.

v) Performance of MPSCTP with BERP congestion control has been studied only in a simulated environment. MPSCTP-BERP has been found to outperform both CMT as well as MPTCP. However, implementation of BERP algorithm along with MPSCTP in the Linux kernel with more elaborate tests and study in an Internet like environments may be carried out.

vi) The proposed MPSCTP augments SCTP which has the potential to replace TCP and UDP. Therefore, scope exists in devising future migration plans for wider acceptance of SCTP and MPSCTP in Internet like environments.

vii) Network Coding is a relatively new area of research. Use of Network Coding in multipath scenario is largely unexplored. We envisage that use of Network Coding in multipath scenario can be an interesting problem.

viii) Data offloading has been of major interest for service providers to improve the performance of the network. We believe that it may be possible to use multipath protocols for seamless

data offloading and in turn improve the network performance.

Bibliography

[1] T. Hacker, B. Athey, and B. Noble, "The end-to-end performance effects of parallel tcp sockets on a lossy wide-area network," in *Parallel and Distributed Processing Symposium., Proceedings International, IPDPS 2002, Abstracts and CD-ROM*, 2002, pp. 434 – 443. doi: 10.1109/IPDPS.2002.1015527

[2] H. Sivakumar, S. Bailey, and R. Grossman, "Psockets: The case for application-level network striping for data intensive applications using high speed wide area networks," in *Supercomputing, ACM/IEEE 2000 Conference*, nov 2000, pp. 1–8. doi: 10.1109/SC.2000.10040

[3] S. H. Low and D. E. Lapsley, "Optimization flow control—i: basic algorithm and convergence," *IEEE/ACM Trans. Netw.*, vol. 7, no. 6, pp. 861–874, 1999.

[4] S. Low, "Optimization flow control with on-line measurement or multiple paths," in *In Proceedings of the ITC*, 1999, pp. 237–249.

[5] S. Athuraliya and S. Low, "Optimization flow control with newton-like algorithm," in *Global Telecommunications Conference, 1999. GLOBECOM '99*, vol. 2, 1999, pp. 1264 –1268. doi: 10.1109/GLOCOM.1999.829974

[6] R. Stewart, Q. Xie, K. Morneault, C. Sharp, H. Schwarzbauer, T. Taylor, I. Rytina, M. Kalla, L. Zhang, and V. Paxson, "Stream Control Transmission Protocol," RFC 2960 (Proposed Standard), Internet Engineering Task Force, Oct. 2000, obsoleted by RFC 4960, updated by RFC 3309. [Online]. Available: http://www.ietf.org/rfc/rfc2960.txt

[7] L. Ong, I. Rytina, M. Garcia, H. Schwarzbauer, L. Coene, H. Lin, I. Juhasz, M. Holdrege, and C. Sharp, "Framework Architecture for Signaling Transport," RFC 2719 (Informational), Internet Engineering Task Force, Oct. 1999. [Online]. Available: http://www.ietf.org/rfc/rfc2719.txt

[8] J. Loughney, M. Tuexen, and J. Pastor-Balbas, "Security Considerations for Signaling Transport (SIGTRAN) Protocols," RFC 3788 (Proposed Standard), Internet Engineering Task Force, Jun. 2004. [Online]. Available: http://www.ietf.org/rfc/rfc3788.txt

[9] K. Morneault, R. Dantu, G. Sidebottom, B. Bidulock, and J. Heitz, "Signaling System 7 (SS7) Message Transfer Part 2 (MTP2) - User Adaptation Layer," RFC 3331 (Proposed Standard), Internet Engineering Task Force, Sep. 2002. [Online]. Available: http://www.ietf.org/rfc/rfc3331.txt

[10] T. George, B. Bidulock, R. Dantu, H. Schwarzbauer, and K. Morneault, "Signaling System 7 (SS7) Message Transfer Part 2 (MTP2) - User Peer-to-Peer Adaptation Layer (M2PA)," RFC 4165 (Proposed Standard), Internet Engineering Task Force, Sep. 2005. [Online]. Available: http://www.ietf.org/rfc/rfc4165.txt

[11] K. Morneault and J. Pastor-Balbas, "Signaling System 7 (SS7) Message Transfer Part 3 (MTP3) - User Adaptation Layer (M3UA)," RFC 4666 (Proposed Standard), Internet Engineering Task Force, Sep. 2006. [Online]. Available: http://www.ietf.org/rfc/rfc4666.txt

[12] J. Postel, "Transmission Control Protocol," RFC 793 (Standard), Internet Engineering Task Force, Sep. 1981, updated by RFCs 1122, 3168, 6093, 6528. [Online]. Available: http://www.ietf.org/rfc/rfc793.txt

[13] E. Rathgeb, C. Hohendorf, and M. Nordhoff, "On the robustness of sctp against dos attacks," in *Convergence and Hybrid Information Technology, 2008. ICCIT '08. Third International Conference on*, vol. 2, nov 2008, pp. 1144 –1149. doi: 10.1109/ICCIT.2008.196

[14] M. Scharf and S. Kiesel, "Head-of-line blocking in tcp and sctp: Analysis and measurements," in *Global Telecommunications Conference, 2006. GLOBECOM '06. IEEE*, nov 2006, pp. 1 –5. doi: 10.1109/GLOCOM.2006.333

[15] K.-J. Grinnemo, T. Andersson, and A. Brunstrom, "Performance benefits of avoiding head-of-line blocking in sctp," in *Autonomic and Autonomous Systems and International Conference on Networking and Services, 2005. ICAS-ICNS 2005. Joint International Conference on*, oct 2005, pp. 1–8. doi: 10.1109/ICAS-ICNS.2005.73

[16] "Connecting to multiple ieee 802.11 networks with one wifi card," http://research.microsoft.com/en-us/um/redmond/projects/virtualwifi/.

[17] J. Iyengar, P. Amer, and R. Stewart, "Concurrent multipath transfer using sctp multihoming over independent end-to-end paths," *Networking, IEEE/ACM Transactions on*, vol. 14, no. 5, pp. 951 – 964, oct 2006.

[18] "Sctp for the linux kernel," http://lksctp.sourceforge.net.

[19] T. Itu, "G.1010: End-user multimedia QoS categories," ITU, Tech. Rep., 2001.

[20] J. Postel, "User Datagram Protocol," RFC 768 (Standard), Internet Engineering Task Force, Aug. 1980. [Online]. Available: http://www.ietf.org/rfc/rfc768.txt

[21] T. Hacker, B. Noble, and B. Athey, "Improving throughput and maintaining fairness using parallel tcp," in *INFOCOM 2004. Twenty-third AnnualJoint Conference of the IEEE Computer and Communications Societies*, vol. 4, mar 2004, pp. 2480 – 2489 vol.4. doi: 10.1109/INFCOM.2004.1354669

[22] M. Handley, E. Rescorla, and IAB, "Internet Denial-of-Service Considerations," RFC 4732 (Informational), Internet Engineering Task Force, Dec. 2006. [Online]. Available: http://www.ietf.org/rfc/rfc4732.txt

[23] "Multi network datagram transmission protocol," http://tools.ietf.org/html/draft-ietf-sigtran-mdtp-06.

[24] R. Stewart, "Stream Control Transmission Protocol," RFC 4960 (Proposed Standard), Internet Engineering Task Force, Sep. 2007, updated by RFCs 6096, 6335. [Online]. Available: http://www.ietf.org/rfc/rfc4960.txt

[25] S. Fu and M. Atiquzzaman, "Sctp: state of the art in research, products, and technical challenges," *Communications Magazine, IEEE*, vol. 42, no. 4, pp. 64 – 76, apr 2004.

[26] P. Natarajan, F. Baker, P. Amer, and J. Leighton, "Sctp: What, why, and how," *Internet Computing, IEEE*, vol. 13, no. 5, pp. 81 –85, sep-oct 2009.

[27] S. Fu and M. Atiquzzaman, "Performance modeling of sctp multihoming," in *Global Telecommunications Conference, 2005. GLOBECOM '05. IEEE*, vol. 2, nov-dec 2005, pp. 786–791. doi: 10.1109/GLOCOM.2005.1577747

[28] A. Abd El Al, T. Saadawi, and M. Lee, "Bandwidth aggregation in stream control transmission protocol," in *Computers and Communications, 2004. Proceedings. ISCC 2004. Ninth International Symposium on*, vol. 2, 2004, pp. 975 – 980. doi: 10.1109/ISCC.2004.1358667

[29] G. Ye, T. Saadawi, and M. Lee, "Ipcc-sctp: an enhancement to the standard sctp to support multihoming efficiently," in *Performance, Computing, and Communications, 2004 IEEE International Conference on*, 2004, pp. 523 – 530. doi: 10.1109/PCCC.2004.1395081

[30] J. Iyengar, P. Amer, and R. Stewart, "Receive buffer blocking in concurrent multipath transfer," in *Global Telecommunications Conference, 2005. GLOBECOM '05. IEEE*, vol. 1, 2005, pp. 121–125. doi: 10.1109/GLOCOM.2005.1577365

[31] ——, "Retransmission policies for concurrent multipath transfer using sctp multihoming," in *Proceedings of 12th IEEE International Conference on Networks, 2004 (ICON 2004).*, vol. 2, nov 2004, pp. 713 – 719. doi: 10.1109/ICON.2004.1409269

[32] A. L. Caro, P. D. Amer, and R. R. Stewart, "Retransmission policies for multihomed transport protocols," *Computer Communications*, vol. 29, no. 10, pp. 1798 – 1810, 2006, monitoring and measurements of IP Networks. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0140366405003944

[33] J. Liao, J. Wang, and X. Zhu, "cmpsctp: An extension of sctp to support concurrent multi-path transfer," in *Communications, 2008. ICC '08. IEEE International Conference on*, 2008, pp. 5762 – 5766. doi: 10.1109/ICC.2008.1078

[34] I. Joe and S. Yan, "Sctp throughput improvement with best load sharing based on multihoming," in *INC, IMS and IDC, 2009. NCM '09. Fifth International Joint Conference on*, 25-27 2009, pp. 138 –142. doi: 10.1109/NCM.2009.170

[35] Y. Yuan, Z. Zhang, J. Li, J. Shi, J. Zhou, G. Fang, and E. Dutkiewicz, "Extension of sctp for concurrent multi-path transfer with parallel subflows," in *Wireless Communications and Networking Conference (WCNC), 2010 IEEE*, apr 2010, pp. 1 –6. doi: 10.1109/WCNC.2010.5506559

[36] A. Ford, C. Raiciu, M. Handley, S. Barre, and J. Iyengar, "Architectural Guidelines for Multipath TCP Development," RFC 6182 (Informational), Internet Engineering Task Force, Mar. 2011. [Online]. Available: http://www.ietf.org/rfc/rfc6182.txt

[37] C. Raiciu, D. Wischik, and M. Handle, "Practical congestion control for multipath transport protocols," University College London, UK, Tech. Rep., 2009.

[38] R. Krishnan and J. Silvester, "Choice of allocation granularity in multipath source routing schemes," in *INFOCOM '93. Proceedings.Twelfth Annual Joint Conference of the IEEE Computer and Communications Societies. Networking: Foundation for the Future. IEEE*, vol. 1, 1993, pp. 322 –329. doi: 10.1109/INFCOM.1993.253345

[39] Y. S. M. D. Lei Wang, Lianfang Zhang, "Multipath source routing in wireless ad hoc networks," in *Canadian Conference on Electrical and Computer Engineering*, vol. 1, 2000, pp. 479–483.

[40] L. Zhang, Z. Zhao, Y. Shu, L. Wang, and O. Yang, "Load balancing of multipath source routing in ad hoc networks," in *Communications, 2002. ICC 2002. IEEE International Conference on*, vol. 5, 2002, pp. 3197 – 3201. doi: 10.1109/ICC.2002.997425

[41] K. Lee, A. Toguyeni, and A. Rahmani, "Hybrid multipath routing algorithms for load balancing in mpls based ip network," in *Advanced Information Networking and Applications, 2006. AINA 2006. 20th International Conference on*, vol. 1, 2006, pp. 18–20. doi: 10.1109/AINA.2006.194

[42] S. F. Ossama Younis, "Flowmate: Scalable on-line flow clustering," *IEEE/ACM Transactions on Networking (TON)*, vol. 13, pp. 288–301, 2005.

[43] Y. Kitatsuji, S. Katsuno, M. Tsuru, T. Takine, and Y. Oie, "On flow distribution over multiple paths based on traffic characteristics," *Lecture Notes in Computer Science, Springer Berlin / Heidelberg*, vol. 3961/2006, pp. 483–492, nov 2006.

[44] C.-Q. Yang and A. Reddy, "A taxonomy for congestion control algorithms in packet switching networks," *Network, IEEE*, vol. 9, no. 4, pp. 34 –45, jul-aug 1995.

[45] S. Keshav, "A control-theoretic approach to flow control," *SIGCOMM Comput. Commun. Rev.*, vol. 25, no. 1, pp. 188–201, jan 1995. [Online]. Available: http://doi.acm.org/10.1145/205447.205463

[46] S. Golestani, "Congestion-free communication in high-speed packet networks," *Communications, IEEE Transactions on*, vol. 39, no. 12, pp. 1802 –1812, dec 1991.

[47] P. Mishra, H. Kanakia, and S. Tripathi, "On hop-by-hop rate-based congestion control," *Networking, IEEE/ACM Transactions on*, vol. 4, no. 2, pp. 224 –239, apr 1996.

[48] A. Demers, S. Keshav, and S. Shenker, "Analysis and simulation of a fair queueing algorithm," *SIGCOMM Comput. Commun. Rev.*, vol. 19, no. 4, pp. 1–12, aug 1989. [Online]. Available: http://doi.acm.org/10.1145/75247.75248

[49] D. Davies, "The control of congestion in packet-switching networks," *Communications, IEEE Transactions on*, vol. 20, no. 3, pp. 546 – 550, jun 1972.

[50] D. Comer and R. Yavatkar, "A rate-based congestion avoidance and control scheme for packet switched networks," in *Distributed Computing Systems, 1990. Proceedings., 10th International Conference on*, may-jun 1990, pp. 390 –397. doi: 10.1109/ICDCS.1990.89307

[51] M. Allman, V. Paxson, and W. Stevens, "TCP Congestion Control," RFC 2581 (Proposed Standard), Internet Engineering Task Force, Apr. 1999, obsoleted by RFC 5681, updated by RFC 3390. [Online]. Available: http://www.ietf.org/rfc/rfc2581.txt

[52] L. Brakmo and L. Peterson, "Tcp vegas: end to end congestion avoidance on a global internet," *Selected Areas in Communications, IEEE Journal on*, vol. 13, no. 8, pp. 1465 –1480, oct 1995.

[53] S. Floyd, T. Henderson, and A. Gurtov, "The NewReno Modification to TCP's Fast Recovery Algorithm," RFC 3782 (Proposed Standard), Internet Engineering Task Force, Apr. 2004, obsoleted by RFC 6582. [Online]. Available: http://www.ietf.org/rfc/rfc3782.txt

[54] S. Mascolo, C. Casetti, M. Gerla, M. Y. Sanadidi, and R. Wang, "Tcp westwood: Bandwidth estimation for enhanced transport over wireless links," in *Proceedings of the 7th annual international conference on Mobile computing and networking*, ser. MobiCom '01. New York, NY, USA: ACM, 2001, pp. 287–297. [Online]. Available: http://doi.acm.org/10.1145/381677.381704. doi: 10.1145/381677.381704

[55] C. Raiciu, M. Handley, and D. Wischik, "Coupled Congestion Control for Multipath Transport Protocols," RFC 6356 (Experimental), Internet Engineering Task Force, Oct. 2011. [Online]. Available: http://www.ietf.org/rfc/rfc6356.txt

[56] D. Wischik, M. Handley, and M. B. Braun, "The resource pooling principle," *SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 5, pp. 47–52, 2008.

[57] H. M. Chaskar, T. V. Lakshman, and U. Madhow, "Tcp over wireless with link level error control: analysis and design methodology," *IEEE/ACM Trans. Netw.*, vol. 7, no. 5, pp. 605–615, 1999.

[58] W. Tan, W. Cui, and J. G. Apostolopoulos, "Playback buffer equalization for streaming media using stateless transport prioritization," in *Packet Video 2003*, Nantes, France, apr 2003.

[59] S. Shailendra, R. Bhattacharjee, and S. K. Bose, "Optimized flow based modeling of multi-path multi-homed transmission," in *Communication Systems (ICCS), 2010 IEEE International Conference on*, nov 2010, pp. 411 –415. doi: 10.1109/ICCS.2010.5686517

[60] J. Cao, W. S. Cleveland, D. Lin, and D. X. Sun, "Internet traffic tends to poisson and independent as the load increases," Tech. Rep., 2001.

[61] S. K. Bose, *An Introduction To Queueing Systems*. Kluwer Academic/Plenum Publishers, New York, 2002.

[62] *AMPL*. [Online]. Available: www.ampl.com

[63] "Minos." [Online]. Available: http://www.ampl.com/DOWNLOADS/details.html

[64] W. T. Strayer and A. C. Weaver, "Evaluation of transport protocols for real-time communication," Tech. Rep., 1988.

[65] S. Iren, P. D. Amer, and P. T. Conrad, "The transport layer: tutorial and survey," *ACM Comput. Surv.*, vol. 31, no. 4, pp. 360–404, dec 1999. [Online]. Available: http://doi.acm.org/10.1145/344588.344609

[66] A. L. Caro, Jr., "End-to-end fault tolerance using transport layer multihoming," Ph.D. dissertation, Newark, DE, USA, 2005.

[67] S. Shailendra, R. Bhattacharjee, and S. K. Bose, "Mpsctp: A simple and efficient multipath algorithm for sctp," *Communications Letters, IEEE*, vol. 15, no. 10, pp. 1139 –1141, oct 2011.

[68] "The Network Simulator NS-2," http://www.isi.edu/nsnam/ns/.

[69] "Evalvid, A Video Quality Evaluation Toolset," http://www.tkn.tu-berlin.de/research/evalvid.

[70] T. Dreibholz, M. Becke, H. Adhari, and E. P. Rathgeb, "On the impact of congestion control for concurrent multipath transfer on the transport layer," in *Proceedings of the 11th IEEE International Conference on Telecommunications (ConTEL)*, Graz/Austria, jun 2011, pp. 397–404, ISBN 978-953-184-152-8. [Online]. Available: http://www.tdr.wiwi.uni-due.de/fileadmin/fileupload/I-TDR/SCTP/Paper/ConTEL2011.pdf

[71] C. Casetti, M. Gerla, S. Lee, S. Mascolo, M. Sanadidi, and R. Wang, "Enhancing tcp congestion control via connection bandwidth estimation, a performance study," CiteSeerX - Scientific Literature Digital Library and Search Engine (United States), Tech. Rep., 2007. [Online]. Available: http://www.cs.ucla.edu/NRL/hpi/tcpw/tcpw_papers/tech020024-tcpw.pdf

[72] S. Shailendra, R. Bhattacharjee, and S. K. Bose, "Mpsctp: A multipath variant of sctp and its performance comparison with other multipath protocols," in *Communication Systems (ICCS), 2012 IEEE International Conference on*, nov 2012, pp. 1–4. doi: 10.1109/ICCS.2012.6406154

[73] "Multipath TCP," http://code.google.com/p/multipath-tcp/.

[74] J. Postel, "The TCP Maximum Segment Size and Related Topics," RFC 879, Internet Engineering Task Force, Nov. 1983, updated by RFC 6691. [Online]. Available: http://www.ietf.org/rfc/rfc879.txt

[75] R. Jain, D. Chiu, and W. Hawe, "A quantitative measure of fairness and discrimination for resource allocation in shared computer systems," DEC Research Report TR-301, Tech. Rep., sep 1984.

[76] W. H. Wang, M. Palaniswami, and S. H. Low, "Optimal flow control and routing in multi-path networks," *Perform. Eval.*, vol. 52, no. 2-3, pp. 119–132, 2003.

[77] S. Shailendra, R. Bhattachrjee, and S. K. Bose, "Optimized flow division modeling for multi-path transport," in *India Conference (INDICON), 2010 Annual IEEE*, dec 2010, pp. 1–4. doi: 10.1109/INDCON.2010.5712713

[78] S. Shailendra, R. Bhattacharjee, and S. K. Bose, "Improving congestion control for concurrent multipath transfer through bandwidth estimation based resource pooling," in *Information, Communications and Signal Processing (ICICS) 2011 8th International Conference on*, dec 2011, pp. 1–5. doi: 10.1109/ICICS.2011.6174285

[79] D. G. Luenberger, *Linear and Nonlinear Programming*. Addison-Wesley Publishing Company, may 1989.

[80] S. Shailendra, R. Bhattacharjee, and S. K. Bose, "An implementation of minmax optimization for multipath sctp through bandwidth estimation based resource pooling technique," *AEU - International Journal of Electronics and Communications*, 2012. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1434841112001987

[81] L. Lam, K. Su, C. Chan, and X. Liu, "Modeling of round trip time over the internet," in *Asian Control Conference, 2009. ASCC 2009. 7th*, aug 2009, pp. 292 –297.

[82] "The dummynet project," http://info.iet.unipi.it/~luigi/dummynet/.

[83] "The linux kernel archives," http://www.kernel.org.

[84] J. Bennett, C. Partridge, and N. Shectman, "Packet reordering is not pathological network behavior," *Networking, IEEE/ACM Transactions on*, vol. 7, no. 6, pp. 789 –798, dec 1999.