2007-01-01

Concurrent Multi-Path Real-Time Transmission Control Protocol

Anand Jayaraman
University of Miami, anandj@miami.edu

UNIVERSITY OF MIAMI

CONCURRENT MULTI-PATH REAL-TIME TRANSMISSION CONTROL PROTOCOL

By

Anand Jayaraman

A THESIS

Submitted to the Faculty
of the University of Miami
in partial fulfillment of the requirements for
the degree of Master of Science

Coral Gables, Florida

December 2007

UNIVERSITY OF MIAMI

A thesis submitted in partial fulfillment of
the requirements for the degree of
Master of Science

CONCURRENT MULTI-PATH REAL-TIME TRANSMISSION CONTROL PROTOCOL

Anand Jayaraman

Approved:

Dr. Dilip Sarkar Associate Professor of Computer Science	Dr. Terri A. Scandura Dean of the Graduate School
Dr. Burton Rosenberg Associate Professor of Computer Science	Dr. Uttam Sarkar Visiting Professor of Computer Science
Dr. Akmal A. Younis Assistant Professor of Electrical & Computer Engineering	Dr. Huseyin Kocak Chairman & Professor of Computer Science

JAYARAMAN, ANAND (M.S., Computer Science)
Concurrent Multi-path (December 2007)
Real-time Transmission Control Protocol

Abstract of a thesis at the University of Miami.

Thesis supervised by Dr. Dilip Sarkar.
No. of pages in text. (50)

In this thesis, a new transport protocol, the Concurrent Multi-Path Real-time Transmission Control Protocol (cmpRTCP) is proposed. The proposed protocol has been designed to handle real-time streams (video and audio) over IP-networks. One of the key strengths of this protocol lies in its ability to intelligently exploit the availability of multiple paths between multi-homed hosts for concurrent transmission of unicast real-time streams. This work describes the architecture and operation of cmpRTCP in detail. In addition, the limitations of currently used transport protocols in handling real-time streams are also discussed. These limitations of other protocols have played a vital role in the design process of the proposed protocol. Experiments to evaluate the performance of cmpRTCP against other protocols and the results obtained therein are also documented in this work. Results show that cmpRTCP is a best effort protocol that tries to maximize the amount of data that is successfully delivered to the destination in a timely manner under varying drop and delay conditions of the network.

Dedication

This thesis is dedicated to my Guru - Sri Sathya Sai Baba, who has not only been a true friend, tutor and guide but also a source of immense inspiration in my life.

Acknowledgements

I would like to thank Dr. Dilip Sarkar for his support during the time frame of this research activity. I would also like to thank the Chairman of the Department of Computer Science Dr. Huseyin Kocak and all members of my thesis committee for their support, guidance and co-operation. I extend my appreciation towards the University of Miami for all the facilities provided.

Special thanks to the Associate Dean of Arts & Science Dr. Jacqueline Dixon, Program Director of Visual Journalism Lelen Bourgoignie-Robert and Vice Dean of School of Communication Sanjeev Chatterjee for their advice and words of encouragement.

A million thanks to the people of my home for their everlasting encouragement and support.

Table of Contents

List of Tables

List of Figures

Glossary

ATO	Acknowledgment Timeout is the maximum time duration for which a sender waits for an acknowledgment on a particular path before considering the path to have gone bad
CDT	Concurrent Data Transport, also termed CMT (Concurrent Multi-path Transmission) is a mode of transmission where all the available paths from source to destination are utilized concurrently for data transfer
Chunk	A block of data that is dispatched as payload to the IP layer from the transport layer
Congestion Avoidance phase	A congestion window growth phase where the congestion window grows linearly
CTSNA	Cumulative TSN Acknowledgment is the TSN value that indicates that all chunks with TSN values less than or equal to the CTSNA have been received by the receiver
CWND	Congestion Window for a path limits the data, in number of bytes that a sender can send along that particular path before receiving an acknowledgment

GOP	A group of successive pictures within an MPEG-coded video stream
H.264	An ITU standard for compressing video based on MPEG-4
LTSNF	The TSN of the last chunk that has been flushed to the upper layer at the receiver's end
MTU	Maximum Transmission Unit is the size of the largest packet that the network layer can handle as a single block without fragmenting it
Multi-homing	The ability of a transport layer protocol to probe for and utilize multiple network paths between source and destination
RTDTL	Real-Time Delay Tolerance Limit is the maximum amount of time that a receiver will wait for a chunk to arrive before considering it to be lost in the network
RTT	Round Trip Time for a path is the time required for a packet dispatched on that path to reach the destination and for the destination's acknowledgment to return back to the source
SACK	Selective Acknowledgment is an acknowledgment packet that is dispatched by the receiver acknowledging the reception of new data chunks
Slow Start phase	A congestion window growth phase where the congestion window grows exponentially

SSTHRESH	Slow Start Threshold for a path is a threshold value for the CWND of that path. When the CWND exceeds the threshold, the window growth switches from Slow Start phase to Congestion Avoidance phase
TSN	Transmission Sequence Number is a unique number assigned to every chunk from a monotonically increasing 32-bit number sequence to identify the relative position of a chunk during the transmission of a stream of chunks
WUQ	Window Update Queue for a path is the queue that holds relevant information about the chunks that were dispatched on that particular path

Chapter 1

Introduction

This chapter introduces the reader to the role of transport protocols in real-time data transport. It provides the reader with a generic idea of the commonly used transport protocols and the problems associated with them. It also describes some of the research work of the recent past that has been done in the area of transport protocols to make them more suitable for real-time streams.

1.1 Background

Recent advances in digital networking technology coupled with the rapid increase in consumption of digital content over the intra / internet have placed a greater emphasis on bandwidth aggregation, *network load balancing* (NLB) and reliable communication. The challenges to be tackled only get bigger when considering real-time video/audio streams owing to the time-sensitive nature of real-time data. As stated in [20], application level end-to-end delays exceeding 250 ms affect data delivery of real-time streams leading to unintelligible real-time interaction from an end-user's perspective. Also, real-time data transfers cannot be compared with voluminous file data transfers because the idea is not to utilize the highest available network bandwidth for fast transmission but rather transmit data at the rate at which it is dispatched by the real-time source while ensuring minimal jitter.

Currently, real-time applications utilize the *user datagram protocol* (UDP) at the transport layer for their transmissions. UDP is connection-less and does not re-transmit packets, making it a lightweight protocol. Also, UDP does not take care of re-ordering packets arriving out of order at the destination. Applications using UDP have no knowledge of network status and hence may under-utilize available bandwidth or worsen the congestion in the network [4]. The standard *Transmission Control Protocol* (TCP) [11] on the other hand is connection-oriented, takes care of retransmission and does re-ordering of packets arriving out of order at the destination. Although TCP does have some knowledge of the network congestion status; TCP's retransmission to ensure that each and every packet does reach the destination is an expensive (time consuming) process for real-time streams. Retransmitted packets over networks with reasonable delays have little value at the receiving end in real-time applications such as *Voice over IP* (VoIP) and *Digital Video over IP* (DVIP) because of their late arrival. Also, in the process of retransmission of a set of data packets, newer data being dispatched from the source gets held up until the retransmission is complete. Thus a cycle of constantly increasing delay in data delivery sets in during the length of the transmission which is unacceptable for real-time streams.

Another limitation of both TCP and UDP is their inability to probe for and utilize multiple paths if available between hosts equipped with multiple network interfaces (multi-homed hosts). TCP / UDP can bind to only one IP-endpoint at either end. Applications can however split data across multiple connections to enable multi-homed streaming [1]. It is shown in [10] that multi-homed streaming can improve quality of reception (Q^2) by 30% or more. While some studies on non real-time traffic have proposed multipath data transfer solutions at application layer and network layer (see [15, 23] and the references therein), it has been clearly shown in [15] that it is the transport layer that is best equipped with end-to-end information and hence most suitable for positioning the multipath data transport capability.

[1] Multi-homed Streaming: Process of streaming data from source to destination via multiple paths

[2] Q: fraction of data packets from a given stream session that reach the receiver on or before their respective scheduled play-out time

A protocol that currently exists for real-time applications (RTP) [24] utilizes a UDP channel for data transmission and a TCP channel for transmitting out of band control information (RTCP [24]). The control overhead is about 5% of the data load as stated in [24]. Although classified as a transport protocol, RTP attaches itself to an application and needs an underlying transport protocol (UDP in general) for data transmission. RTCP, the out of band control protocol does not define a congestion control mechanism for RTP. It merely has the ability to provide statistical information to the sending application periodically about data reception at the receiver. The application may use this information to downgrade or enhance its encoding rate of the real-time stream that is being generated. RTP is also not designed to take advantage of the multi-homing capability of hosts.

Considering these limitations of TCP and UDP as well as the problems of multi-homing at the application layer for real-time streams, this work proposes a new transport protocol - the *concurrent multi-path real-time TCP* (cmpRTCP). The new protocol has been designed to track congestion in the network, not to retransmit packets but to ensure re-ordering of packets that arrive out of order at the destination. One of the key features of this protocol is its ability to intelligently exploit multiple paths between multi-homed hosts to achieve concurrent real-time data transmission. This protocol implements a feedback control system that takes care of maximizing the amount of data that is successfully delivered to the destination in a timely manner. The key features of the proposed protocol are (*i*) multi-path congestion control (*ii*) path viability estimation and multi-path load balancing (*iii*) increased end-end reliability using multiple paths (*iv*) improved QoS by scheduling more packets on better paths.

1.2 Recent Research Work

Several research studies have been conducted in the recent past in the areas of real-time stream transmission enhancement and multi-path data transfer. Studies have

shown that computers equipped with *multiple network interfaces* (MNIs) can be utilized for concurrent multi-path data transport suitable for bandwidth aggregation as well as increased reliability [3, 18, 10, 1, 2, 21]. Most of the multi-path data transport solutions have extended the idea from *Stream Control Transmission Protocol* (SCTP) [26] owing to its ability to probe for and establish multiple paths between multi-homed hosts. However, the multi-path solutions extending SCTP are designed for reliability with full retransmission making them appropriate for non real-time data transfers and not for real-time streams.

A multi-path protocol for real-time streams called *MRTP* has been proposed in [18] with the ability to partition and dispatch packets across multiple paths. MRTP, like RTP, is a layer on top of the transport protocol in the *Open Systems Interconnection* (OSI) stack [8] that utilizes services offered by TCP / SCTP or UDP.

Adapted versions of the *Partial reliability* (PR) *extension of SCTP* (SCTP-PR) [25] have been used in solutions proposed in [19] and [9] for real-time transport. While these proposals may help improve perceptible quality of the real-time stream at the receiving end, the problem with these solutions is the inability of the receiver to drop late arriving packets at the receiving end without explicit instruction from the sender to do so. These proposed protocols do not have a mechanism to identify late arriving packets nor the ability to notify the sender about them. The protocol proposed in [9] has incorporated a mechanism for bandwidth estimation and a scheduling policy that uses the estimate to reduce relative delay variations on concurrent paths. However, loss rates on various paths are not used in scheduling the packets.

The novel *concurrent multi-path real-time transmission control protocol* proposed in this work not only overcomes the problems faced by the different adapted versions of SCTP-PR, but also preserves the novel design and implementation features of cmpTCP (the non real-time counterpart of cmpRTCP) reported in [23].

1.3 Motivation and Goals

The different transport protocols suffer from several limitations that have been briefly described in this chapter and are described in greater detail in the next chapter under section 2.4. The primary motivation for this work has been to bringforth these limitations and propose a protocol that is capable of overcoming these limitations thereby ensuring real-time data transport in a timely manner.

1.3.1 Goals

The main goal of this work has been the design and implementation of a transport layer protocol that is capable of intelligently exploiting multiple paths between multi-homed hosts for real-time streaming.

1. Increased Availability: By definition, availability is the ability of data channels to be ready for use when required and not be committed to other tasks. The protocol must increase availability by probing for multiple paths and utilizing them concurrently for data transport.

2. Increased Robustness: Robustness is the ability of a system to withstand changes in its operating environment while ensuring minimal impact on the task at hand. The protocol be robust enough to handle changes in path delay / drop conditions to ensure minimal data loss.

3. Increased Scalability: Scalability is a measure of how well a system expands. The protocol must be able to scale well with increase in number of available paths

1.4 Outline of this work

The rest of the work described in this document is organized as follows. Chapter 2 discusses the operation of the various transport layer protocols, their limitations

when handling real-time streams in greater detail, and the need for the proposed protocol. Chapter 3 describes the detailed architecture and operation of cmpRTCP. In chapter 4, the experiments that were conducted to evaluate the performance of the protocol have been described along with the results from those experiments. Finally, in chapter 5 an overall conclusion is drawn from the work in its entirety.

Chapter 2

Transport Protocol Families

This chapter introduces the reader to the families of transport protocols that have influenced the design of cmpRTCP because of their limitations and also paved the way for its implementation.

2.1 Categorization

Transport protocols can be categorized along multiple dimensions. The primary ways of categorizing them are listed below

1. Based on their ability to establish multiple paths between source and destination, they are categorized as either *Multi-Path* or *Single-Path* transport.

2. Based on their ability to utilize multiple paths simulataneously for data transmission, they are categorized as either *Concurrent* or *Non-Concurrent* transport.

3. Based on their ability to detect and react to congestion and loss in the network, they are categorized as either *Loss Aware* or *Loss Unaware* transport.

4. Based on their ability to deliver all user messages (messages sent from the layer above the transport layer in the protocol stack) that were dispatched to the destination, they can be categorized as either *Reliable* or *Unreliable* transport.

5. Based on their ability to deliver user messages / user bytes at the destination in the same order as they were dispatched at the source, they can be categorized as either *Ordered* or *Unordered* transport.

2.2 Single-Path Transport

Protocols that belong to this family, bind to a single IP endpoint at both the source and the destination ends (refer Fig. 2.1) when a user session is started. Data transmission during the entire session is along the path connecting the two IP endpoints. TCP and UDP belong to this family of protocols.

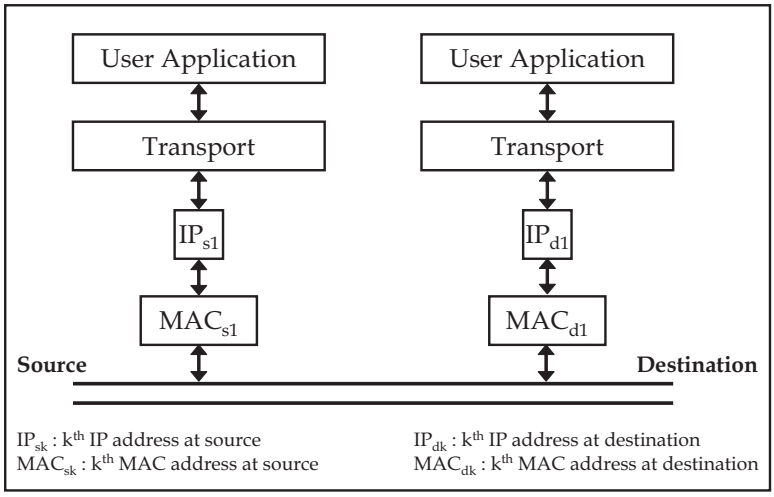

Figure 2.1: Block Diagram of Single-Path Transport

2.2.1 UDP

User Datagram Protocol, a light weight transport protocol that can be categorized as a single-path, loss unaware, unreliable and unordered delivery transport (refer Table 2.1). UDP [22] is message oriented in nature which means that if an application at the destination receives a set of bytes from the underlying UDP transport layer, the set of bytes constitute a complete user message sent from the source. When applications use UDP, messages may arrive out of order or go missing without notice. The fact that the overhead of checking if every packet arrives at the destination is avoided,

	Multi-Path	Concurrent	Loss Aware	Reliable	Ordered Delivery
UDP					
TCP			Yes	Yes	Yes
SCTP	Yes		Yes	Yes	Yes
SCTP-PR	Yes		Yes		Yes
SCTP (CDT)	Yes	Yes	Yes	Yes	Yes
cmpTCP	Yes	Yes	Yes	Yes	Yes
SCTP-PR (CDT)	Yes	Yes	Yes		Yes

Table 2.1: Protocol Feature Comparison Chart

	Message Oriented	Multi-streaming	Bundling
UDP	Yes		
TCP			
SCTP	Yes	Yes	Yes
SCTP-PR	Yes	Yes	Yes
SCTP (CMT)	Yes	Yes	Yes
cmpTCP	Yes	Yes	Yes
SCTP-PR (CMT)	Yes	Yes	Yes

Table 2.2: Protocol Feature Comparison Chart

makes UDP faster and more efficient than most protocols. The limitations of using UDP for real-time data transport are mentioned in section 2.4.

2.2.2 TCP

Transmission Control Protocol is the core protocol used on the internet for reliable transmission of data. TCP [11] is categorized as a single-path, loss aware, reliable and fully ordered delivery transport (refer Table 2.1). TCP is stream oriented in nature which means that data is received by applications at the destination as a continuous stream of bytes without any demarcation at message boundaries. The strict ordering and reliability of TCP makes it extremely useful for lossless transfer of data from source to destination. However, there are problems when using TCP for real-time data (refer section 2.4).

2.3 Multi-path Transport

Protocols that belong to this family, bind to multiple IP endpoints at both the source and the destination ends (refer Fig. 2.2) at the beginning of a user session. Data transmission is along one or more paths connecting the multiple IP endpoints for the session. Some of these protocols also support addition of new IP endpoints as well as removal of IP endpoints when a session is in progress. SCTP, SCTP-PR, SCTP (CDT), SCTP-PR (CDT) and cmpTCP belong to this family of protocols.

2.3.1 Multi-path Non-concurrent Transport

These are protocols that establish multiple paths between source and destination but utilize only one path for transmission while reserving the rest for fail-over.

2.3.1.1 SCTP

The *Stream Control Transmission Protocol* (SCTP) [26] was the first protocol of its kind that enabled multi-homed hosts to communicate via multiple paths. SCTP

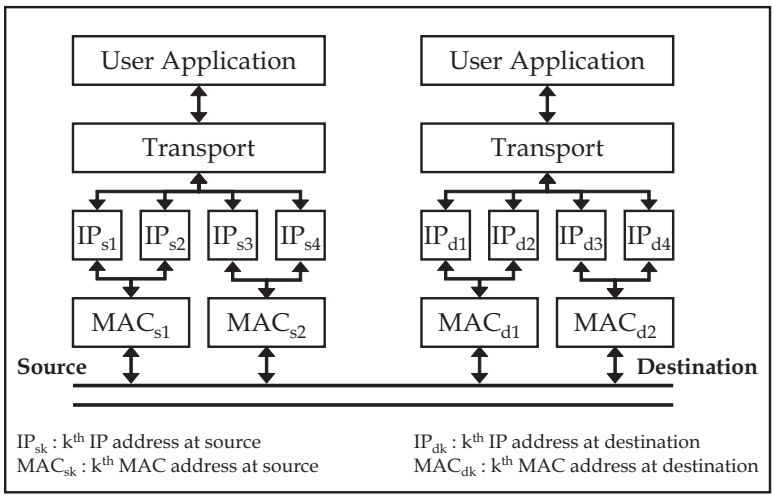

Figure 2.2: Block Diagram of Multi-Path Transport

provides features such as sequenced delivery of user messages within multiple streams, optional bundling of multiple user messages into a single SCTP packet and network-level fault tolerance through supporting of multi-homing at either or both ends of an SCTP association.

On a multi-homed host, SCTP has the capability to tie down multiple IP addresses of the host to a common SCTP endpoint that can be used for data transmission or reception. This means that when an SCTP association is established between two multi-homed hosts, all potential network paths for data transfer are scouted for and kept track of by the protocol. SCTP's ability to scout for and establish multiple paths for communication has in fact made it the backbone for concurrent data transport protocols described in section 2.3.2.

In [17] it has been shown that SCTP is definitely a better protocol than TCP for MPEG-4 real-time transmission. At the same time, SCTP also suffers from some limitations (refer section 2.4).

2.3.1.2 SCTP-PR

The *Stream Control Transport Protocol - Partial Reliability Extension* (SCTP-PR) [25] is the extension of SCTP that supports partially reliable data transmission service to the upper layer protocol. This means that unlike SCTP which does a complete

ordered delivery of messages to the upper layer protocol, SCTP-PR has the feature where the sender can inform the receiver to not wait for certain data packets that may not have arrived at the receiver. This is particularly useful when considering real-time transmissions where every user message has a lifetime before which it must reach the destination. Messages that have expired at the sender no longer need to be transmitted or retransmitted and the receiver can be notified of the same using this protocol. SCTP-PR has formed the backbone for many research studies that have tried to propose a solution for concurrent real-time data transport owing to its ability to establish multiple paths for communication and support partially reliable data transfer service.

In [27] it has been shown that SCTP-PR performs much better than TCP and UDP for real-time transmission by reducing relative retransmission and transmission delays respectively. In [19], it has been shown that better picture quality (PSNR[1]) of real-time video can be obtained using SCTP-PR as compared to UDP by exploiting the partially reliable data transfer service of SCTP-PR to retransmit only I-frames. The problems associated with SCTP-PR are documented in section 2.4.

2.3.2 Multi-path Concurrent Transport

These are protocols that establish multiple paths between source and destination and utilize all the available paths concurrently to transfer data.

2.3.2.1 SCTP (CDT)

There are many adapted versions of SCTP that have been proposed for concurrent data transport [3, 7, 1, 2, 12, 13, 14, 15, 16, 23]. These adaptations allow data transmission on multiple paths concurrently by tracking and controlling the amount of packets dispatched on each path at any point during the session, based on the reception of acknowledgments from the receiver. The higher the number of acknowledgments that are received for packets dispatched on a particular path, the higher

[1]PSNR: The peak signal to noise ratio that is a measure of the quality of picture reconstruction

the potential of that path to transmit more data is the common philosophy applied across all these versions. Section 2.4 highlights the limitations of using the concurrent transport SCTP for real-time streams. The *Concurrent Multi-Path Transmission Control Protocol* (CMPTCP) described in [23] is an adaptation that utilizes a single transmission queue and a scheduler beneath the queue that dispatches the packets on the available paths in a round robin fashion. cmpTCP is in fact the non real-time counterpart of cmpRTCP.

2.3.2.2 SCTP-PR (CDT)

Similar to the many adapted versions of SCTP as described in section 2.3.2.1, SCTP-PR also has many adapted versions for concurrent transport. The general adaptation as described in [9] as a greedy scheduler is one where a scheduler picks up the packets from a global transmission queue and schedules them over the different available paths in a round robin fashion. This adaptation is referred to as SCTP-PR (CDT-1) in latter sections. One of the enhanced adaptations has also been clearly described in [9] which utilizes a separate transmission queue for every available path and a scheduler to dispatch the user messages from the upper layer protocol to one of these transmission queues depending on a contrived bandwidth heuristic for each one of those paths. This adaptation has been termed SCTP-PR (CDT-2) in latter sections. The section on limitations of the various protocols (2.4) highlights the problems of using SCTP-PR (CDT) for real-time streams.

2.3.3 Misc. Protocols

These are management protocols that operate on top of the transport layer in the protocol stack to achieve some form of concurrency in multi-path transmission. These protocols can utilize either TCP, UDP, SCTP or any of its variants for the actual data transport.

2.3.3.1 AppStripe TCP

AppStripe TCP is not strictly a transport layer protocol. Here, it is the application that establishes multiple TCP connections between source and destination, and transmits over all paths [15]. The application must manage path failures, re-ordering of packets sent over the different paths at the destination and also the round robin scheduling over the available paths.

2.3.3.2 AppStripe UDP

Similar to AppStripe TCP mentioned in section 2.3.3.1, AppStripe UDP is not a transport layer protocol. The application establishes multiple UDP connections between source and destination, and transmits over all paths. Again, it is the application which manages path failures, round robin scheduling of messages over the available paths and re-ordering of packets sent over the different paths at the receiving end.

2.3.4 MRTP

The *Multi-flow Real-time Transport Protocol* (MRTP) proposed in [18] is a protocol with the ability to partition and dispatch user messages across multiple paths. It forms a layer on top of the transport protocol in the stack and establishes multiple flow connections utilizing the underlying protocol (TCP / UDP / SCTP). In [18], it has been shown that by appropriately positioning the protocol over one of the transports, the quality of real-time streams can be enhanced.

2.4 Limitations of the Various Protocols

2.4.1 UDP

UDP has no knowledge of the congestion or losses occurring in the network and hence can severely hamper the quality of real-time data transmission by under-utilizing available bandwidth or worsening congestion in the network [4]. Also, UDP requires

that the application at the receiver take care of re-ordering messages that arrive out of sequence.

2.4.2 TCP

The main problem with TCP when used for real-time streams is TCP's retransmission policy. While the protocol ensures that each and every single packet does reach the destination by making the required retransmissions, it is not useful because retransmitted packets have little value at the receiving end owing to the delay involved. The fact that retransmission of a set of data packets holds up newer data from being dispatched until retransmission is complete, sets in a cycle of constantly increasing delay in data delivery at the receiver. The other problems of TCP include the inability to parallelize several transmission streams without creating multiple TCP connections (a problem that has been addressed in SCTP).

2.4.3 SCTP

One of the limitations of SCTP is that it uses only one path which is designated as the primary path for transmission. Alternate paths are used only when the primary path fails. Even the alternate paths are used only one at a time based on their precedence. In any case, it is clear that SCTP is not meant for concurrent data transport on multiple paths. When used for real-time transmission on a single path, it suffers from the same retransmission problem as TCP.

2.4.4 SCTP-PR

SCTP-PR suffers from three major limitations. (*i*) Similar to SCTP, SCTP-PR transmits only on one path (primary path or an alternate path) at any given time. All other paths are used as backup. (*ii*) The transport layer at the receiver does not have the ability to drop late arriving packets without explicit instruction from the sender to do so. (*iii*) The receiver has no way to notify the sender of late arriving packets.

2.4.5 SCTP (CDT)

This protocol is not suitable for real-time transmission because it supports complete ordered delivery of user messages to the upper layer protocol at the receiving end which poses the same problem as that of TCP and SCTP. Also, the other problems with this protocol are (*i*) The receiver does not have the ability to drop late arriving packets. (*ii*) The receiver has no way to notify the sender of late arriving packets.

2.4.6 SCTP-PR (CDT)

Although this is the best protocol of the entire lot for transporting real-time streams, both the general version of this protocol that has the partial reliability extension added to the SCTP Concurrent version and the version described in [9] suffer from the same retransmission problem as in most other protocols mentioned above. The partial reliability extension does support for packets to be discarded as and when they expire at the sender, eliminating the need for retransmission of those packets, but the design of the protocol is such that for reasonable delays in the network, even the few retransmissions can cause head of line blocking. Also, for every set of packets that expire, a forward-TSN (refer [25]) needs to be sent by the sender to the receiver to indicate the same, until when the packets that are being buffered at the receiver may expire. These issues have been clearly illustrated in the various scenarios shown in chapter 4.

The major limitations of the protocol illustrated in [9] are (*i*) packet loss rates on the various paths do not influence the choice of paths. (*ii*) Delayed packet delivery at the receiver is not communicated to the sender so as to influence the choice of paths. These are in addition to the head of line blocking problem as with the general version.

2.4.7 AppStripe TCP

All the limitations of TCP with the retransmission issues also apply here. In addition, the application at the receiver has the overhead of reordering packets received from the multiple connections. At the sending end, the application has the overhead of scheduling packets without the knowledge of congestion in the different paths. This information although present at the transport layer in all the connections is not available to the application.

2.4.8 AppStripe UDP

The application at the receiver has the overhead of reordering the packets in this case. Also, since UDP has no network congestion awareness, this protocol cannot favor one path over another.

2.4.9 MRTP

The major limitation of this protocol is the very fact that it is not a transport layer protocol and hence requires a lot of overhead to communicate path status and other network congestion related parameters to either end. It has been very clearly shown in [15] that it is the transport layer that is best equipped with end-to-end network information for handling multi-path data transport. While a combination of MRTP and UDP protocols is useful for aggregation of bandwidth in wireless ad hoc networks, it is not major quite useful for improving *quality of service* (QoS) at the receiving end without sending redundant packets nor for decreasing total packet loss rate or increasing reliability of application level connectivity.

2.5 Defining the Real-Time Transport Features

From the previous sections that have highlighted the different protocol families and the limitations of the various protocols, the attributes of a protocol capable of real-time data transport can be enumerated.

1. The protocol must be capable of concurrent data transport so as to exploit the availability of multiple paths between hosts.

2. The protocol must be loss aware and react to data loss in the network

3. The protocol must be able to divert traffic to better paths and achieve optimal load balancing.

4. It is preferable if the protocol is an ordered delivery protocol so as to eliminate need for different applications to implement their own ordering schemes for every session that they open.

5. The protocol must be able to minimize and trade off its reliability for timely delivery.

This clearly marks the boundary and scope of operation for the cmpRTCP protocol whose design has been discussed in subsequent chapters.

Chapter 3

cmpRTCP

3.1 Protocol Design

3.1.1 Basic Operation

cmpRTCP establishes a multi-homed connection between the source and destination hosts in the same manner as cmpTCP [23] (very similar to the mechanism in SCTP [26]). The process of connection establishment is described in section 3.1.2. The entire design of cmpRTCP beyond the connection establishment is based upon the simplistic goal that the sender must make a best effort to ensure that every packet / data chunk reaches the destination with no retransmission. For this purpose, the transport protocol at the sender must be equipped with

1. A congestion window manager that continually tracks the network congestion status of the multiple paths that have been setup for concurrent data transport.

2. A real-time scheduler that schedules packets over the multiple paths based on the inputs from the congestion window manager.

Similarly, the receiver must be equipped with the ability to aid the sender by informing it of

1. Packets that are arriving late on particular paths

2. Packets that have not shown up at all within a reasonable time limit.

This is of course in addition to the normal multi-path acknowledgements with gap reports (refer cmpTCP [23]).

3.1.2 Connection Establishment

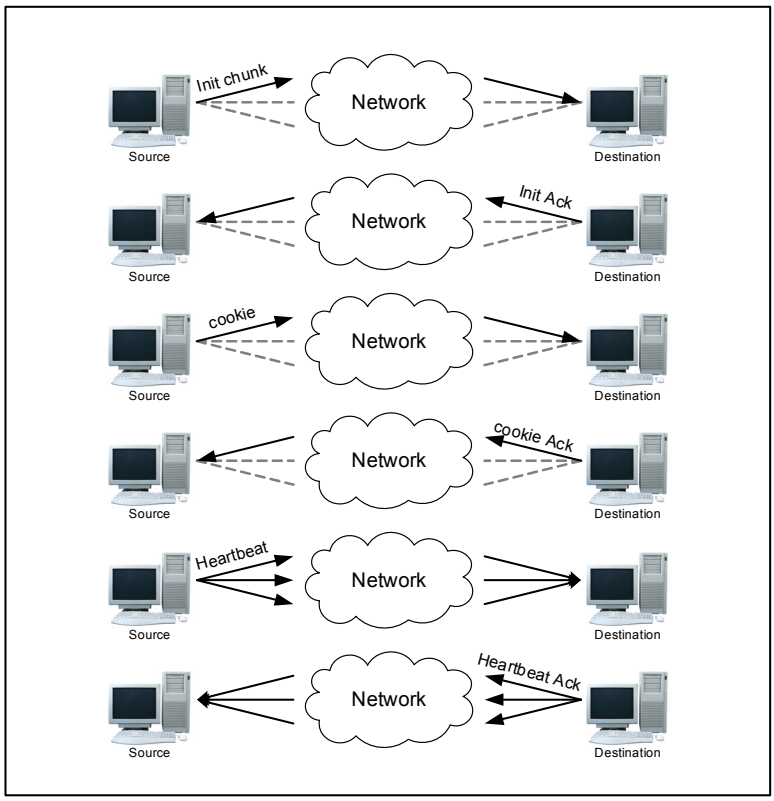

Figure 3.1: Connection Establishment Process

The connection establishment procedure is a 6-way handshake as shown in Fig. 3.1. The process can be carried out on top of IPv4 or IPv6 layers [26]. Firstly, the sender sends a packet with a *connection initialization data chunk* (INIT chunk) which primarily contains information about the sender's multiple IP addresses. The receiver responds back with a *connection initialization acknowledgement chunk* (INIT ACK chunk) which contains the multiple IP addresses that the receiver is ready to accept data on along with a *state cookie* [26] and a *message authentication code* (MAC) [26] that are used for authenticating the sender in the next stage. The sender responds to

the INIT ACK with a *Cookie Echo chunk* that contains the original state cookie and the MAC. The receiver upon receiving the cookie echo, authenticates the sender by verifying that the MAC that is computed from the state cookie in the echoed chunk is the same as the MAC on the echoed chunk. Upon authentication, the receiver sends a *Cookie Ack chunk* to confirm establishment of the connection.

The next two stages of the handshake are for activating the multiple paths if available, for concurrent transmission. The sender broadcasts a *Heartbeat chunk* to every IP address of the multi-homed destination that was present in the INIT ACK chunk. The receiver responds with a *Heartbeat ACK chunk* on every path in which it received the heartbeat chunk, notifying the sender that the path is indeed active and suitable for data transmission. Data transmission commences right after the paths are registered as active at the sender after reception of all the heartbeat acknowledgements.

3.1.3 Sender Design

Fig. 3.2 depicts the overall architecture of cmpRTCP at the sender end. In addition to the two core modules (the congestion window manager and the real-time scheduler, the other modules that peform the relevant supporting roles are the Stream engine, the SACK processing module and the Packet dispatcher. The upper layer is allowed to send multiple parallel streams of real-time data for transmission through the established connection. In order to accommodate and manage the flow of the various data streams that need to be transported, there exists the stream engine which acts as a stream multiplexing, message fragmenting and time-stamping unit, creating data chunks out of the messages from the upper layer. The scheduler picks up these chunks queueing up in the transmission queue and chooses a path for dispatching them. The choice of path is based upon a heuristic that combines the following four factors (*i*) size of the *congestion window* of the path, (*ii*) *outstanding bytes in flight* (bytes that

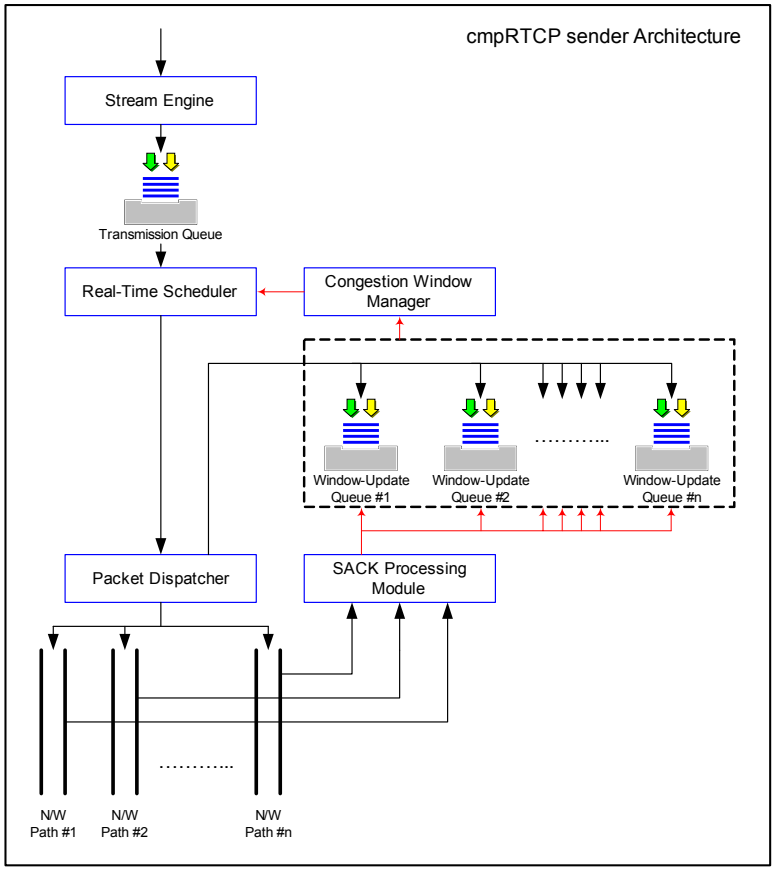

Figure 3.2: Block Diagram of sender side cmpRTCP

are awaiting acknowledgement) on the path, (*iii*) number of chunks that are apparently missing (dropped / delayed) on the path, (*iv*) *Round Trip Time* (RTT) of the path. The window manager tracks the above factors to aid the scheduler.

The packet dispatcher transmits packets scheduled for transmission and places the important attributes of every transmitted data chunk - size, *Transmission Sequence Number* (TSN) and timestamp (time when the chunk was handed over to the protocol for dispatch) in the appropriate *Window Update Queue* (WUQ) corresponding to the path of transmission.

The SACK processing module scans every incoming acknowledgement packet and uses the information contained in the packet to update the WUQs. For information contained in acknowledgements, refer section 3.1.4.2.

3.1.3.1 Real-time Scheduling

The scheduler as described previously (section 3.1.3) has the responsibility to load balance across the multiple paths based on their availability. Algorithms 1 and 2 illustrated in this section, are two scheduling algorithms that were developed and deployed to understand the importance of good scheduling. The basic cmpRTCP protocol uses the first algorithm shown while the second algorithm is used in cmpRTCPa (a variant of cmpRTCP). These algorithms execute as atomic operations. New events from layers in the protocol stack above or below, queue up until the algorithm completes a full pass across all paths. From the algorithms, it can be seen that during every round, when a burst of packets arrive from the upper layer, cmpRTCP fills up each available path to its full capacity (capacity of each path is determined by the window manager - 3.1.3.2) before moving on to the next path in a round robin fashion. Every successive round takes over from the path that was previously used if it was not filled completely in the previous round; the next path by round robin otherwise.

cmpRTCPa - the variant of cmpRTCP, is based on the idea that information about the missing packets can not only be used to control the amount of data being

forall *i such that i is a valid path number* **do**
  if $obpa(i) < cwnd(i)$ **then**
    transmit on path i until $obpa(i) = cwnd(i)$;
    if *more data is pending transmission in queue* **then**
      choose next path i;
    end
  end
end
if *no path was available for transmission and data is pending transmission* **then**
  find path j that has the minimum ratio of $\frac{obpa(j)}{cwnd(j)}$ over all paths;
  transmit one MTU of data on path j;
end

Algorithm 1: cmpRTCP Scheduler

Data: $PathSet \leftarrow$ set of valid path numbers sorted in ascending order of the number of packets missing on the respective paths
Let j be the first path from $PathSet$;
while *data is pending transmission* **do**
  if $(obpa(j) < cwnd(j))$ *or* $(missing(j) + sentAlready(j) < missing(j+1))$ **then**
    transmit on path j;
  end
  else
    choose path $j + 1$;
    if $j + 1$ *is not a valid path number* **then**
      break out of the loop;
    end
  end
end

Algorithm 2: cmpRTCPa Scheduler

dispatched on each path but also direct the decision control of choosing a path. The variant thus operates by sorting the paths in the increasing order of the number of packets missing on the respective paths and then choosing paths in that order, dispatches packets as much as their respective congestion windows would permit. In addition, cmpRTCPa may dispatch data exceeding the path capacity if it finds that loss of all of that data in excess of the capacity, still does not make that path worse than the next best.

3.1.3.2 Window Management

The congestion window manager uses an *additive increase multiple decrease* (AIMD) congestion controller for every path. The basic AIMD technique is similar to cmpTCP [23]. The window growth algorithm is illustrated in Table 3.1. However the algorithm needs to account for several factors that don't play a part in conventional non real-time data transfer. Starting with the fact that no retransmission ever occurs.

(*i*) When packet drops are detected and the congestion window shrinks, the number of packets in-flight would be more than the new window size. In a transport protocol with retransmission, this would immediately trigger a retransmission of the reportedly missing packets so as to free up the window for newer data. But here, since there is no retransmission, if it so happens that all paths have more bytes in-flight than their respective windows, transmission would freeze. To prevent transmission from freezing, at least one *maximum transmission unit* (MTU) of data is transmitted even if there is no path whose window accommodates a data packet.

(*ii*) The beginning of any real-time data transmission is usually accompanied by a sudden burst of data. Message fragments that don't make it to the network during the initial burst will have to wait for at least a whole *round trip time* (RTT) in the transmission queue and another $\frac{1}{2}$ RTT to travel to the destination by when there is a strong possibility that they expire. To prevent severe loss of these fragments at the beginning, the initial window size is set to an integral multiple of the MTU size.

Table 3.1: Window Management Scheme

Begin Transmission
Condition : Before first dispatch on path p
$ssthresh(p)$ = receiver window size
$cwnd(p)$ = integral multiple of $MTU(p)$
Slow Start
Condition : $cwnd(p) = ssthresh(p)$
And b bytes acknowledged in the latest SACK
for chunks dispatched over path p
$cwnd(p) = cwnd(p) + min[MTU(p), b]$
Congestion Avoidance
Condition : $cwnd(p) > ssthresh(p)$
$cwnd(p) = cwnd(p) + \frac{MTU(p)}{cwnd(p)}$
Packet Drop Detection
Condition : for a chunk dispatched on path p,
Gap report > 3 or Receiver notifies late packet arrival via SACK
And the congestion window size of p is not locked
$ssthresh(p) = max[\frac{1}{2}cwnd(p), 2MTU(p)]$
$cwnd(p) = max[\frac{1}{2}cwnd(p), MTU(p)]$
lock the window size of p for a duration RTT(p)
Acknowledgment Timeout (ATO)
Condition : T3 timer expired with duration ATO for path p
And the congestion window size of p is not locked
$ssthresh(p) = max[\frac{1}{2}cwnd(p), 2MTU(p)]$
$cwnd(p) = MTU(p)$
Idle Path
Condition : No transmission on path p for duration of 1x ATO
$cwnd(p)$ = integral multiple of $MTU(p)$

(*iii*) To ensure that minimum data is sent over lossy paths when multiple paths are actually available for transmission, the congestion window is allowed to shrink to a minimum of a single MTU.

(*iv*) To prevent a sudden burst loss from immediately sealing the window, the time interval between successive collapses of the congestion window is chosen to be a single RTT of the corresponding path (as opposed to a fast retransmit phase [26] that locks the window).

(*v*) If an incoming acknowledgment packet indicates new data received at the destination, then the amount of bytes corresponding to the data packets acknowledged is used for appropriately incrementing the window sizes of the corresponding paths on which those data packets were dispatched. On the other hand, if the incoming acknowledgment indicates data being flushed to the upper layer at the destination (refer section 3.1.4.2 - LTSNF), all unacknowledged data prior to the data flush indicated by the incoming acknowledgment, are considered lost and window sizes of paths on which the unacknowledged data was originally dispatched are collapsed appropriately.

3.1.3.3 Queue Management

The WUQs (Fig. 3.2) hold information about every chunk dispatched over the network. Each queue has details about chunks dispatched in the path corresponding to that queue, in the increasing order of TSNs. The purpose of holding information in this sorted manner is to ensure that true gaps in the transmission sequence arising out of packet drops in the network are promptly identified and used by the congestion window manager to adjust the window size of each path, while ensuring that spurious gap reports that arise out of relative path delay differences are discarded.

Everytime an unacknowledged chunk is encountered in any of the WUQs such that there is another chunk with a higher TSN in the same queue that is acknowledged, the gap count (a value that indicates the number of times a loss has been identified for a particular chunk) of the unacknowledged chunk is incremented by one. When

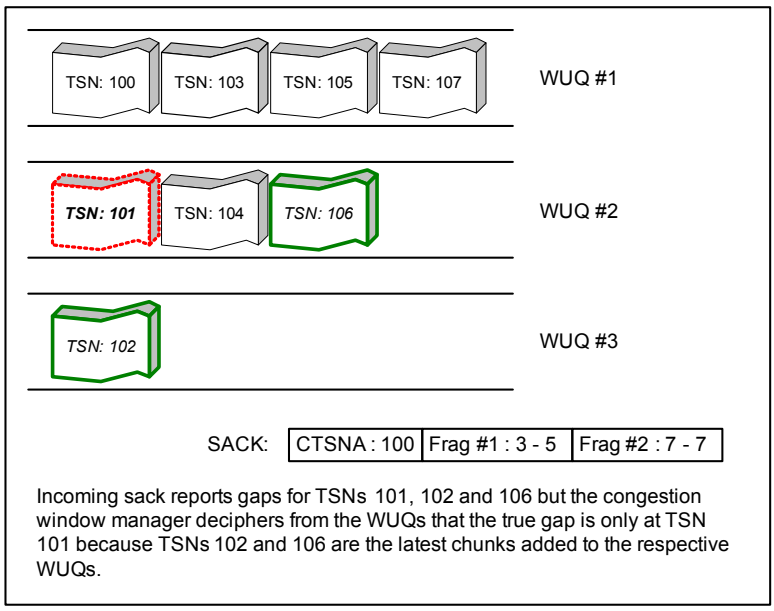

Figure 3.3: Gap Inference Technique

the gap count of any unacknowledged chunk exceeds a predefined count (standard count is '3' in TCP), the chunk is considered lost.

Information about a chunk is removed from the WUQs under two conditions.

(i) When an incoming acknowledgement indicates that the receiver has received all chunks upto or beyond the TSN of the chunk under consideration.

(ii) When the incoming acknowledgement indicates that the receiver has flushed another chunk with a higher TSN to the upper layer at the destination.

Fig. 3.3 illustrates with an example the technique by which true gaps alone are identified in the WUQs. The figure shows the WUQs at the sender's end. Due to the relative delay differences across the paths, the receiver has received all the chunks dispatched so far on path-1 (chunks 100 through 107). Chunks 101, 104 and 106 had been dispatched on path-2 but 101 has been dropped by the network. Chunk 104 has reached the destination. Chunk 106 is still on the way to the destination. Chunk 102 is on its way to the destination on path-3. Reception of chunks 103, 104, 105 and 107 at the receiver would generate acknowledgements indicating that 102 has not arrived. This is a spurious gap report as far as 102 is concerned and should not be wrongly inferred as congestion on path-3. Similarly, reception of chunks with

TSN 107 or higher on path-1 does not have any bearing on 106 although the receiver would still generate gap reports for 106 until it receives 106. The only true gap is 101 because 104, which was dispatched on the same path as 101 has reached the destination. Hence, by tracking the chunks on separate WUQs, spurious gap reports are discarded.

3.1.4 Receiver Design

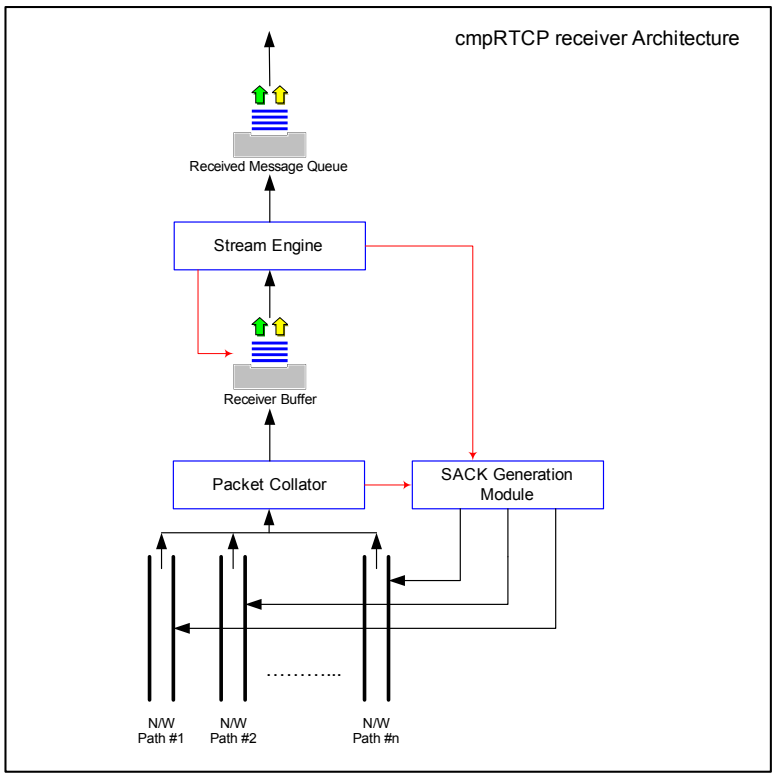

Figure 3.4: Block Diagram of receiver side cmpRTCP

Fig. 3.4 depicts the architecture of the cmpRTCP receiver. The receiver functions by imposing a *real-time delay tolerance limit* (RTDTL) on all the arriving packets. As the packets arrive from the network, the packet collator ensures that late arriving packets beyond the RTDTL (100 ms in our experiments) are forcefully dropped. Packets that do manage to get admitted are placed in a receiver buffer where they wait to be picked up by the stream engine. The stream engine takes care of reordering and defragmentation of the messages by waiting for missing chunks in the receiver

buffer as long as the RTDTL after which the packets waiting in the buffer are flushed to the upper layer before they expire.

3.1.4.1 Receiver Buffer Management

The receiver buffer holds time sensitive information i.e. the data packets in the buffer become worthless if the total time spent by them in transit and in the buffer is more than the RTDTL. However, not every packet can be flushed to the upper layer as and when it queues up in the buffer because ordered delivery to the upper layer is a requirement of the protocol.

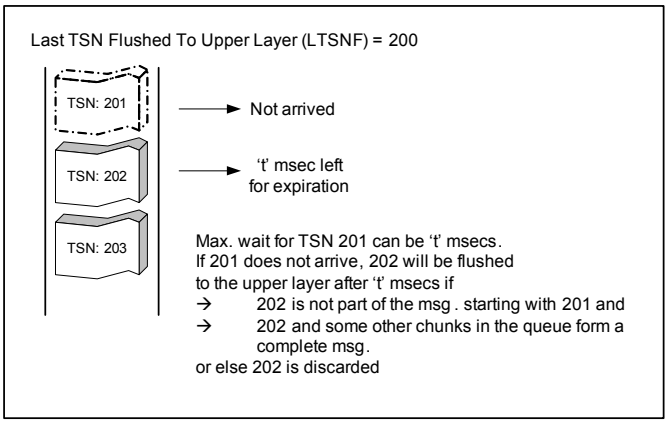

Figure 3.5: Message Flushing Mechanism at Receiver

When packets arrive at the destination over multiple network paths, a packet with a lower TSN may take longer over one network path while a packet with a higher TSN may have already reached the receiver buffer. The receiver cannot wait for the packet with the lower TSN indefinitely because the lifetime of the packet with the higher TSN in the buffer would run out. The maximum duration of waiting for a packet can only be as long as the time left for the first packet at the head of the receiver buffer to expire. After this duration, the first packet in the buffer is flushed to the upper layer and the sender is notified about the last TSN flushed to the upper layer via a *selective acknowledgement* (SACK) packet. The message flushing mechanism is illustrated in Fig. 3.5.

3.1.4.2 Acknowledgement Generation

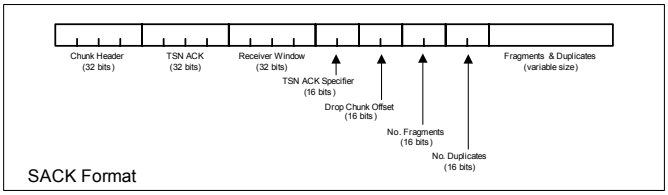

Figure 3.6: Format of SACK generated at the receiver

The TSN ACK field can hold one of two values; either the cumulative TSN acknowledgment (CTSNA) or the last TSN flushed to upper layer (LTSNF). A CTSNA indicates that all packets upto and including the TSN in the TSN ACK field have been received at the receiver. An LTSNF indicates that the TSN in the TSN ACK field is the last TSN that has been flushed to the upper layer. To distinguish the CTSNA from the LTSNF, one bit of the TSN ACK Specifier field is used. If the receiver drops a late arriving packet, the TSN of the dropped packet is put into the Drop Chunk Offset field as an offset from the CTSNA / LTSNF.

In order to determine if it is the CTSNA that will be dispatched in the SACK packet or the LTSNF, the algorithm shown in Fig. 3.7 is used.

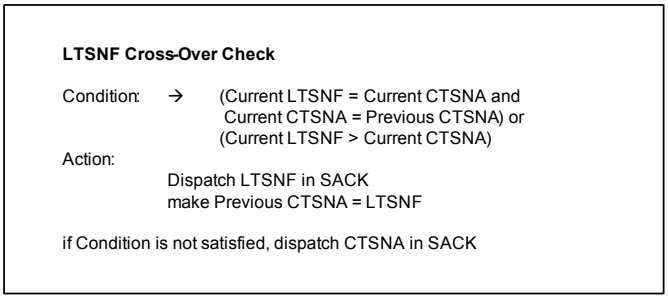

Figure 3.7: CTSNA / LTSNF Crossover Algorithm

Having the LTSNF and the *Crossover* algorithm is very crucial to the protocol. Without the LTSNF, during transmission, the CTSNA would have got locked at a particular TSN value soon after loss of a packet. Further loss during transmission would have led to an arbitrary growth in the fragment report, thereby increasing the SACK size. The size of the SACK would have reached a threshold (MTU) beyond

which the IP layer would have started fragmenting the SACK, which is undesirable. Thus the main purpose of using LTSNF is to enable the sender identify gaps in the reception sequence which might otherwise be masked. Also, the *Crossover* algorithm enables the CTSNA to keep advancing over the duration of the transmission.

Chapter 4

Performance Evaluation

4.1 Experimental Setup

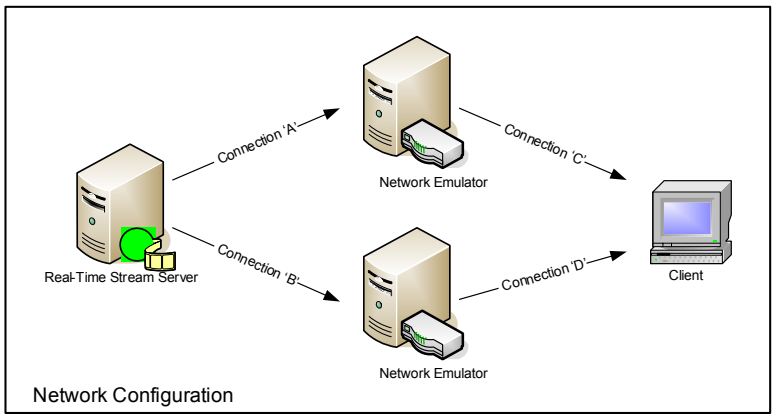

Figure 4.1: Block Diagram of the experimental setup

4.1.1 Network Setup

A block diagram of the experimental network setup is shown in Fig. 4.1. The setup includes four computers - a real-time video server, a client that receives the video and two other machines that serve as network emulators. As shown in Fig. 4.1, network connections 'A' and 'C' are interconnected and similarly network connections 'B' and 'D' are interconnected via the emulators. For simplicity one of the paths is referred to as path I (network path combining 'A' and 'C') and the other as path II (network

path combining 'B' and 'D'). The network emulators and the client are equipped with Pentium4 2.2 GHz processors with 512 KB cache and 512 MB of RAM. The video server has a Pentium4 2.8 GHz processor onboard with 1 MB cache and 512 MB of RAM. The network emulation was done using Nistnet 2.0.12c [6]. Experiments done for cmpRTCP were also repeated for its variant (cmpRTCPa). Experiments were also performed for an application level multi-path UDP real-time data streamer (AppStripe UDP) for contrast.

4.1.2 Video Encoding and Real-time Payload Generation

All experiments presented here involved transmission of a 10.2 MB pre-encoded H.264 stream of the foreman video clip (2098 frames; YUV 4:2:0) at CIF resolution - 352x288 at 25 fps with a GOP structure (12,3) and average I, P and B-frame sizes of 16.56 KB, 6.1 KB and 3.14 KB respectively. The H.264 encoding was done such that every encoded frame would be sliced and packed into *real-time payload* (RTP) packets, the size of each packet being close to 1200 bytes (less than a single MTU of 1500 bytes). The distribution of frame sizes is shown in Fig. 4.2. Fig. 4.3 shows the frame sizes of the clip, sorted in descending order of frame size.

4.1.3 Round Trip Delay and Bandwidth Constraints

4.1.3.1 Round Trip Delay

Round Trip Delay for a path also referred to as RTT (Round Trip Time) is the total amount of time that it takes for a packet to travel from the source to the destination along that path and for the acknowledgement from the destination to return back to the source. Typical RTT between hosts within the United States range from less than 10 ms to as large as 100 ms. In all experimental scenarios described, unless otherwise stated, the round trip delay has been set to 40 ms (20 ms in either direction).

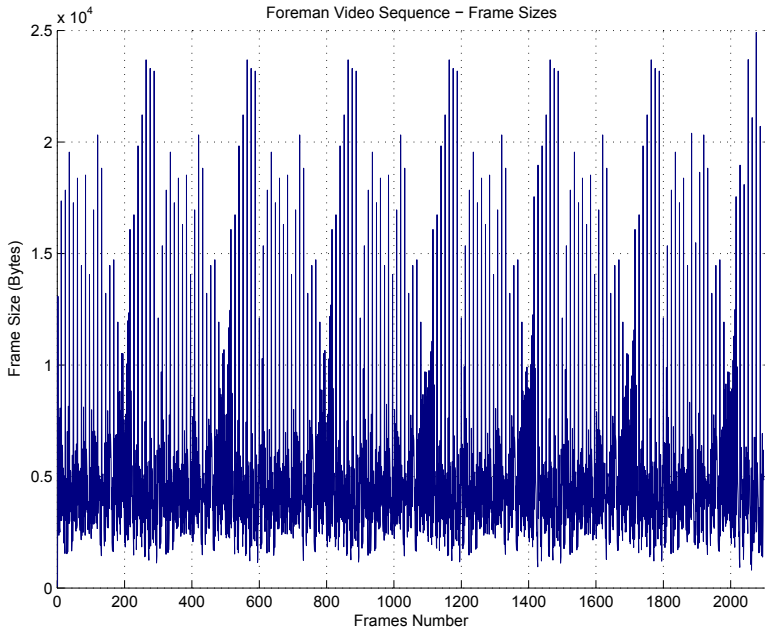

Figure 4.2: Size of Frames in the Foreman Video Sequence

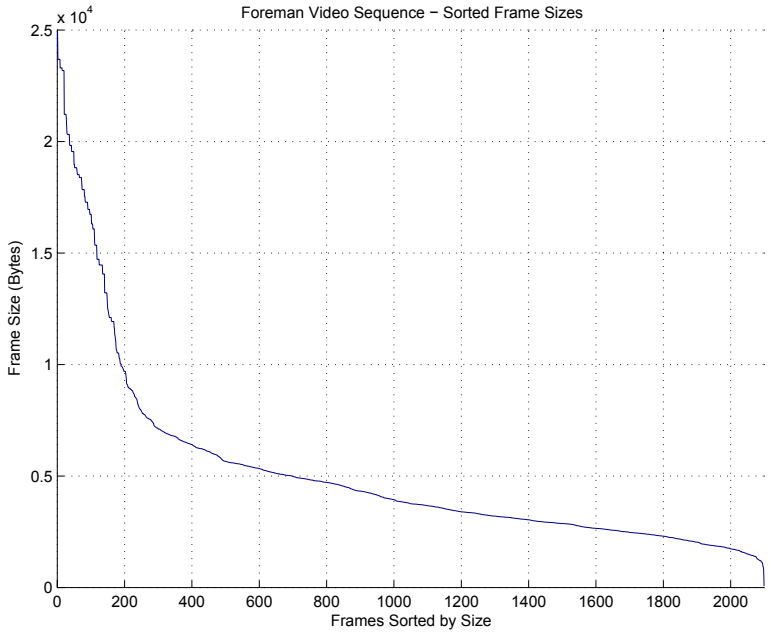

Figure 4.3: Sorted View of Frames in the Foreman Video Sequence

4.1.3.2 Bandwidth

Bandwidth of a path is the amount of bytes that can be transmitted on that path every second. Higher the bandwidth, faster the speed of transmission. For all the experiments described below, the amount of bandwidth alloted was unlimited.

4.2 Effect of Drop Rate Imbalance across paths

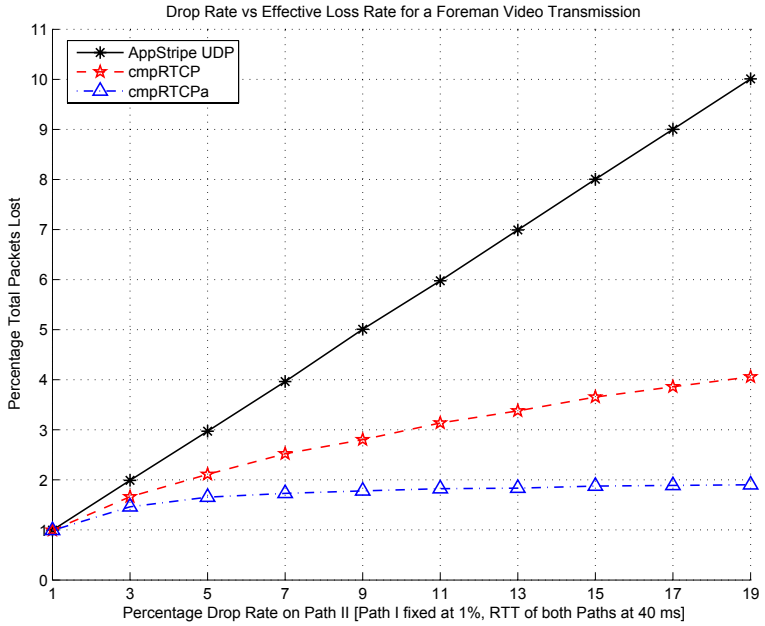

Figure 4.4: Comparison of Effective Loss Rates for Drop Rate Imbalance

The idea behind this experiment was to study the performance of cmpRTCP over networks that exhibit a drop rate differential across multiple paths that are available for data transmission. While setting the packet drop rate on path I at 1%, the drop rate on path II was varied from 1% to 19%. Fig. 4.4 contrasts the effective loss rate (percentage of packets lost) between cmpRTCP, cmpRTCPa and an Application level UDP streamer.

From Fig. 4.4, it is clear that as the drop rate differential across the paths increase, cmpRTCP and its variant perform increasingly better than the others. When the drop rate on path II is set to 19%, it can be seen that cmpRTCP shows a 60%

improvement while its variant cmpRTCPa, an even higher 80% improvement over UDP. The improvement can be attributed to the fact that cmpRTCP uses the packet drop detection mechanism to control the amount of data flowing through each path. cmpRTCPa takes it one step further and makes a best effort to choose a path with minimum number of missing packets during every round of transmission. The rapidly shrinking congestion window of the bad path with increasing drop rate differential is clearly seen in Fig. 4.5 and 4.6 for cmpRTCP and cmpRTCPa respectively.

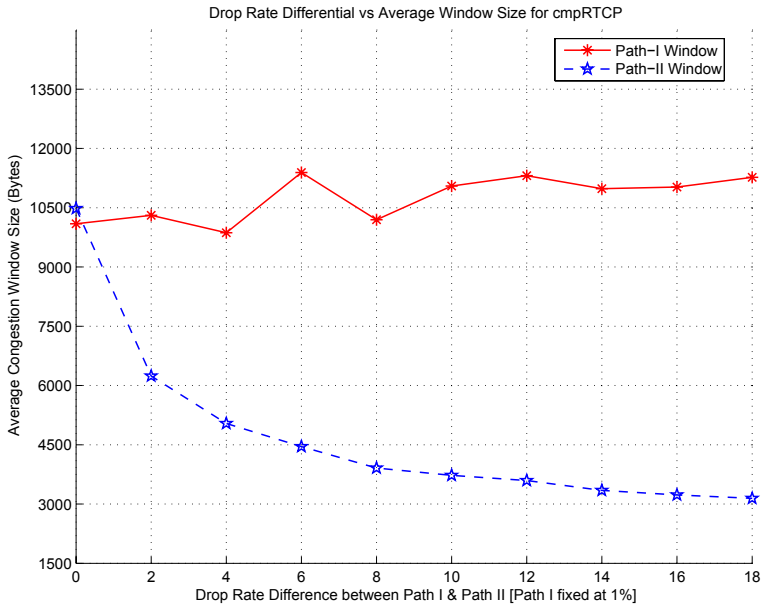

Figure 4.5: Congestion Window Plot for cmpRTCP

4.2.1 Comparison of Packet and Byte Loss Rates

When an IP packet is transmitted over a network, either the whole packet makes it to the other side or the whole packet is lost in transit. If each frame of the H.264 video stream were to be dispatched as a single IP packet, there would be no correlation between the total number of packets lost in the network and the actual amount of bytes lost. This is because H.264 streams are composed of I, P and B-frames whose sizes differ considerably. Thus the distribution of data over the sequence of frames

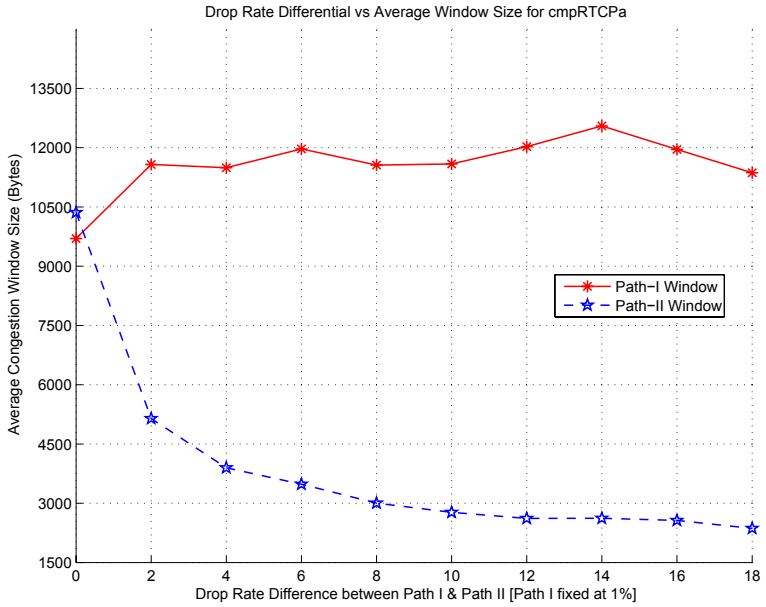

Figure 4.6: Congestion Window Plot for cmpRTCPa

is very non-uniform (loss of an I-frame could be the same as loss of 5 or more B-frames in terms of bytes lost). In addition to this, as the transport layer fragments the messages from the upper layer into chunks that are no more than the size of an MTU, loss of a fragment would render the rest of the fragments of a message useless to the upper layer at the receiving end. This raises a question about the plot in Fig. 4.4. Is it possible to infer the loss rates in terms of bytes? It is for this reason that the RTP packet sizes were restricted to less than a single MTU during their generation. This prevents the problem of fragmentation and also helps achieve an almost perfect correlation between the percentage of bytes lost and percentage of packets lost as shown in Fig. 4.7 (correlation coefficient = 0.998). Hence, the effective loss rates shown in Fig. 4.4 are the same irrespective of the metric (lost bytes or lost packets).

4.3 Effect of Round Trip Delay

Here, the idea was to study, compare and contrast the behavior of the various protocols under scenarios of various network round trip delays. A superior protocol would

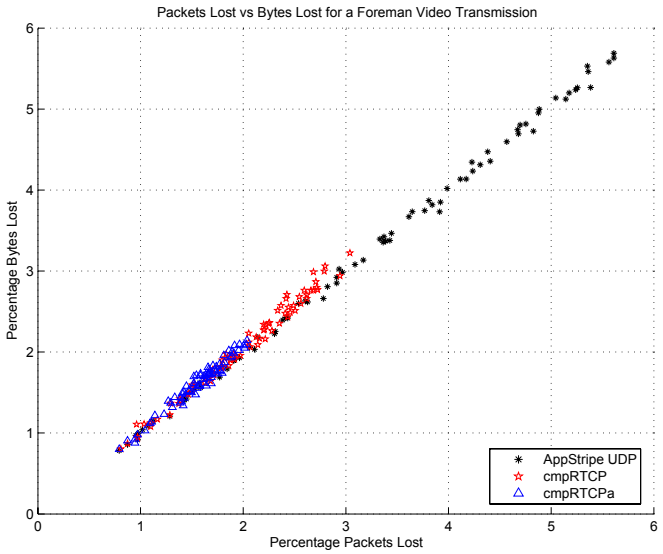

Figure 4.7: Byte and Packet Loss Correlation

be one whose deliverability does not get drastically affected by change in round trip delays.

4.3.1 Effect of Balanced Round Trip Delay

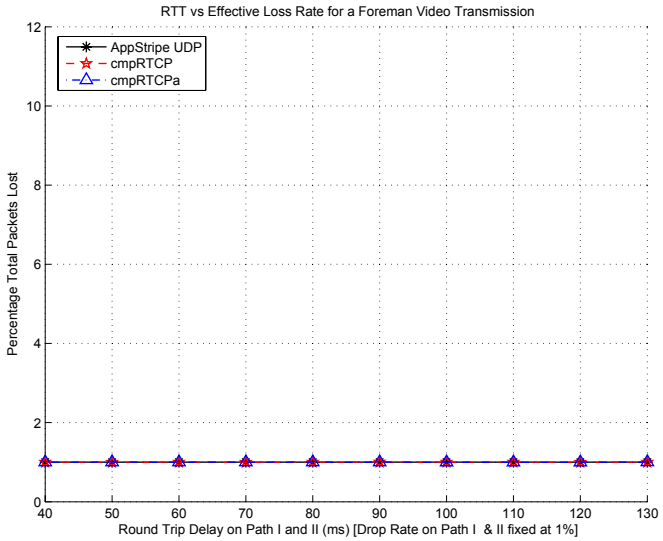

Figure 4.8: Comparison of Effective Loss Rates for Balanced Round Trip Delay

In this case, the drop rates on both paths were set to 1%. The round trip delay was varied symmetrically on both paths from 40 ms to 130 ms as shown in Fig. 4.8.

It can be seen in Fig. 4.8 that cmpRTCP and cmpRTCPa perform the same as the AppStripe UDP streamer delivering 99% of the packets to the destination on-time irrespective of the round trip delay within the range 40 - 130 ms. This can be explained considering the fact that both cmpRTCP and its variant do not retransmit any packets and equally schedule packets on both paths which leads to an average loss of 1% irrespective of the RTT.

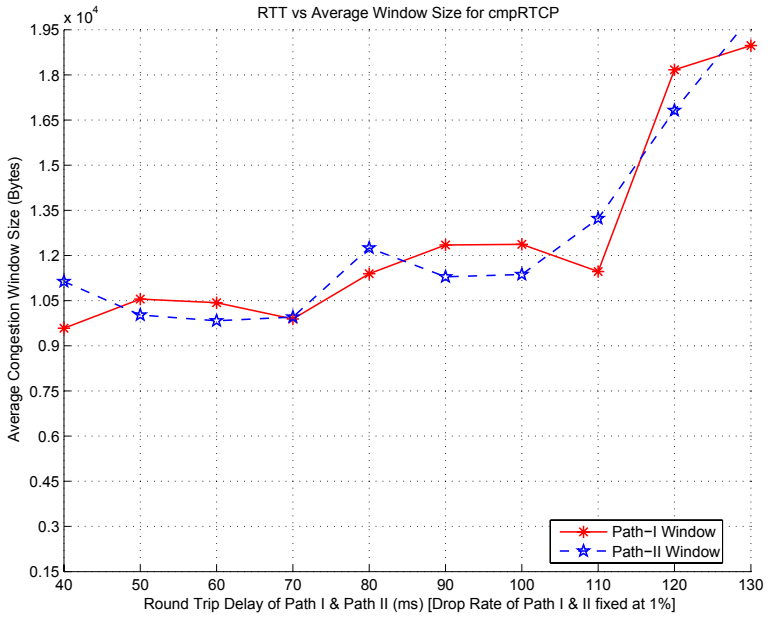

Figure 4.9: Congestion Window Plot for cmpRTCP

The individual congestion window plots for the protocols are shown in Fig. 4.9 and 4.10. It can be seen that cmpRTCP and its variant's window sizes increase with increase in round trip delay which is expected as there are more bytes in flight.

4.3.2 Effect of Unbalanced Round Trip Delay

Here, the round trip delay on one path was varied from 20 ms to 90 ms while holding the RTT of the other path at 90 ms. The drop rates on both paths were set to 1% similar to the scenario described in section 4.3.1. It can be seen in Fig. 4.11 that cmpRTCP and cmpRTCPa perform the same as the AppStripe UDP streamer delivering 99% of the packets to the destination on-time irrespective of the imbalance

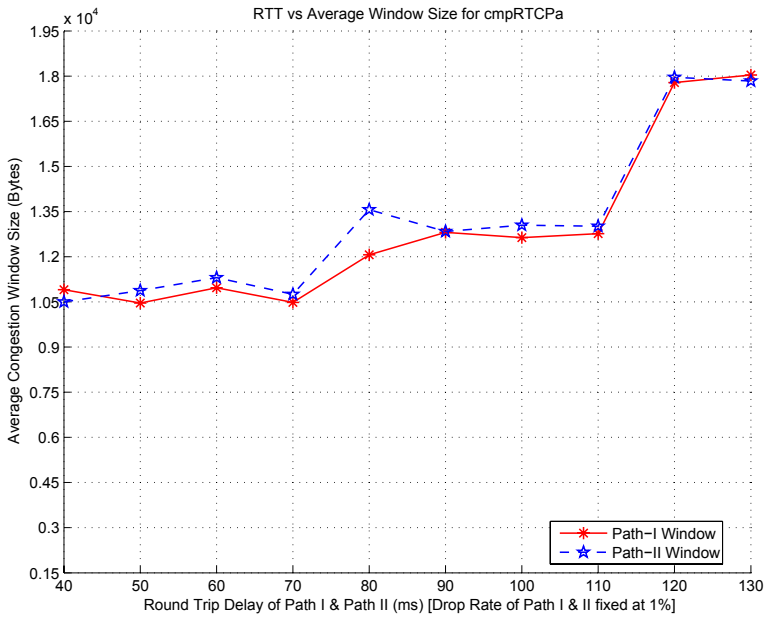

Figure 4.10: Congestion Window Plot for cmpRTCPa

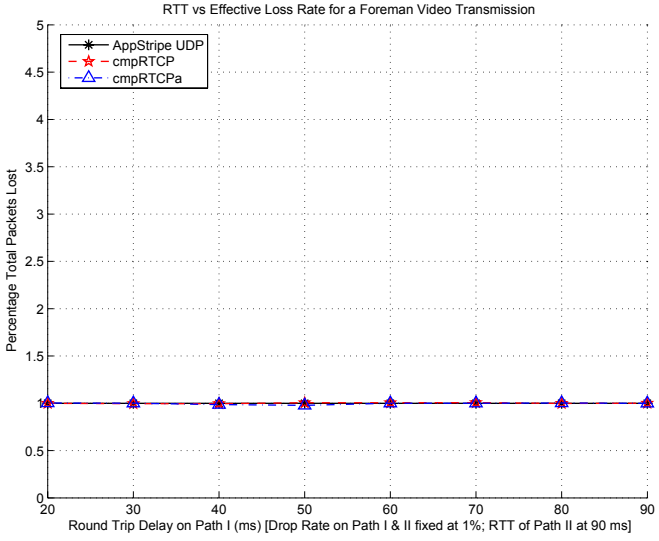

Figure 4.11: Comparison of Effective Loss Rates for Unbalanced Round Trip Delay

in the round trip delay across the paths. The reason is the same as in the previous section (refer section 4.3.1) for both cmpRTCP and its variant.

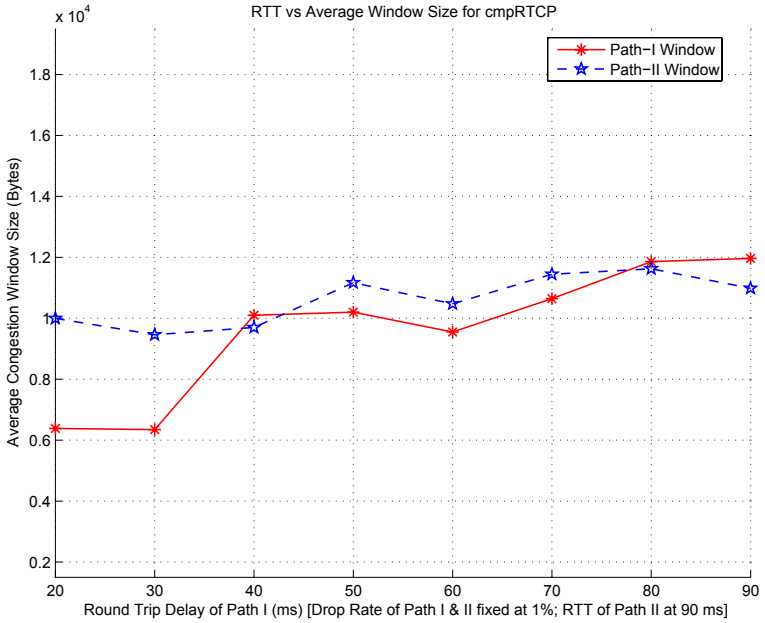

Figure 4.12: Congestion Window Plot for cmpRTCP

4.4 Effect of Drop Rate Fluctuation

Here, the packet drop rate on path I was set to 5% and the drop rate on path II was allowed to oscillate between 1% and 9% (which averages to 5%). The time period of oscillation was varied from 1 second to 19 seconds, incremented in steps of 1 second at the end of every run. RTT on both paths were set to 40 ms. Fig. 4.14 contrasts the effective loss rates of the protocols. UDP, not being aware of the fluctuations on path II, dispatches packets equally on both paths. This leads to the constant average drop rate of 5% that is observed, irrespective of the fluctuation. On the other hand, cmpRTCP and its variant are able to track the fluctuations and dispatch the packets appropriately to reduce the effective loss rate. As the time span of fluctuation increases above 2 seconds, cmpRTCP shows about 10% improvement over UDP and cmpRTCPa shows an improvement of about 20%. Shorter fluctuation spans do not

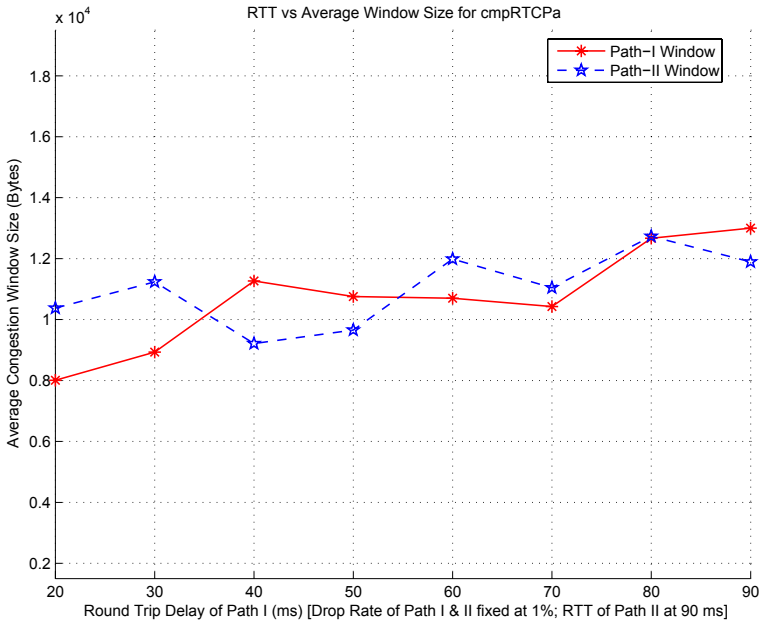

Figure 4.13: Congestion Window Plot for cmpRTCPa

give the protocol enough time to react and hence lead to slightly higher effective loss rates (which are still less than 5%).

4.5 Effect of Delay Variations

To look at the performance of the protocol under situations of network induced delay variations, the following experiment was performed. Path I was allotted a mean delay of 40 ms with a std. deviation [1] of 4 ms and a drop rate of 1%. Path II also had a drop rate of 1%. The mean delay in the forward direction on path II was varied from 20 ms to 160 ms with the std. deviation set to 20% of the chosen mean delay. The RTDTL was set to 100 ms for the experiment. The correlation for the std. deviation between successive packet delays was set to 0.8 on nistnet to minimize packet re-ordering within each individual path[5].

Fig. 4.15 shows that as the mean delay starts rising above 70 ms, all protocols start showing some amount of additional loss (in addition to the 1% drop rate on

[1] std. deviation(σ): standard deviation from the mean when using a standard normal distribution curve

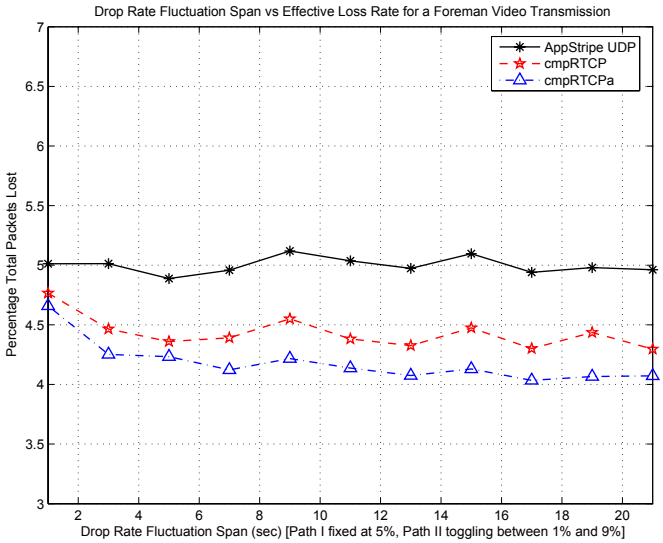

Figure 4.14: Effective Loss Rate Comparison for Fluctuating Drop Rates

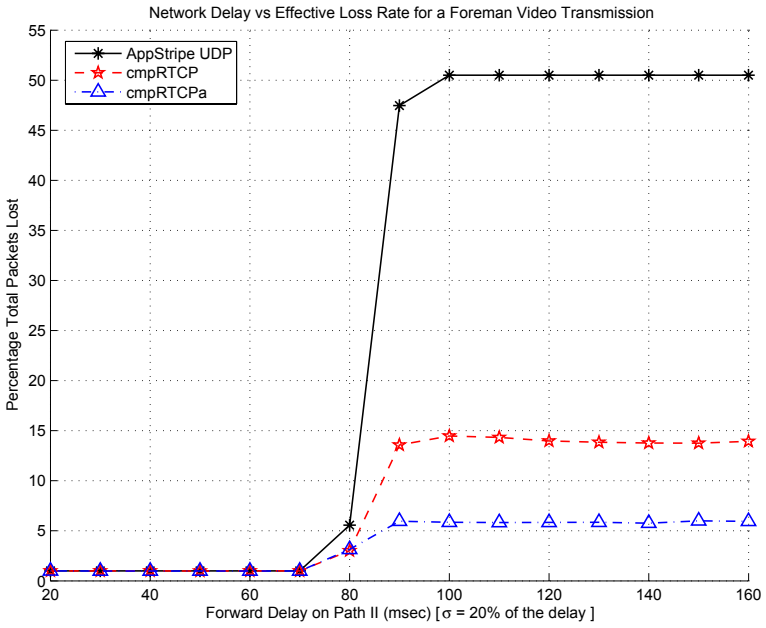

Figure 4.15: Effective Loss Rate comparison for Network Delay Variations

either path). While there is a substantial increase in the effective loss rate for UDP as the mean delay approaches the RTDTL (100 ms), cmpRTCP and its variant perform much better by maintaining a low effective loss rate. It can be seen that at a mean delay of 160 ms ($\sigma = 32$) where less than 3% of the packets transmitted (normal distribution) over path II would reach the destination under the RTDTL, an application streaming its data over multiple UDP connections manages to deliver only 50% of the total packets at the destination while cmpRTCP delivers 86% of the packets and cmpRTCPa delivers 94% of the packets. This clearly shows that delays in the network are promptly identified and the delay information used very effectively in cmpRTCP and its variant.

Hence the response of the window management scheme to delays in data reception at the destination is similar to data dropped in transit over the network thereby enhancing data delivery by using more of the better paths.

4.5.1 Effect of Scaling

Another experiment was conducted to look at the effect of scaling the number of paths to see how well the protocol is able to adapt itself. The packet drop rate on Path I was set to 1% (signifying a good path) and the number of bad paths were increased from 1 to 5 (drop rate on each bad path was set to 19%). Fig. 4.16 clearly shows that cmpRTCP scales well although there is an increase in the net effective loss rate with addition of more bad paths. This is expected because the protocol has to account for the fact that the bad paths may not continue to remain bad throughout the period of transmission. This effect of increase in the net effective loss rate is brought down further by cmpRTCPa because the algorithm tries to equalize the number of missing packets across all paths.

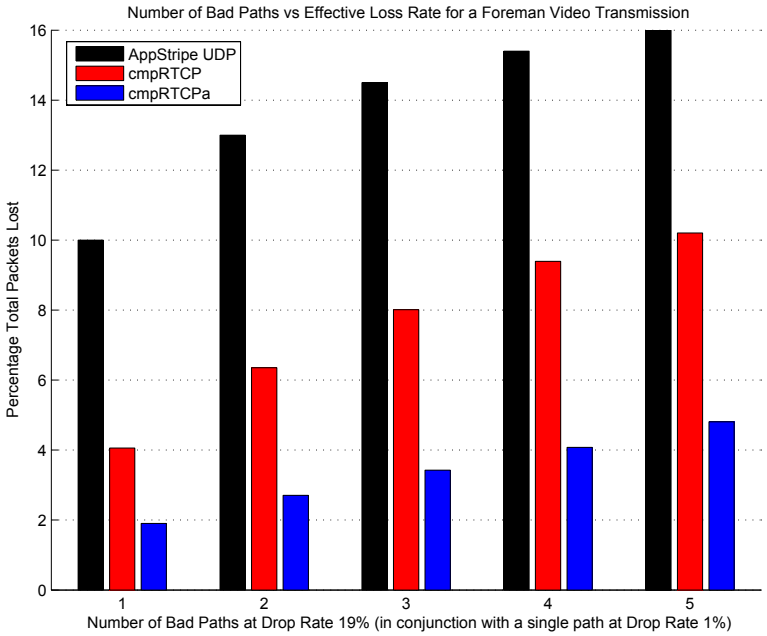

Figure 4.16: Effective Loss Rate comparison when introducing more bad paths

Chapter 5

Conclusion

The primary purpose of this work was to come up with a robust transport protocol for transmission of real-time streams between multi-homed hosts. In this paper, the advantage of having a protocol that makes use of a TCP like congestion controller work in conjunction with a packet scheduler to achieve substantial gains has been clearly highlighted. Studies and experiments have shown that this protocol is indeed capable of performing very well when streaming real-time video over IP-networks with fixed as well as varying drop rate and delay characteristics. Some of the important tasks ahead include performing an exhaustive study of the protocols performance under dynamic conditions of varying bandwidth, extending the protocol to support retransmission of select packets under application request, replacing the AIMD congestion controller with a more sophisticated bandwidth manager and developing an analytical model for the protocol.

References

[1] A. A. E. Al, T. Saadawi, and M. Lee. LS-SCTP: A bandwidth aggregation technique for stream control transmission protocol. *Computer Communications*, 27(10), 2004.

[2] A. E. Al, T. Saadawi, and M. Lee. A Transport Layer Load Sharing Mechanism for Mobile Wireless Hosts. In *IEEE PerCom 2004*, 2004.

[3] A. Argyriou and V. Madisetti. Bandwidth aggregation with SCTP. In *IEEE Globecom*, San Fransisco, CA, Dec 2003.

[4] S. Bangolae, A. P. Jayasumana, and V. Chandrasekar. TCP-friendly congestion control mechanism for a UDP-based high-speed radar application and characterization of its fariness. In *ICCS*, Singapore, Nov 2002.

[5] J. But, U. Keller, and G. Armitage. Passive TCP stream estimation of RTT and jitter parameters. In *LCN '05: Proceedings of the The IEEE Conference on Local Computer Networks 30th Anniversary*, pages 433–441, Washington, DC, USA, 2005. IEEE Computer Society.

[6] M. Carson and D. Santay. NIST Net: a Linux-based network emulation tool. *ACM SIGCOMM Computer Communication Review*, 33(3):111–126, 2003.

[7] C. Casetti and M. Meo. Westwood SCTP: Load balancing over multipaths using bandwidth-aware source scheduling. In *IEEE Vehicular Technology Conference*, 2004.

[8] J. Day and H. Zimmermann. The osi reference model. *Proceedings of the IEEE*, 71(12):1334–1340, Dec. 1983.

[9] M. Fiore and C. Casetti. An adaptive transport protocol for balanced multihoming of real-time traffic. In *Globecom*, 2005.

[10] A. Habib and J. Chuang. Multi-homing media streaming. In *IPCCC*, 2005.

[11] I. S. Institute. *Transmission Control Protocol (TCP).* RFC 793, September 1981.

[12] J. Iyengar, P. Amer, and R. Stewart. Concurrent multipath transfer using transport layer multihoming: Performance under varying band- width proportions. In *MILCOM*, Monterey. CA, Oct 2004.

[13] J. Iyengar, P. Amer, and R. Stewart. Retransmission policies for concurrent multipath transfer using SCTP multihoming. In *ICON*, Singapore, Nov 2004.

[14] J. Iyengar, P. Amer, and R. Stewart. Receive buffer management for concurrent multipath transport using SCTP multihoming. Technical Report TR2005-10, CIS Dept, University of Delaware, Jan 2005.

[15] J. Iyengar, P. Amer, and R. Stewart. Concurrent Multipath Transfer Using SCTP Multihoming Over Independent End-to-End Paths. *Networking, IEEE/ACM Transactions on*, 14(5):951–964, Oct. 2006.

[16] J. Iyengar, K. Shah, P. Amer, and R. Stewart. Concurrent multipath transfer using SCTP multihoming. In *SPECTS*, San Jose, California, July 2004.

[17] Z. Lifen, S. Yanlei, and L. Ju. The performance study of transmitting MPEG4 over SCTP. In *Neural Networks and Signal Processing, 2003. Proceedings of the 2003 International Conference on*, volume 2, pages 1639–1642, dec 2003.

[18] S. Mao, D. Bushmitch, S. Narayanan, and S. S. Panwar. MRTP: A multi-flow realtime transport protocol for ad hoc networks. In *IEEE Vehicular Technology Conference*, Orlando, Florida, Oct 2003.

[19] M. Molteni and M. Villari. Using SCTP with partial reliability for MPEG-4 multimedia streaming. In *Proc. of BSDCon Europe*, Netherlands, Nov 2002.

[20] P. Papadimitriou and V. Tsaoussidis. End-to-end congestion management for real-time streaming video over the internet. In *IEEE Globecom*, San Fransisco, CA, Nov 2006.

[21] D. S. Phatak and T. Goff. A novel mechanism for data streaming across multiple IP links for improving throughput and reliability in mobile environments. In *IEEE INFOCOM*, New York, NY, June 2002.

[22] J. B. Postel. *The User Datagram Protocol (UDP).* RFC 768, August 1980.

[23] D. Sarkar. A concurrent multipath TCP and its markov model. In *ICC*, Istanbul, Turkey, June 2006.

[24] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson. *RTP: A Transport Protocol for Real-Time Applications.* RFC 3550, IETF, July 2003.

[25] R. Stewart, M. Ramalho, Q. Xie, M. Tuexen, and P. Conrad. *Stream Control Transmission Protocol (SCTP) Partial Reliability Extension.* RFC 3758, May 2004.

[26] R. Stewart, Q. Xie, K. Momeault, C. Sharp, H. Schwarzbauer, T. Taylor, I. Rytina, M. Kalla, L. Zhang, and V. Paxson. *Stream Control Transmission Protocol.* RFC2960, IETF, Oct 2000.

[27] H. Wang, Y. Jin, W. Wang, J. Ma, and D. Zhang. The performance comparison of PR-SCTP, TCP and UDP for MPEG-4 multimedia traffic in mobile network. In *ICCT*, April 2003.

Vita

Anand Jayaraman was born in Bangalore, India, on May 24, 1980. He completed his elementary and secondary education in Kendriya Vidyalaya NAL in Bangalore, India. In October 1997 he entered the Bangalore University from which he graduated with a BE degree with *distinction*, majoring in Computer Science & Engineering, in August 2001. He worked with Microsoft R&D as Program Manager for the Windows System Resource Manager from November 2001 to October 2003. In August 2004 he was admitted to the Graduate School of the University of Miami where he was granted a Master's degree in Computer Science in December 2007.