ADAPTIVE BEST EFFORT PROTOCOLS FOR VIDEO DELIVERY

A Thesis

by

ASHWIN RAJ MADHWARAJ

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

August 1998

Major Subject: Electrical Engineering

ADAPTIVE BEST EFFORT PROTOCOLS FOR VIDEO DELIVERY

A Thesis

by

ASHWIN RAJ MADHWARAJ

Submitted to Texas A&M University
in partial fulfillment of the requirements
for the degree of

MASTER OF SCIENCE

Approved as to style and content by:

A.L.Narasimha Reddy (Chair of Committee)	
P. Cantrell (Member)	S.P. Bhattacharyya (Member)
R. Bettati (Member)	C. Singh (Head of Department)

August 1998

Major Subject: Electrical Engineering

ABSTRACT

Adaptive Best Effort Protocols for Video Delivery. (August 1998)

Ashwin Raj Madhwaraj, B.E., Bharatidasan University

Chair of Advisory Committee: Dr. A.L. Narasimha Reddy

This thesis describes the design and implementation of transport level protocols running on top of the IP layer to transmit video data over packet switched networks. These protocols address the issues of meeting real-time deadlines of the video data when transmitted over best-effort networks like the Internet. The protocols are end-to-end, and do not propose any modifications (like reservations) to the network infrastructure - hence, they can be used over the existing Internet. The design emphasis is on trying to obtain the best quality of video reception possible in the absence of any Quality of Service guarantees from the network. We have considered two kinds of media - Stored Media and Live Media. Stored Media is data stored in a server and transmitted to a client whenever requested (like a video file stored in a video server). Live Media is the live transmission of events; the video server retrieves data from a live source and transmits it to the clients. The performance of the protocol is measured by simulations.

To my parents

ACKNOWLEDGMENTS

First of all, I would like to thank my advisor Dr. Reddy for his guidance and advice throughout this research. In addition to funding me for the research, he gave a lot of ideas and direction. He was always open to questions and doubts and patiently answered them. He created a very open learning environment and encouraged free exchange of views, opinions and criticisms. The weekly group meetings, which were a regular feature, helped me enhance my knowledge in related areas of research. I will always be indebted to him.

I would like to thank my committee members for their attention and suggestions in this research. I would also like to thank the students in the Computer Engineering group for maintaining a very friendly and intellectual environment during graduate study. I will cherish the time I have spent with them.

I would also like to thank my parents for their continuous support, encouragement and motivation. They believed in my abilities and helped me build confidence. This would have been impossible without their support.

TABLE OF CONTENTS

CHAPTER		Page
I	INTRODUCTION	1
	A. Motivation	1
	B. Related work	5
	C. Organization of the thesis	7
II	BEST EFFORT PROTOCOLS	8
	A. End-to-end solution	8
	B. Multimedia server	9
	C. Smooth transmission	12
	D. Prioritization	13
	E. Estimation of the available bandwidth	13
	F. Packet pair technique	14
	G. Desired bandwidth and transmitted bandwidth	16
	H. Rate controlled adaption	17
	I. How "nice" is the protocol?	18
III	LIVE MEDIA AND STORED MEDIA PROTOCOLS	19
	A. Live media	19
	1. Live media sender	20
	2. Live media receiver	21
	B. Stored media	22
	1. Stored media sender	24
	2. Stored media receiver	25
IV	SIMULATIONS AND RESULTS	27
V	CONCLUSIONS	39
REFERENCES		41
VITA		44

LIST OF TABLES

TABLE		Page
I	Natural congestion - loss percentages	29
II	Simulated congestion - loss percentages	29
III	Loss percentages with multiple parallel sessions of UDP and FTP . .	32
IV	Loss percentages with parallel UDP streams (with intervals)	36
V	Effect of smoothing - loss percentages	37

LIST OF FIGURES

FIGURE		Page
1	Variable Bit Rate	4
2	Protocol stack	9
3	Scope of the protocol	10
4	Effect of smoothing transmission	11
5	Packet pair technique	15
6	Cushion in the stored media protocol	23
7	Sending/received rates for live media protocol	30
8	Sending/received rates for stored media protocol	31
9	Multiple live media protocol streams	33
10	Multiple stored media protocol streams	34
11	Experiment with different packet sizes	38

CHAPTER I

INTRODUCTION

A. Motivation

The recent spurt in the growth of real-time multimedia applications on the Internet has thrown light on the inadequacies and restrictions of the existing Internet. The Internet is a packet switched network and was designed primarily for data traffic and is not suited for real time transmissions of audio or video. The Internet Protocol (IP) treats all packets equally - and does not know about the type of data contained in the packets, nor about the flow which the packet belongs. There is no bound on the delay suffered by a packet and the bandwidth available on the Internet. The routers on the Internet provide only a best-effort service to the higher layers, and do not support any kind of Quality of Service (QoS) guarantees.

However, multimedia data (like video and audio streams) have specific real-time deadline requirements. For example, data pertaining to a particular frame in a video stream has to be received by the receiver before that frame is played back. If the data arrives after the frame is played back, that data is no longer useful. Additionally, video files tend to be very large in size and this imposes bandwidth requirements which the Internet should provide. However, the bandwidth available on a particular path on the Internet depends on the number of routers on that path, behavior of each router, and number of packets crossing each router at the time of transmission. Thus, the behavior of the Internet is highly dependent on the time of the day and the number of users on the Internet.

The Internet infrastructure, based on IP routers, has already been established

---

The journal model is *IEEE Transactions on Automatic Control.*

providing only a limited amount of services to the upper layers. For video/audio transfer over the Internet, we can have either an end-to-end transport layer protocol or we can modify the routers on the path from the video server to the client to provide more real-time services. For a general purpose protocol which can be potentially used on any two machines over the Internet, the latter solution is not practical. There has been a lot of work going on to figure out what kind of services the network should provide in order to guarantee good quality reception of video/audio data. A few examples are RSVP [1] and ATM based guarantees. However, all these fall into the latter category, and require modification of the current Internet infrastructure. Though these can be treated as long term solutions, end-to-end efforts work well in the existing Internet without needing any modification to the existing router infrastructure. The aim of the proposed end-to-end best-effort protocol for multimedia servers is not to ensure guaranteed delivery, and not to conform to any QoS specifications, but to try to do the best in the current situation in terms of packet delivery and latency.

Video files tend to be very large in size (of the order of hundreds of Megabytes to Gigabytes). If the client who is requesting the video has a large enough buffer, the whole video file can be down-loaded from the video server and then played back. This approach, which was followed till recently on the Internet, has two main problems. First, there is an initial latency where the receiver has to wait for the video data to be received before playback can start. This tends to be large for large video files. Also, based on the time of the day, and the traffic, the time of down-load can vary enormously. Second, this approach requires a very large buffer on the receiver side. This is not practical in the Internet, since there are a multitude of machines of vastly differing processing powers, memory size, network connection speed, etc.

The solution to this problem is to stream the media data. This means that

while the receiver is receiving the data, it is playing back data received earlier. This approach requires a much smaller buffer on the receiver side, does not depend on the length of the video/audio stream, and has a relatively small initial latency. However, in this approach, the time of reception of the video data becomes important as they have deadlines to be met. The receiver has a limited amount of buffer and the play back has to go on smoothly. Since the buffer is limited, the playback and reception of packets cannot go very "far" from each other; while to ensure smooth playback, the reception of packets should be as far as possible from the playback to ensure that all the packets are received in time.

The Internet is a rapidly expanding network. The number of users is enormous and is growing exponentially every year. This causes a lot of stress on the infrastructure and reduction of quality for the users. Depending on the time of the day, number of users currently on the network, and the usage of the network, the bandwidth available between a specific source and destination can vary randomly over time. Video files (e.g. MPEG compressed video files) are Variable Bit Rate (VBR) streams - the bandwidth requirement of a particular stream varies deterministically over time. Our protocol deals with the issue of providing a deterministically varying required bandwidth over a randomly varying available bandwidth.

A VBR source produces highly bursty data and has very complex traffic characteristics. The burstiness of the data makes it difficult to determine the amount of resources required by the sender to transmit the data over the network. The bandwidth utilized by the VBR stream will vary depending on the highly variable bit rate of the stream. This variability of data rate of the input stream in addition to the high variability of the available bandwidth of the network adds to the complexity of handling video data. Fig. 1 shows the variance of the number of bits with respect to frames of a typical VBR movie (this figure shows the number of bits in the first 400

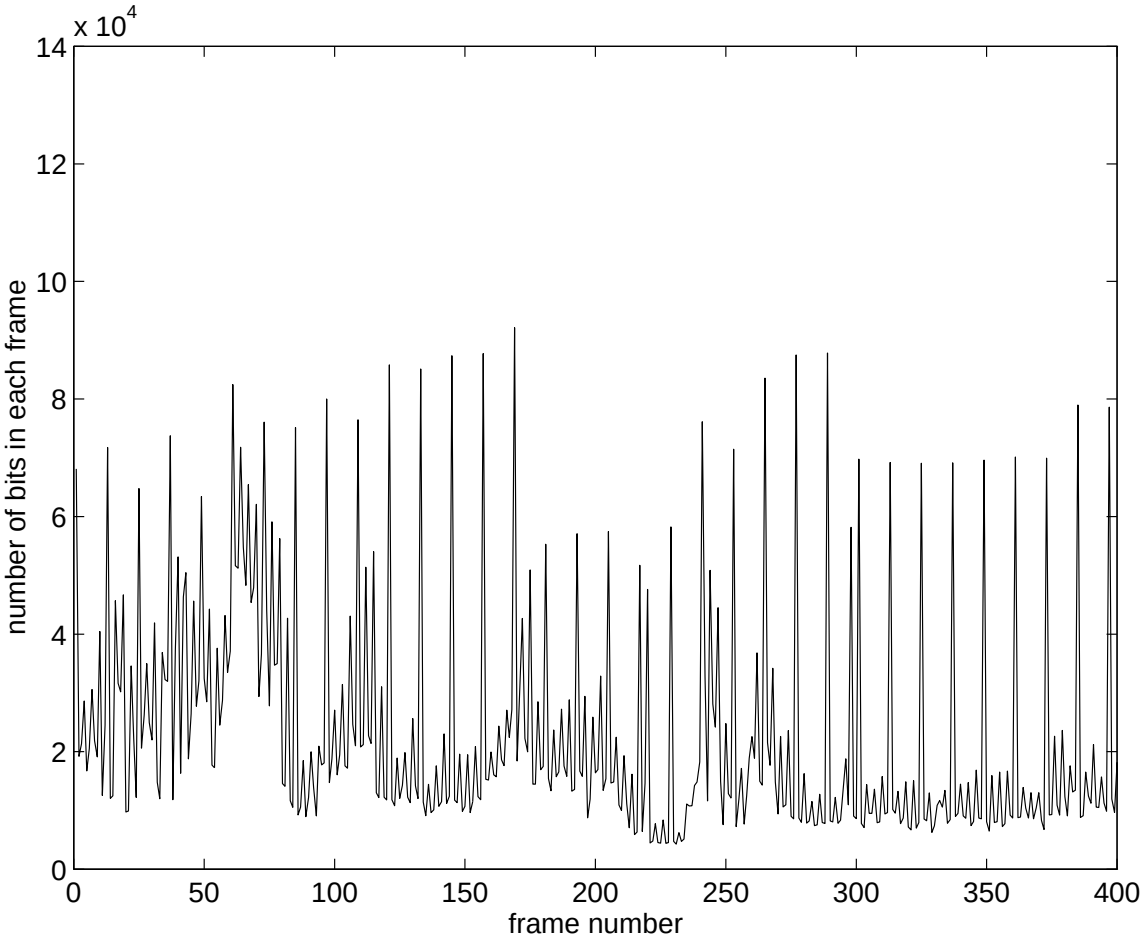

Fig. 1. Variable Bit Rate

frames in an MPEG compressed *Simpsons* movie [2]).

In Fig. 1, the peaks correspond to the I frames, while the values between succeeding peaks correspond to the P and B frames. The I, P and B frames are the three types of frames in the MPEG standard. The I frames are reference frames and are encoded without any compression (i.e., each pixel/sample is represented by a fixed number of bits representing the hue, intensity, color of the pixel/sample). The P and B frames are difference frames and contain only the difference between the preceding/ succeeding I frames. This works well with video, since adjacent frames have a lot of correlation and if encoded completely, information is redundantly stored and this leads not only to more storage requirement, but also, more data to be transmitted from the server to the client.

This thesis proposes an end-to-end streaming solution (transport layer protocol) for Variable Bit Rate streams. The proposed protocol uses network feedback to control congestion, takes care of the required bandwidth of the MPEG input data stream, the variable available bandwidth of the network, the buffer capacity of the receiver, and other issues dealt with in Chapter II.

B. Related work

Brian Smith proposed a best effort protocol, Cyclic UDP, [3] for media transfer over the Internet. Cyclic UDP uses prioritized MJPEG packets and a delay based feedback mechanism to improve the delivered quality of video over Internet. Measurements on the Internet have shown that Cyclic-UDP adapts to congestion well, but the behavior in the presence of other applications is not studied. It has been shown that the prioritization can also be done on MPEG video [4].

Jeffay et al. [5] have also proposed best effort protocols. They have proposed a

facility for varying the synchronization between displayed audio and video to achieve high-fidelity audio, and a network congestion monitoring mechanism that is used to control audio/video latency and dynamically adapt the frame rate to the bandwidth available in the network.

Kanakia et al. [6] have proposed a scheme in which explicit feedback information from the network is used to control the data generation rate of a video source. Their simulations show that even though the network feedback information is available only after a significant delay, the perceptual quality degrades gracefully.

Schulzrinne et al. [7] proposes the Real Time Protocol which provides some services for real time applications like timestamping, sequence numbering and feedback from receiver about quality of data being received. The Real Time Control Protocol (RTCP) can be used to monitor the data delivery and provide feedback, while RTP is used for the data transfer services.

McCanne and Jacobson [8] have worked on providing a framework for real time applications. They developed a Video Conferencing tool, *vic*, which was independent of the network layer, supported hardward based codecs, and supported diverse compression algorithms.

There have been some commercial real-time streaming audio/video applications lately on the Internet. Companies like Real Networks and VXtreme (Microsoft) have come up with streaming audio/video applications which run over the Internet. The High Speed Networking group at INRIA labs have developed *FreePhone*[1] an audio tool for the Internet. These tools use the facilities of RTP/RTCP predominantly to transfer media data over the Internet. The low quality reception using these tools reiterates that streaming technology is still in its infancy and lots of work remains to

[1]http://www.inria.fr/rodeo/fphone

be done in future.

C. Organization of the thesis

The rest of the thesis is organized as follows. Chapter II discusses the issues we have dealt with and the solutions we have proposed. Chapter III explains the live media and stored media protocols in detail. The simulation environment and the results are described in Chapter IV. Chapter V concludes the thesis.

CHAPTER II

BEST EFFORT PROTOCOLS

A. End-to-end solution

In order to meet our primary requirement of a simple ready-to-use protocol, an end-to-end solution is proposed. This means that there is no modification needed at the router level and hence, the current router infrastructure can be used as such.

Typical end to end solutions are the transport level protocols like UDP and TCP. These protocols provide various features and functionalities which are used by applications requiring different services. UDP [9] is a simple and thin protocol and has two main functions - multiplexing and checksum. UDP multiplexes all data from different applications (different ports) to the IP layer for transmission over the network. On the receiver side, it demultiplexes the data received to the different ports based on the UDP header. The checksum provides a way to the receiver to check whether the received packet is corrupted.

TCP [10] is a more complex protocol. It provides reliable delivery, this means that it has zero loss rate. It takes care of in-order delivery to the application at the receiver side via buffering and retransmissions. TCP is a streaming protocol and does not have any record capability. It is connection oriented, has flow control, involves congestion control, and is window based. TCP adapts its sending rate based on its notion of the congestion in the network - it treats all packet losses as due to congestion.

A transport level end-to-end protocol for video delivery should contain many of the above features, as well as additional functionality like some form of congestion control; taking care of real time deadlines of the data; some means of measuring

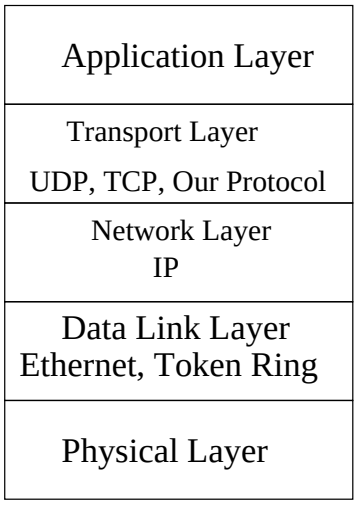

Fig. 2. Protocol stack

available bandwidth, and comparison with the required bandwidth; use the notion of prioritization; among other things.

The protocol described in this thesis supports all the above mentioned functionality, and is designed to be a transport layer protocol to provide an end to end solution as shown in Fig. 2.

B. Multimedia server

We assume that the multimedia server schedules service to all its clients in a periodic manner. That is, the sender periodically services each client by reading a sequence of video frames or audio samples from the disk (or other source - in case of Live Media) to a buffer. This buffer has to be transmitted to the client before the next service time for that client. The protocol has a given amount of data to be transmitted to the client in a fixed amount of time.

Fig. 3 shows the structure of a multimedia server and a client connected to

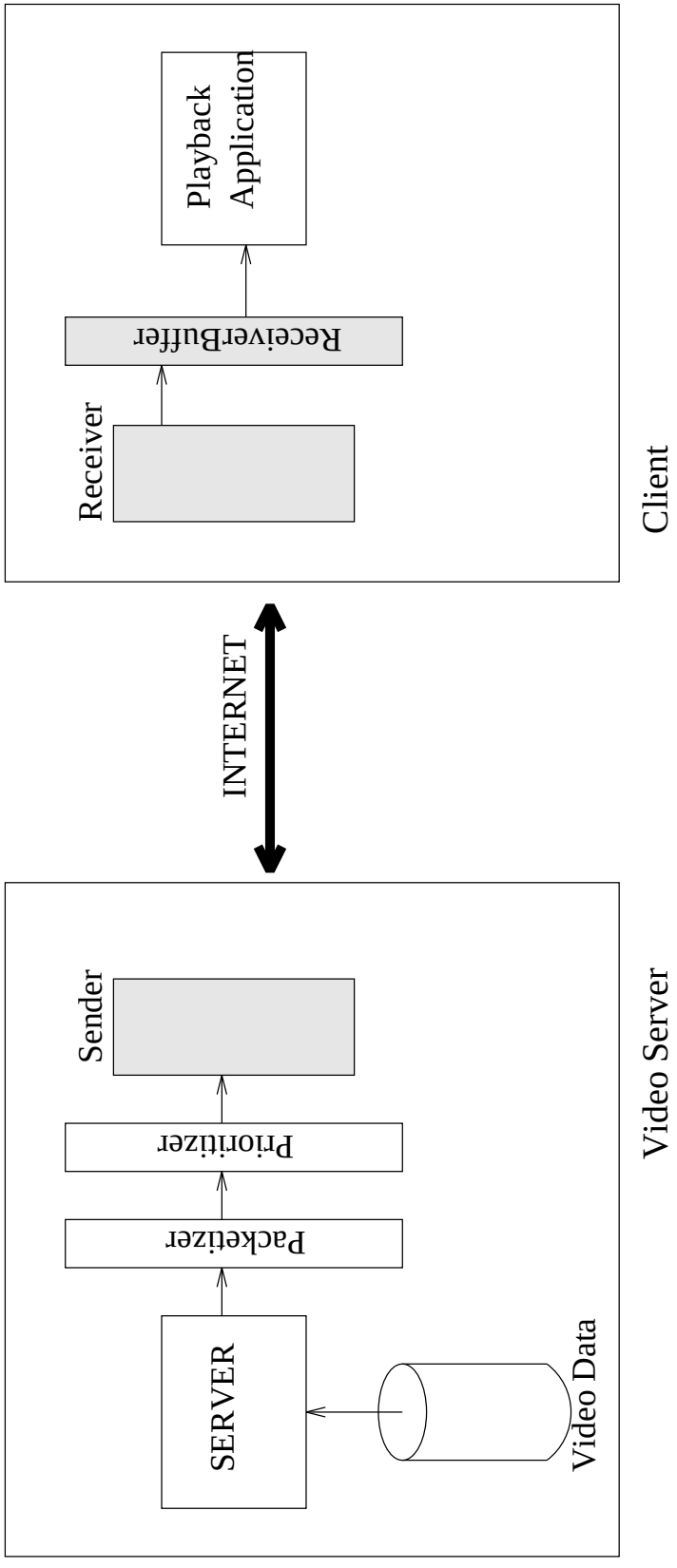

Fig. 3. Scope of the protocol

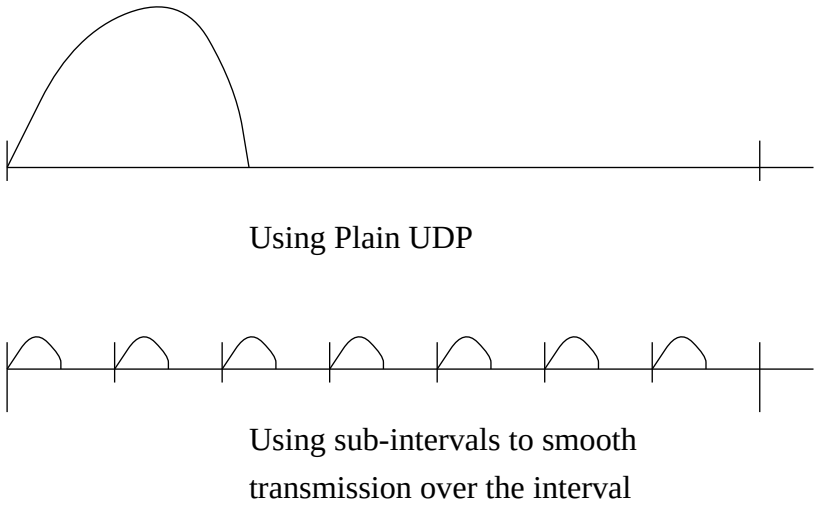

Fig. 4. Effect of smoothing transmission

it via the Internet. The video server contains the video data in the video file in single/multiple disks. The server application reads the video data from the disk, copies the data onto memory and then transfers it onto the packetizer which chops the video data into packets. The size of the packets depends on the packet size which will go unfragmented in the Internet and this is typically 536 Bytes. Sometimes, depending on the path taken, a figure of 1500 Bytes can also be used. The packets are then prioritized (section D) and then fed into the transport level sender protocol, which talks to the transport level receiver protocol on the client side via the Internet. The receiver protocol stores the received data in a sequence in the receiver buffer. The playback application reads the receiver buffer, decodes the data and displays the video.

C. Smooth transmission

As noted earlier, the multimedia server provides a given amount of data to the protocol to be transmitted in a certain amount of time. There are two ways of sending this data in the given time - send all the data in a burst in the beginning of the interval and deal with lost packets and retransmits in the remaining time in the interval; or smooth the transmission of data over the whole interval, i.e. send parts of the data across the entire interval in regular sub-intervals so that the burstiness is reduced as shown in Fig. 4.

Our protocol examines the effect of smoothing (to different extents) within a scheduling interval. Data is not sent in one chunk at the beginning of each interval - transmission is (as far as possible) uniformly distributed over the entire length of the interval. Our protocol examines the effect of the variation of the number of sub-intervals in the interval on the quality of received data.

This smoothing is different from the smoothing done to VBR data to reduce variance in load using various smoothing algorithms [11]. These algorithms exploit client buffering capabilities and determine a smooth rate transmission schedule, while ensuring that a client buffer neither overflows nor underflows. They try to reduce burstiness of the VBR data by modifying the sending rate - by sending data earlier than it should actually be sent. When the VBR rate is low, some data from future frames is added onto this frame, thus increasing the rate of the current frame, while decreasing the rate of future frames which have high rates. Since our protocol does not depend on the type of data it sends, it will support CBR data or smoothed VBR data.

D. Prioritization

Additionally, the data passed from the sender is passed through an encoder-specific *Prioritizer* which re-organizes the packets in terms of its importance. For e.g., for MPEG, data corresponding to I frames will be ordered on top and data corresponding to P and B frames will appear later (This is because, the I frames are "more important" than the P and B frames. If an I frame is lost, information about that frame as well as the adjacent P and B frames are lost since those are just difference frames. However, if a P or a B frame is lost, only information about that particular frame is lost). Our protocol will ensure that the probability of successful transmission of a packet depends on the priority of the data. In the above example, the probability of successful transmission of I frames will be greater than that of the P and the B frames, thus ensuring the best quality achievable at the receiver with the given loss-rate and congestion of the network. (Refer [3, 4] for details about prioritization). Kanakia et al. [6] have shown that prioritized transmission and delivery of video data provide better signal-to-noise ratios when network congestion is possible.

E. Estimation of the available bandwidth

Feedback from the network is used to estimate the amount of available bandwidth on the network and the transmission rate is adjusted based on this feedback. The available bandwidth of a particular path in the Internet depends on the current load of all the routers in the path, the number of users on the network, the type of data transfers (audio, video, text, image), and a number of such factors. Thus, the available bandwidth of the path is random is some sense. Though it is not possible to accurately determine the instantaneous available bandwidth over a period of time, it is possible to estimate the end-to-end bandwidth over the particular set of routers.

There are various techniques for estimating the bandwidth and all protocols which support congestion control have some method of estimating the bandwidth. TCP (including TCP Vegas) uses the number of packets transmitted by the sender between the sending of a marked packet and the reception of the acknowledgment for that marked packet as a measure of the amount of bandwidth available on the network. The proposed protocol uses a packet-pair technique [12] to estimate the available bandwidth of the Internet. Whatever the method of estimating the available bandwidth, any instantaneous measurement is not likely to give results of a good quality. A time-average is used with different weights to the history and the current sample to estimate the available bandwidth. Details about the packet pair technique are given in section F.

Protocols which support congestion control obtain network feedback as explained in the previous paragraph. This feedback is used to vary the sending rate - if the available rate decreases, the sending rate is decreased; and if the available rate increases, the sending rate is increased. The sending rate can be adapted based on the feedback as soon as change in network conditions is detected, or in a lazy manner (not so immediately). The merits and demerits of both these types of protocols are examined.

F. Packet pair technique

A packet-pair technique is used to generate feedback about the current congestion in the network and to get an estimate of the available bandwidth [12].

To measure the maximum bandwidth using the packet pair technique, the sender sends two marked packets back-to-back to the receiver (as shown in Fig. 5). These packets experience three different kinds of delay - transmission delay, propagation

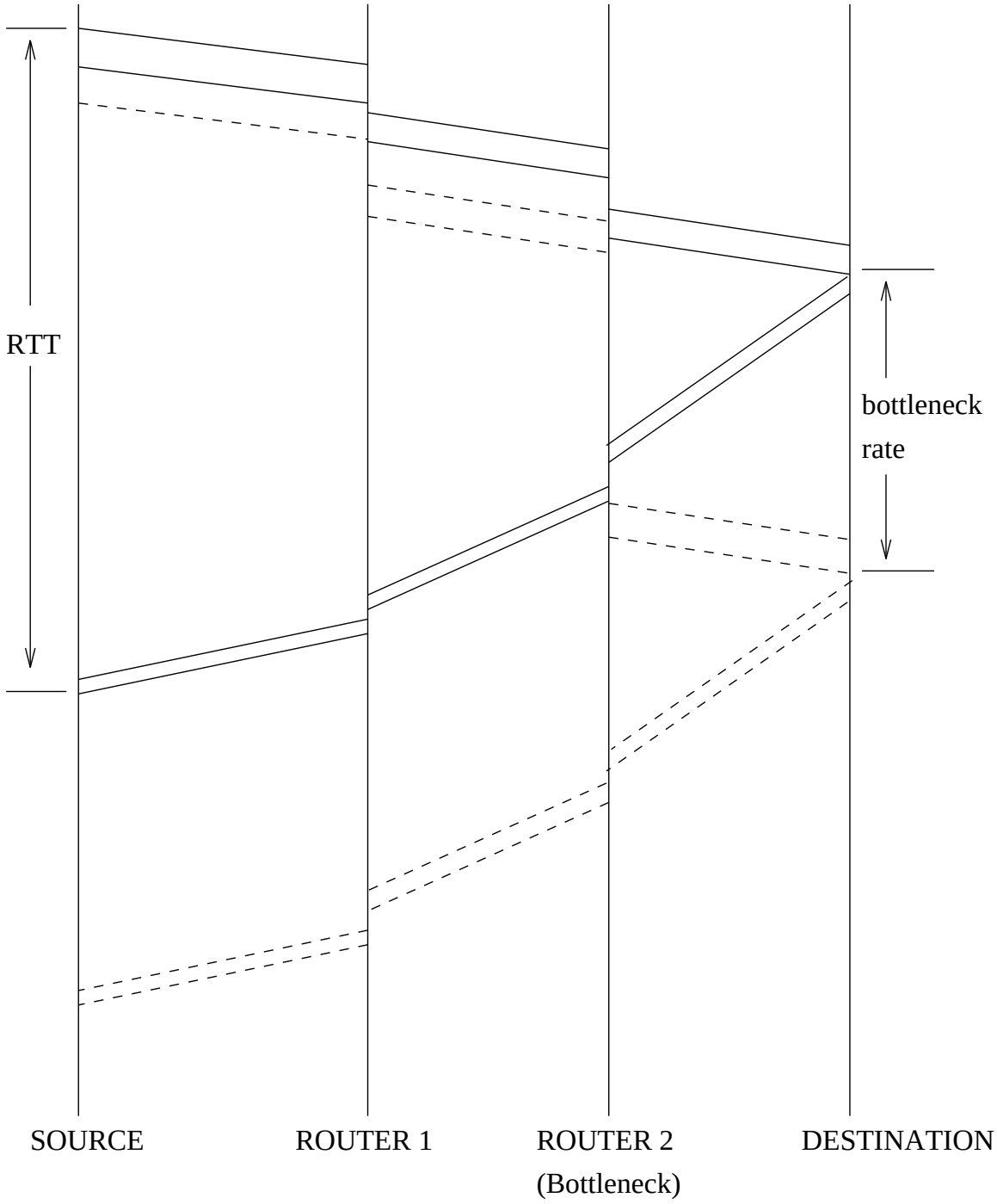

Fig. 5. Packet pair technique

delay and the queuing delay. The transmission delay is caused by the size of the packet; the propagation delay by the link; and the queuing delay by the service rate of the router. The bottleneck router is the one with the maximum service time. The two marked packets pass through the routers on the way and experience different amounts of delay and arrive at different points of time on the receiver. Let us call this value δ. This value δ gives an indication of the service rate of the slowest router in the route. Over a period of time, if this packet pair technique is repeated, the minimum value of δ will indicate the fastest service rate among the routers over that period of time. Hence, the size of the packet divided by the minimum value of δ over a period of time will give a good approximation of the maximum bandwidth of the network.

Some of the maximum bandwidth of the network will be used by the different flows in the network. Hence the $availableBW = maxBW - usedBW$. If u is the utilization of the network, then the available bandwidth can be given by $maxBW * (1 - u)$.

For a M/M/1 queue, the delay of the queue is proportional to $1/(1 - u)$, where u is the utilization [13]. Hence, available Bandwidth can be estimated as $maxBW * (min\_delay/delay)$. In practice, the sender can measure the Round Trip Time (RTT). Thus, the available Bandwidth can be estimated as $availableBW = maxBW * (minRTT/measuredRTT)$.

G. Desired bandwidth and transmitted bandwidth

Throughout the transmission, the sender maintains two rates - the rate which is desired by the application (the data in the queue should be transmitted within the given interval), called the *desired_rate*; and the estimated bandwidth of the network

obtained using the packet-pair mechanism called the *actual_rate*. The *desired_rate* is adjusted to be slightly greater than what is exactly needed by the video application to prevent unfairness to the packets at the end of the queue - the last few packets in the interval may not have sufficient time for error and congestion control and retransmission otherwise. The *actual_rate* gives a fairly accurate figure of the available bandwidth of the network. The protocol has to transmit at the minimum of these two rates. If the *desired_rate* is less than the available bandwidth of the network, it is sufficient to use the *desired_rate* to send the packets. Similarly, if the *actual_rate* is less than the *desired_rate*, then the sender is forced to transmit only at the actual rate to avoid risk of losing packets, generating large delays or increasing congestion in the network.

H. Rate controlled adaption

The protocols described in this paper involve rate-controlled adaptation as against window-controlled adaptation ([14], [15], [16], [17]). Window based adaptation protocols are more useful in the case of reliable transmission as in TCP. In TCP, the flow and congestion control between the sender and receiver is based on windowing, a window of packets are sent to the receiver before an acknowledgment is expected for the first one. The value of this window is based on the size of the buffer on the receiver side and, more importantly, the current congestion in the network. The number and pattern of packet losses detected by the sender decides the value of the congestion window. In one version of TCP, TCP-Vegas [17], the congestion window is calculated based on the current expected rate of the network and the actual rate observed in the network. In all cases, these adaptation protocols do not advance the window unless the acknowledgment for the first packet is received. TCP is a reliable protocol and

is interested in delivering all the data - irrespective of the time taken in doing so. However, for real-time data, we are more interested in meeting the deadlines of the packets and at the same time, providing good quality delivery to the receiver. Hence a rate-based adaptation is followed in our protocols.

I. How "nice" is the protocol?

Any new protocol designed for the Internet should take into account its behavior compared to other protocols. It is important to be a good neighbor and try to divide the available bandwidth among the competing flows, rather than try to consume the entire bandwidth. A badly behaving protocol causes a lot of congestion and denies service to the other behaving protocols.

Thus, it is important to measure how well the protocols estimate and adapt to congestion in the network. The behavior of these protocols in presence of other bandwidth snatching aggressive applications is to be studied as well; aggressive applications should not drive the adaptive algorithms to a very small rate. The impact of these adaptive protocols on other adaptive protocols (real time/non real time) has to be examined. These issues are dealt with in our simulations.

CHAPTER III

LIVE MEDIA AND STORED MEDIA PROTOCOLS

The media data from the multimedia server is characterized into two types - Live Media and stored media. Stored Media is data retrieved from the disk for transmission to the client. In this case, data corresponding to previous frames are available at later points of time during the entire transmission. The client can wait until sufficient data is received and buffered such that the sender can stay ahead of the receiver even in the face of some network congestion. And, if the buffer size on the client side is large, retransmission of lost data from *future* frames (frames which have been received but not yet displayed) can be requested while displaying the current frame. When there are bursts of losses, the client can opt to idle for some time when the data is being retransmitted to ensure good quality picture. Client buffering enables the sender to *work ahead* of the receiver. This allows a larger time window for transmitting/retransmitting any given data packet.

In the case of Live Media, data is not stored on a disk and the source of data is from a camera or some such device. Here, data corresponding to previous frames are not available at later points of time during the transmission. The client cannot wait for frames to be retransmitted - because new data is being generated during that time. Live Media protocols can also be used in applications such as teleconferencing where the transmission delay is an important factor of quality.

A. Live media

In this case, the receiver follows a simple protocol (details explained in Section 2). The receiver is not too rigid regarding the quality of reception in the current display interval to avoid the risk of losing new data in the next interval. The receiver cannot

buffer data for too long since there is more data being generated in that time, and hence the receiver cannot wait for frames to be retransmitted beyond the current interval. Live Media is expected to be used in teleconferencing type of applications.

1. Live media sender

This protocol assumes that the data given to it for transmission to the receiver is packetized and the packets are ordered in terms of their priority. The packets are in a queue waiting to be transmitted.

Each cycle (interval) is divided into many sub-intervals. At the start of the each interval, the desired_rate is calculated by the formula

$$desired\_rate = \frac{\theta * (amount\ of\ data\ to\ be\ transmitted)}{(time\ of\ interval)} \tag{3.1}$$

The factor of $\theta > 1$, (in our experiments, $\theta = 1.25$) is used to allow time for retransmission and for scheduling delays at the receiver. The measurement of actual_rate made in previous interval is used at the beginning of the current interval. The target_rate is calculated as the minimum of the actual_rate and the desired_rate.

In each sub-interval, we calculate the number of packets to be transmitted in that sub-interval (*num_pkt_in_subinterval*). This is calculated as :

$$num\_pkt\_in\_subinterval = \frac{target\_rate * sub\_interval}{PacketSize} \tag{3.2}$$

The first two packets in a sub-interval are used for measuring the perceived Bandwidth using the packet pair technique. Actual rate is calculated from this perceived bandwidth as :

$$actual\_rate = \alpha * actual\_rate + (1 - \alpha) * perceived\_BW. \tag{3.3}$$

The value of α determines the weight given to the history of the actual_rate as compared to the currently measured value of the *perceived_BW*. This new value of *actual_rate* will be used in the calculation of *target_rate* in the next sub-interval. If either or both of the marked packets are not received, or if the marked acks do not arrive in order, the current value of *actual_rate* is maintained.

The sender retransmits a packet as soon as it detects that the packet is lost. The transmission of other packets (yet to be transmitted) are delayed and the retransmission is given priority over the other later packets. Since the higher priority packets are transmitted earlier, more priority is given for reception of the higher priority packets than the transmission of lower priority packets. This behavior of the protocol ensures that the higher priority packets have a higher probability of reception than the lower priority packets.

At the end of every sub-interval, *desired_rate* will be updated using

$$desired\_rate = \frac{(num\_pkt\_in\_interval - num\_packets\_sent) * PacketSize}{Time\ remaining\ in\ Interval} \quad (3.4)$$

where *num_pkt_in_interval* is the total number of packets to be transmitted in that interval. This *desired_rate* will be used in the next sub-interval to calculate the new *target_rate*.

2. Live media receiver

The Receiver waits for packets to arrive. Once a packet is received, it stores the packet in the local buffer (according to the sequence number of the packet) and marks that sequence number as received. An ack with the sequence number equal to the smallest sequence number which is not marked received, is sent to the sender.

When a packet from a new interval is received, all packets of the previous time-intervals are marked received. This means that the receiver does not request for

retransmissions for packets of the last interval. Live Media requires that receiver pay attention to receiving packets in the current interval.

B. Stored media

In case of Stored Media, the receiver waits for its buffer to be built up to a sufficient level so that the sender is always working ahead of the receiver. As long as the receiver is able to receive at the rate at which the sender is transmitting, this work-ahead always exists. However, if congestion appears in the network, the receiver starts to fall behind and the work-ahead of the sender decreases and may reduce to zero if congestion persists. In such a case, we have assumed that the receiver will effect a *stall* (the user will be informed about network congestion and be asked to wait), while it waits for more data to be received. This will allow the sender to build the work-ahead back again in some amount of time, which depends on how long the network congestion lasts. We assume that the receiver is interested in a certain minimum quality of the received data, and it can tolerate stalls (the receiver may also wish to limit the number of stalls in a particular stream reception). This kind of approach is also used in commercial streaming video players like Vxtreme[1] and RealVideo[2].

Alternately, the receiver may not wish to stall when the work-ahead gets to zero, but display the frames with lesser quality. In this case, the work-ahead has to be built up when more network bandwidth is available at a later point of time. Till such time, the receiver should be content with low quality pictures. In our protocols, we have used the former approach where the receiver will stall during network congestion and will wait for the work-ahead to be established again before proceeding.

---

[1]http://www.vxtreme.com

[2]http://www.real.com

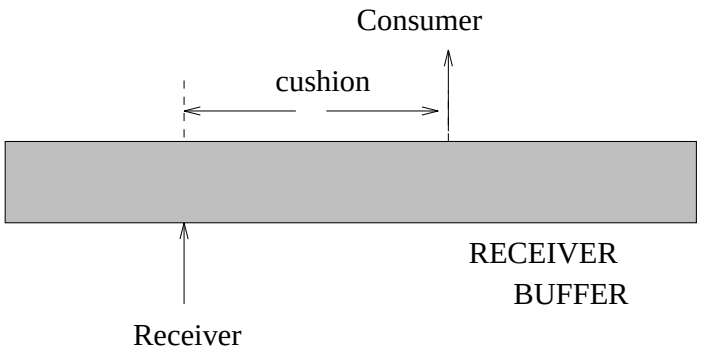

Fig. 6. Cushion in the stored media protocol

The receiver, in this Stored Media protocol, is seen as logically comprising of two parts - a *consumer* and a *packet receiver*. The packet receiver receives the packets from the network and stores them in a buffer. The consumer reads the data from this buffer, processes the compressed data and plays it as video. The requirement is that the consumer should be supplied with the required amount of data every time interval. As long as this is ensured, a smooth playback is achievable. The finite buffer at the receiver can be seen as a circular buffer with two pointers - one maintained by the packet receiver (*receiver_pointer*) and the other maintained by the consumer (*consumer_pointer*) as shown in Fig. 6. The goal of this protocol is to ensure that the *receiver_pointer* always remains ahead of the *consumer_pointer* by a certain threshold. If the difference (called the *cushion*) is larger than the threshold, and if packets have been lost in earlier time-intervals, then some time may be devoted for recovery of the lost packets through retransmission. The value of *cushion* is sent as a feedback packet to the sender, which modifies the *desired_rate* value. The details are explained below.

Two thresholds are defined for the cushion value - MIN_THR and MAX_THR. Ideally, the cushion value should remain between these two values and should be as close to the MAX_THR as possible so more lost data has a chance of getting

recovered through retransmission. The MIN_THR is a safe distance away from the consumer_pointer, while the MAX_THR depends on the buffer availability of the receiver.

Similar technique of using buffer cushion is proposed by Kanakia, Mishra and Reibman [6] to be used at every hop in the network. Our protocol is based on providing the feedback only from the receiver and not from every hop in the network. In our protocol, the sender adapts delivery by utilizing both the measured actual rate and the cushion at the receiver.

1. Stored media sender

The Sender Protocol remains the same as the sender protocol described in Section 1 except processing the cushion feedback from the receiver. The sender modifies desired rate based on the cushion feedback.

If cushion is less than MAX_THR,

$$desired\_rate = desired\_rate * (1 + \frac{\beta * (MAX\_THR - cushion)}{MAX\_THR}) \qquad (3.5)$$

This makes an attempt to increase the cushion back upto MAX_THR level. The constant β, (≤ 1), is used to effect a gradual modification to desired rate instead of a rapid increase.

If cushion is greater than MAX_THR, the desired_rate is not increased so that the cushion can fall back to the stable region. Retransmits are allowed for previous intervals when cushion is above MIN_THR. When cushion is below this level, retransmits are not honored, i.e., the protocol trades off quality for smooth playback.

2. Stored media receiver

The stored-media receiver protocol has many differences with its live-media counterpart. There are two parts to the receiver as explained above - the packet receiver and the consumer. The consumer starts consuming data after sufficient amount of data has been received to fill the buffer to a considerable extent, i.e., the consumer is always going to be lagging behind the receiver (and hence the sender) by some amount of time. This conforms to the requirement requests of real-time playback.

The consumer models the consumption of data by the user. Every interval, the consumer reads data for one interval from the buffer and makes that part of the buffer write-able again. Once the data is consumed, no retransmit requests are sent for these packets. The value of the cushion is calculated as the difference between the *consumer_pointer* and the *receiver_pointer*. If the cushion value is zero (i.e. the receiver has caught up with the consumer), then a stall occurs at the consumer. This means that the consumer has to wait till a reasonable number of packets have been received before it starts playing the video again. (This could be shown to the user as - *Network congestion : waiting for packets*). The stall occurs till enough data has been received so that the cushion value is greater than MIN_THR. Again, it is emphasized that the protocol can be easily modified to allow no stalls.

The packet receiver does the basic packet reception, processing of the packets, generation of suitable acks and sending the acks to the sender. The *receiver_pointer* maintained by the receiver (to calculate cushion) is advanced only when a certain defined percentage (x%) of packets are received corresponding to the current time-interval. This parameter (x) is to ensure a minimum quality of reception at the receiver. This parameter provides a convenient way of trading quality for stalls at the receiver. When the *receiver_pointer* is currently pointing to i, and x% of packets

corresponding to time-interval $i+1$ have been received, then the *receiver_pointer* is incremented to $i+1$. In the meanwhile, if packets corresponding to time-intervals $i+2$, $i+3$, etc. are received, they are just copied into the receiver buffer. When x% of packets in time-interval $i+2$ are received, the *receiver_pointer* is advanced to $i+2$, and so on.

CHAPTER IV

SIMULATIONS AND RESULTS

This chapter describes a set of experiments performed to evaluate the performance of the above mentioned stored media and live media protocols in various scenarios. The issues dealt in these experiments are :

- How do the protocols estimate and adapt to the available bandwidth of the network?
- How do the protocols behave in the presence of other applications?
- What is the performance of the protocols in the presence of multiple streams running between the same end-to-end machines?

MPEG frame traces for a movie Simpsons was used for all the experiments reported here. Results from other movie streams were similar and not reported here. The video portion of the movie was encoded as a 384 x 288 MPEG bit-stream. The movie lasted for 10 minutes. The traces used were obtained from [2, 18].

Experiments were conducted on a cross-country Internet connection between Texas A&M University and Syracuse University with 15 hops. The experiments consisted of the following :

- Simulated Congestion - To overcome the problem of non-reproducibility, congestion on the Internet was simulated in controlled experiments. A parallel UDP stream which sends data for a random amount of time and sleeps for a random amount of time, was used to simulate the congestion in the network.
- Self Congestion - Multiple parallel streams of the same protocol were run between the same two hosts across the Internet connection.

- Congestion with TCP/UDP application - One or more parallel TCP/UDP applications were run along with a stream of our protocol to study its behavior in such circumstances.

Tests on the actual Internet showed high variability and were not reproducible when run during day time (work hours) due to heavy load on the machines and the network. We found the congestion to be highly variable during day-time, with packets being lost for several seconds at a time. We observed that our protocols adapted fairly well for long periods of bursty losses. However, the results obtained were non-reproducible and not controlled. The congestion experienced during one experiment was very different from another experiment. Hence, results could not be compared fairly. To overcome this problem, we ran the set of experiments at night time when the network usage is not very high. We simulated congestion on the network as explained above.

The experimental results for each experiment for Live Media and Stored Media are presented in this chapter.

Low congestion - The protocols were run with the normal congestion offered by the Internet at night time. The loss rates experienced by the stored media protocol, live media protocol and a UDP stream (without smoothing or retransmissions or rate based control) were measured. The results are given in Table I.

It is to be noted here that these results do not give an accurate picture of the performance of these protocols since the congestion of the Internet is itself highly variable and the results of each of these streams depends on the congestion prevalent at the time of running the experiment.

Simulated congestion - The protocols were run with simulated congestion at night as explained earlier. The simulated congestion consisted of a parallel UDP

Table I. Natural congestion - loss percentages

Experiment	Live Media	Stored Media	Plain UDP
Low congestion	3.82	2.10	3.71

Table II. Simulated congestion - loss percentages

Experiment	Live Media	Stored Media	Plain UDP
with simulated congestion	8.08	2.39	16.0

connection which sends bursts of UDP data for a random amount of time and sleeps for a random amount of time, with this cycle being repeated (The average rate of transmission for simulated congestion was around 45 KBps). The sending rates of the sender (based on the MPEG traces) and the received rates (at the receiver) are shown in Figures 7 and 8. The varying sending rate over time is because of the VBR nature of the data. And, since the receiver does not receive all of the data due to congestion, it is lower than the sending rate at some instants of time.

Table II compares the loss rate of the two protocols with UDP. It is observed that the proposed protocols experience lower loss rate than UDP. The Stored Media protocol performs better than the Live Media protocol due to increased number of retransmissions in the former case.

Self congestion - By running multiple streams of the same protocol simultaneously, we wished to determine the performance of the protocols in presence of other streams of the same kind. The Live Media protocol does not involve many retransmissions (no retransmissions of lost packets of previous intervals), and will transmit

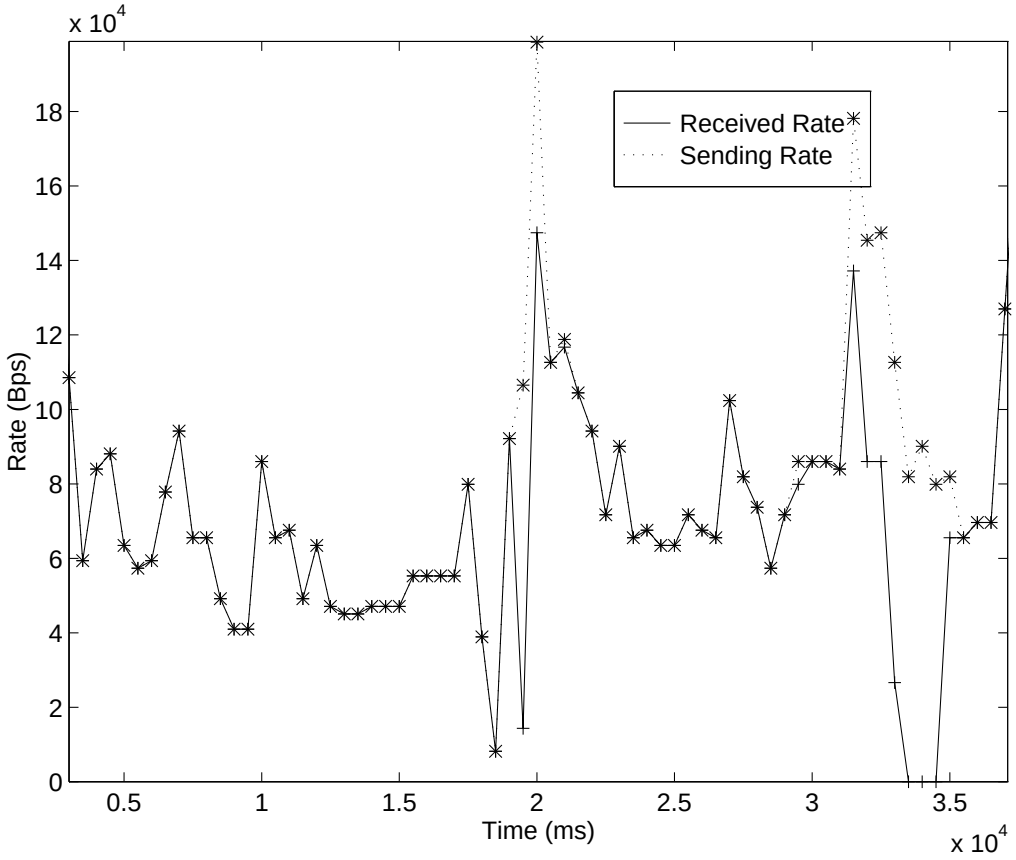

Fig. 7. Sending/received rates for live media protocol

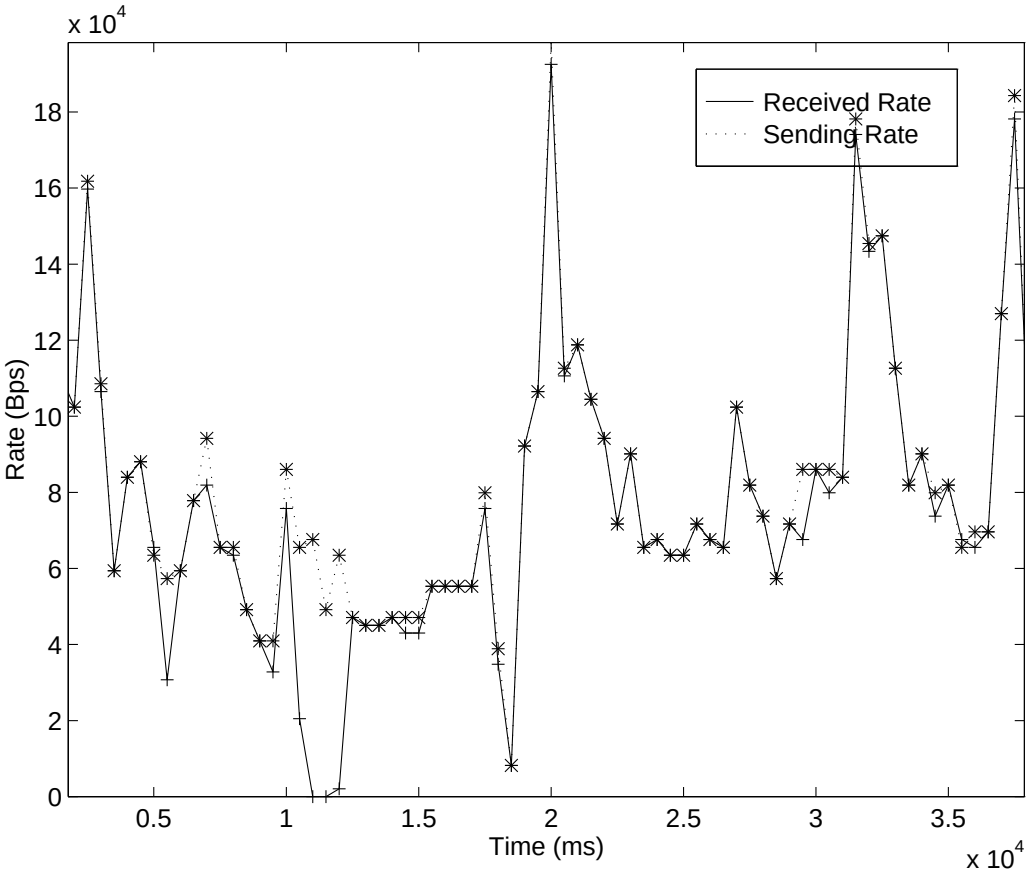

Fig. 8. Sending/received rates for stored media protocol

Table III. Loss percentages with multiple parallel sessions of UDP and FTP

Experiment	Live Media		Stored Media	
	UDP	FTP	UDP	FTP
1 parallel session	24.35	2.07	12.36	0.92
2 parallel sessions	42.11	2.67	29.09	4.33
3 parallel sessions	53.47	2.53	41.37	4.83

only the packets of the current interval. This does not load the network as much as the Stored Media protocol where retransmissions are more common. Figures 9 and 10 show the percentage of lost packets when 2, 3 and 4 streams of parallel Live Media and Stored Media protocols respectively. The plain bars show the loss rates for each of the streams while the shaded bar show the average loss rate in each case. It can be seen from the figures that larger number of parallel streams introduce more amount of congestion, and hence, there will be more losses in each of the protocol streams. However, Stored Media protocols perform worse than Live Media protocols when there are 4 parallel streams, because of the additional overhead introduced by the Stored Media protocols due to the large number of retransmits. Stored Media protocol achieves lower loss rates but is not as scalable as the Live Media protocol.

Parallel TCP/UDP sessions - We ran our Live Media and Stored Media protocol streams with multiple TCP streams like FTP to observe if the increased bandwidth required by the multiple FTPs will affect the performance of our protocols. When multiple streams of FTP are run, they consume large amount of bandwidth and an adaptive application running in parallel will tend to back off in terms of its

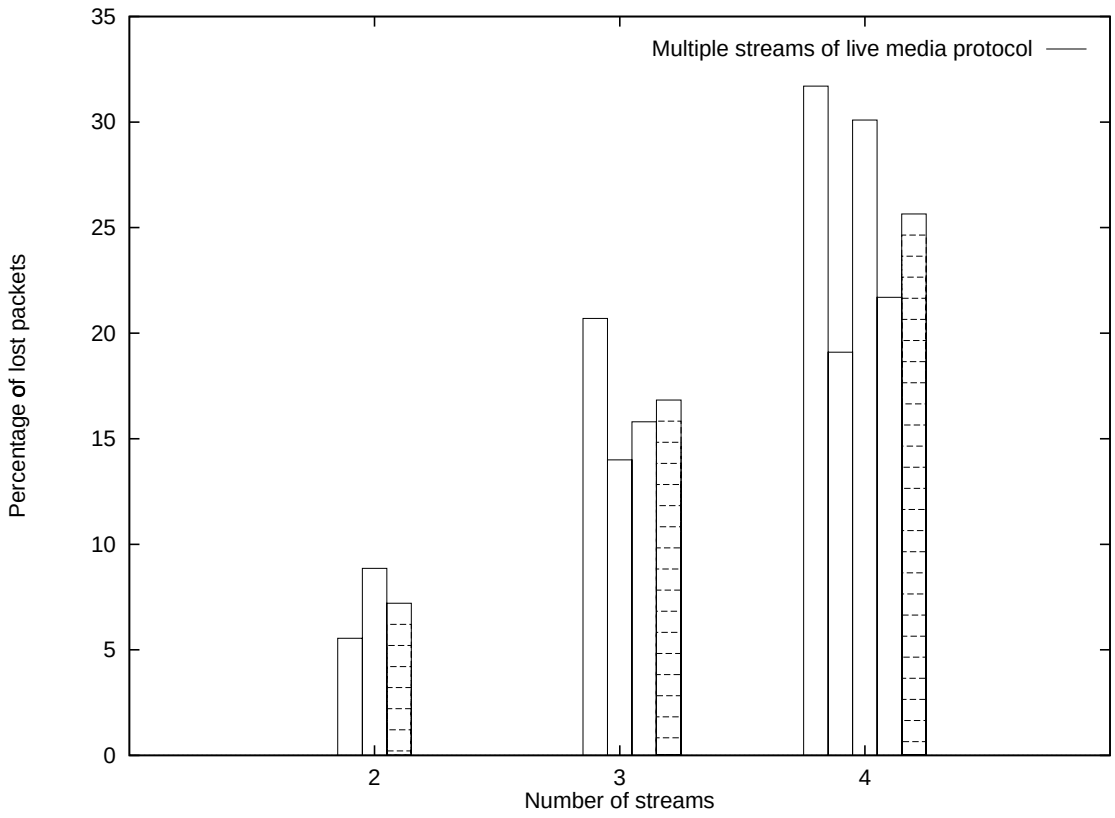

Fig. 9. Multiple live media protocol streams

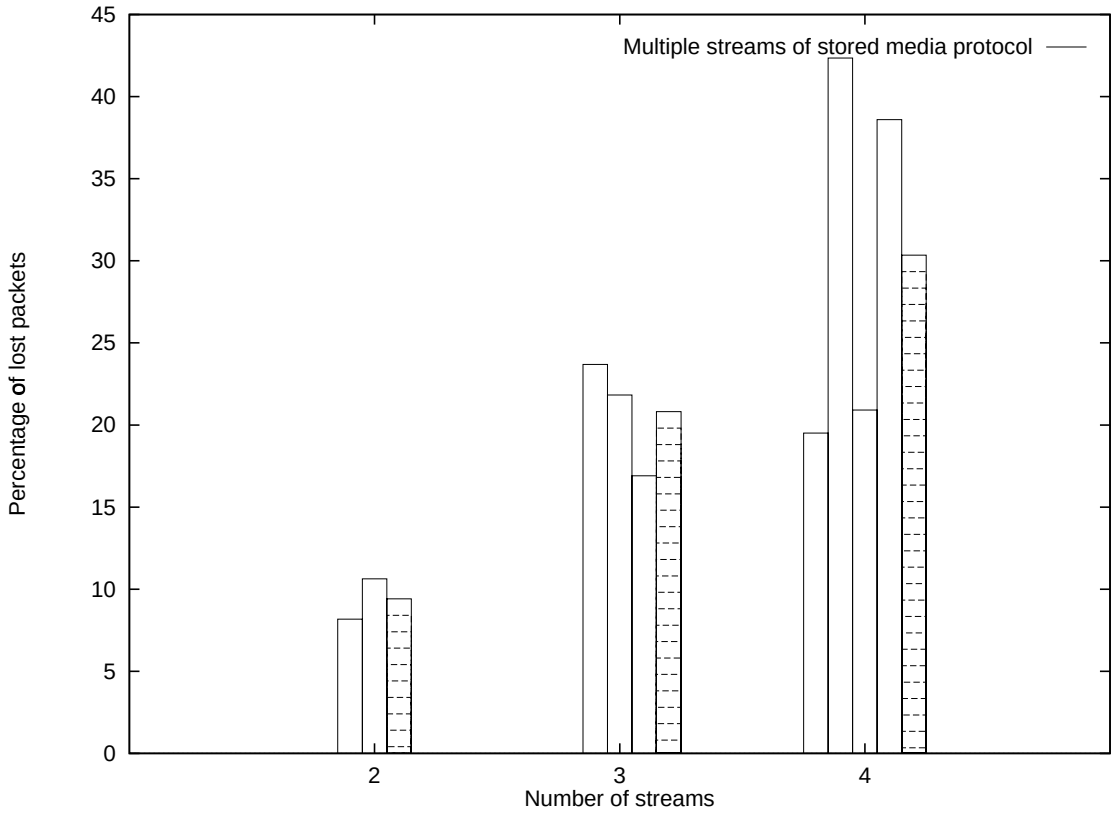

Fig. 10. Multiple stored media protocol streams

usage of bandwidth. TCP also adapts to congestion and hence, it will be interesting to see how these different applications affect each other.

We also ran the Live Media and Stored Media protocols with parallel UDP connections which transmitted the same amount of data as the FTP. We used the same amount of data so that the performance of our protocols in the presence of parallel TCP traffic and parallel UDP traffic can be compared.

Combined results for the above 2 experiments are shown in Table III. It can be observed that our protocols do not experience much losses with many FTP streams in parallel. In fact, as the number of parallel FTP streams are increased, the rate of each of the FTP stream falls - for 1 FTP, the observed rate of transmission was 120KBps, while for 2 and 3 parallel FTP streams, the observed rates were 110KBps and 100KBps respectively. File Transfer Protocol (FTP), running on top of TCP, will back off its sending rate in the presence of congestion in the network. When our protocol is run in parallel with FTP, our protocol tries to utilize a large amount of the available bandwidth, forcing FTP to back off. Our protocols maintain the calculated sending rate for a certain time (sub-interval) before modifying it, whereas FTP backs off as soon as it sees more congestion to accommodate the multiple streams. Because of this, our protocols do not experience much losses even in the presence of multiple parallel streams of a TCP application like FTP.

From Table III, we can observe that parallel UDP connections cause more losses than parallel TCP sessions. Since UDP is not adaptive to congestion, our protocols get impacted more than with parallel FTP sessions which adapt to congestion by reducing their sending rates.

We also ran the Live Media and Stored Media protocol streams with multiple parallel UDP streams, which were supplied with the same MPEG trace data as supplied to the Live Media and Stored Media protocols. The UDP streams send the given

Table IV. Loss percentages with parallel UDP streams (with intervals)

Experiment	Live Media	Stored Media
1 parallel UDP	6.26	7.71
2 parallel UDPs	18.82	15.8
3 parallel UDPs	27.78	43.92

amount of data in a single burst in the beginning of each interval and do not perform any further retransmissions and error corrections for the lost data. The impact on the performance of our protocols is shown in Table IV.

It can be seen from Table IV that the Live Media protocol and the Stored Media protocol suffer comparable losses with one or two UDP streams in parallel. However, Live Media protocol achieved a lower loss rate than Stored Media protocol in the presence of three parallel UDP sessions. This is again because of the better scalability of the Live Media protocol.

The **effect of smoothing** the transmission over sub-intervals is shown in Table V. The duration of the sub-interval is varied from 25ms to 250ms and the loss rates for the Stored Media and Live Media are shown. It is observed that the loss rate decreases as the size of the sub-interval is increased from 25ms to 50ms, but increases beyond that. As we increase the sub-interval, protocols do not adapt to congestion as quickly and hence, results in more losses. Also, at larger sub-intervals, the current state of the network (according to feedback) is not used to calculate the new sending rates for the next sub-interval. This is because, the state of the network is probed in the beginning of each sub-interval and the information gained from this is used only

Table V. Effect of smoothing - loss percentages

Sub-interval	Live Media	Stored Media
25 ms	1.74	2.86
50 ms	1.72	1.24
100 ms	7.23	3.86
250 ms	6.93	5.44

at the beginning of the next sub-interval.

Different packet sizes - We tested our protocols with varying packet sizes. The performance is shown in Fig. 11. This shows that a bigger packet size will work better than a very small packet size - this is due to reduced per-packet overhead with bigger packets. Desirable packet size is atleast 1k bytes.

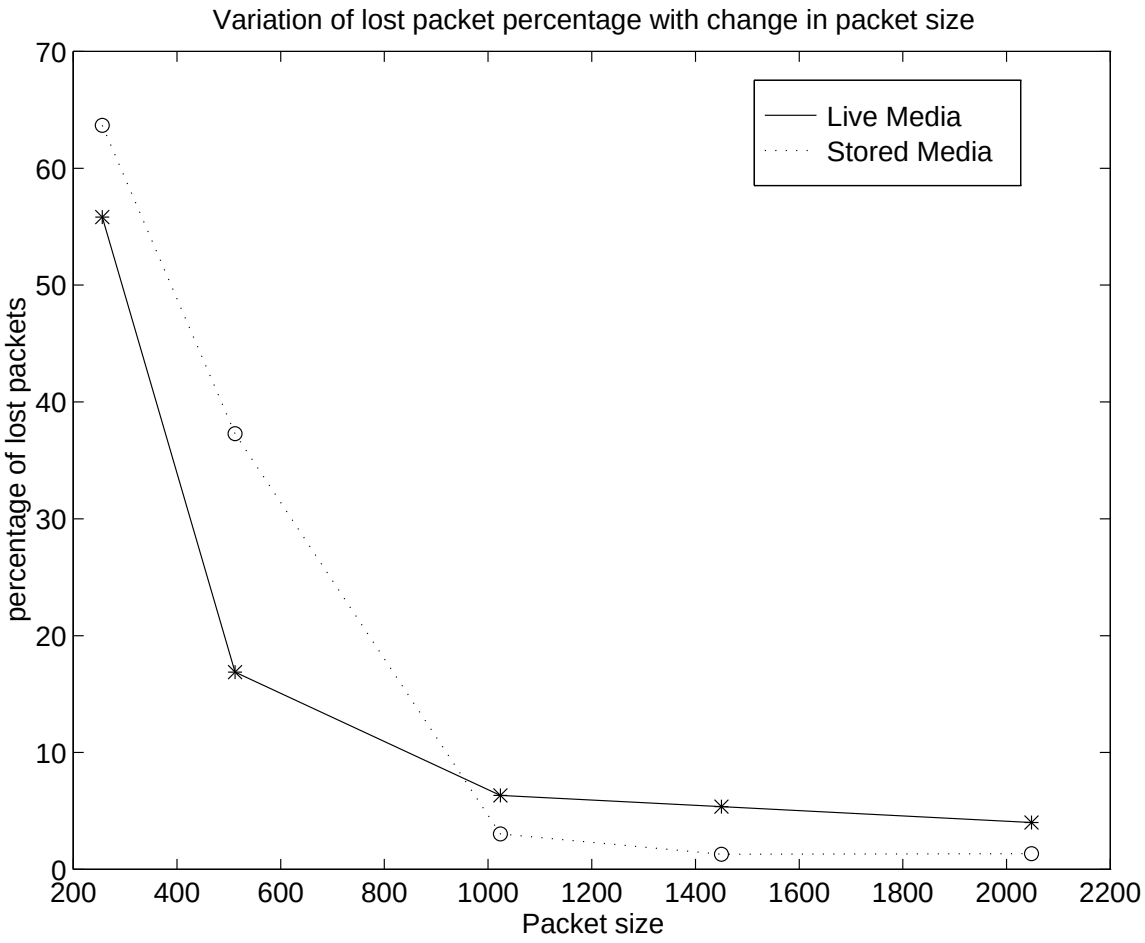

Fig. 11. Experiment with different packet sizes

CHAPTER V

CONCLUSIONS

We presented two protocols for best effort delivery of video data. The protocols are based on the characteristics of the data, Live Media or Stored Media. The Live Media protocol is designed to run smoothly without any stalls, but at lower quality during congestion. The Stored Media protocol exploits *work-ahead* to improve quality. Both the protocols use rate-based adaptation to adapt to network congestion. Both the protocols are designed to work with prioritized data.

To evaluate the performance of the protocols, we conducted several controlled experiments so that reproducible results could be obtained. The following observations were made :

- Both the protocols adapted to moderate congestion well.
- Live Media protocol puts less demand on network resources and performs better than Stored Media protocol in the presence of similar streams and hence more scalable.
- Both the protocols maintained their data rates in the presence of other adaptive applications such as FTP (based on TCP).
- Both protocols suffered more losses in the presence of non-adaptive UDP based applications.

These protocols have inbuilt congestion control and flow control features, avoiding the overhead of TCP like congestion and flow control. While TCP is a very nice protocol in the sense that it reduces its sending rate drastically when a packet is lost, these protocols are more aggressive than TCP. They are not as 'nice' as TCP is to

its neighbors. However, they are not totally non-adaptive like UDP. They have rate based control of sending rate.

Future work may include adding these protocols into the socket library so that applications can use them. Work also needs to be done on how to make these protocols work in a multicast environment. The coexistence of these protocols with the other protocols in the Internet is to be studied before they can be deployed.

Overall, these protocols have shown that streaming media over the existing Internet is not impractical, and it is possible to extract the best quality achievable out of the existing Internet infrastructure.

REFERENCES

[1] L. Zhang, S. Deering, D. Estrin, S. Shenker, and D. Zappala, "RSVP: A New Resource ReSerVation Protocol," *IEEE Network Magazine*, pp. 8–18, September, 1993.

[2] MPEG-1 Frame Size Traces, ftp://ftp-info3.informatik.uni-wuerzburg.de/pub/MPEG.

[3] B. C. Smith, "Cyclic-UDP: A priority driven best-effort protocol," tech. rep., Cornell University, 1994. http://www.cs.cornell.edu/Info/Faculty/bsmith/nossdav.ps.gz.

[4] D. Kozen, Y. Minsky, and B. Smith, "Efficient algorithms for optimal video transmission," Tech. Rep. TR95-1517, Cornell University, Computer Science Department, May 16, 1995.

[5] K. Jeffay, D. Stone, T. Talley, and F. Smith, "Adaptive, best-effort delivery of digital audio and video across packet-switched networks," *In Proc. Network and Operating System Support for Digital Audio and Video, Lecture Notes in Computer Science*, vol. 712, pp. 3–14, 1993.

[6] H. Kanakia, P. Mishra, and A. Reibman, "An adaptive congestion control scheme for real-time packet video transport," in *IEEE/ACM Transactions on Networking*, vol. 3, pp. 671–682, Dec. 1995.

[7] Audio-Video Transport Working Group, H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson, "RFC 1889: RTP: A transport protocol for real-time applications," Jan. 1996. Status: Proposed Standard. ftp://ftp.internic.net/rfc/rfc1889.txt.

[8] S. McCanne and V. Jacobson, "vic: A flexible framework for packet video," in *The Third ACM International Multimedia Conference and Exhibition (MULTIMEDIA '95)*, (New York), pp. 511–522, ACM Press, Nov. 1996.

[9] J. Postel, "RFC 768: User datagram protocol," Aug. 1980. Status: Standard. ftp://ftp.internic.net/rfc/rfc768.txt.

[10] J. Postel, "RFC 793: Transmission control protocol," Sept. 1981. Status: Standard. ftp://ftp.internic.net/rfc/rfc793.txt.

[11] Z.-L. Zhang, J. Kurose, J. D. Salehi, and D. Towsley, "Smoothing, statistical multiplexing and call admission control for stored video," *Accepted for publication in IEEE Journal of Selected Areas in Communication*, 1996. ftp://gaia.cs.umass.edu/pub/Zhang96:JSAC-Smoothing.ps.gz.

[12] S. Keshav, "Packet-pair flow control," tech. rep., AT&T Bell Laboratories, Murray Hill, New Jersey, 1994. ftp://ftp.research.att.com/dist/qos/pp.ps.Z.

[13] L. Kleinrock, *Queueing Systems. Volume I: Theory.* New York: John Wiley & Sons, 1975.

[14] W. R. Stevens, *TCP/IP Illustrated, Volume 1: The Protocols*, vol. 1. Reading, Massachusetts: Addison-Wesley, 1994.

[15] D. E. Comer, *Internetworking with TCP/IP: Principles, Protocols and Architecture.* Upper Saddle River, New Jersey: Prentice Hall, 1992.

[16] L. L. Peterson and B. S. Davie, *Computer Networks - A Systems Approach.* San Francisco, California: Morgan Kaufmann, 1996.

[17] L. S. Brakmo, S. W. O'Malley, and L. L. Peterson, "TCP vegas: New techniques for congestion detection and avoidance," in *Proceedings, 1994 SIGCOMM Conference*, (London, UK), pp. 24–35, Aug. 31st - Sept. 2nd 1994.

[18] O. Rose, "Statistical properties of mpeg video traffic and their impact on traffic modeling in atm systems," tech. rep., University of Wuerzburg, 1995. Institute of Computer Science Research Report Series. Report No. 101.

VITA

Ashwin Raj Madhwaraj was born on October 30, 1974 in Madras, India. He received his Bachelor of Engineering degree in Electronics and Communications from Regional Engineering College, Trichy in May 1995. After graduation, he worked as a Software Engineer at Siemens Communication Software Ltd., Bangalore until July 1996. In August 1996, he was admitted to the Master's program in Electrical Engineering at Texas A&M University. At Texas A&M University, his research has been in multimedia networks. His address is Department of Electrical Engineering, Texas A&M University, College Station, TX 77843.

The typist for this thesis was Ashwin Raj Madhwaraj.