DISTRIBUTED VIDEO STREAMING OVER INTERNET

Thinh PQ Nguyen and Avideh Zakhor
Berkeley, CA, USA
e-mail: *thinhq@eecs.berkeley.edu, avz@eecs.berkeley.edu*

ABSTRACT

With the explosive growth of video applications over the Internet, many approaches have been proposed to stream video effectively over packet switched, best-effort networks. A number of these use techniques from source and channel coding, or implement transport protocols, or modify system architectures in order to deal with delay, loss, and time-varying nature of the Internet. In this paper, we propose a framework for streaming video from multiple senders simultaneously to a single receiver. The main motivation in doing so is to exploit path diversity in order to achieve higher throughput, and to increase tolerance to packet loss and delay due to network congestion. In this framework, we propose a receiver-driven transport protocol to coordinate simultaneous transmissions of video from multiple senders. Our protocol employs two algorithms: rate allocation and packet partition. The rate allocation algorithm determines sending rate for each sender to minimize the packet loss, while the packet partition algorithm minimizes the probability of packets arriving late. Using NS and actual Internet experiments, we demonstrate the effectiveness of our proposed distributed transport protocol in terms of the overall packet loss rate, and compare its performance against a naïve distributed protocol.

1 Introduction

Video streaming over best-effort, packet-switched networks is challenging due to a number of factors such as high bit rates, delay, and loss sensitivity. As such, transport protocols such as TCP are not suitable for streaming applications. To this end, many solutions have been proposed from different perspectives. From source coding perspective, layered and error-resilient video codecs have been proposed. A layered video codec deals with heterogeneity and time-varying nature of the Internet by adapting its bit rate to the available bandwidth [1]. An error-resilient codec attempts to cope with packet losses using error concealment techniques to improve visual quality at the expense of loss in coding efficiency [1, 2, 3]. From channel coding perspective, Forward Error Correction (FEC) techniques have been proposed to reduce delay due to retransmission at the expense of increased bit rate [1, 4, 5]. From protocol perspective, there are approaches based on multicast [5] and TCP-friendly protocols [1] for streaming multimedia data over the Internet. Multicast reduces the network bandwidth by not sending duplicate packets on the same physical link [13], but it is only appropriate for situations with one sender and many receivers. Meanwhile, TCP-friendly protocols use rate-based control to fairly compete with other TCP traffic for bandwidth, and at the same time, stabilize the throughput, thus reducing the jitter for multimedia streaming [6]. From network perspective, content delivery network (CDN) companies such as Akamai, iBeam, and Digital Island use edge architecture as shown in Figure 1 to achieve better load balancing, lower latency, and higher throughput. Edge architecture reduces latency by moving content to the edge of the network in order to reduce round-trip time and to avoid congestion in the Internet. Companies such as FastForward Networks and Akamai strategically place a large number of the servers around the Internet so that each client can choose the server that results in shortest round-trip time and least amount of congestion.

A number of these schemes assume a single fixed route between the receiver and the sender throughout the session. If the network is congested along that route, video streaming suffers from high loss rate and jitter. Even if there is no congestion, as the round-trip time between the sender and the receiver increases, the TCP throughput may reduce to unacceptably low levels for streaming applications. Furthermore, authors in [11, 12] have revealed the ill-behaved systematic properties in Internet routing, which can lead to sub-optimal routing paths. Based on these, it is conceivable to make content available at multiple sources so that the receiver can choose the "best" sender based on bandwidth, loss, and delay. In a way, this is what Akamai does by moving the server closer to the client. However, if no sender can support the required bit rate needed by the application, it is conceivable to have multiple senders simultaneously stream video to a single receiver to effectively provide the required throughput. Having multiple senders is also a diversification scheme in that it combats unpredictability of congestion in the Internet. If the route between a particular sender and the receiver experiences congestion during streaming, the receiver can redistribute streaming rates among other senders, thus resulting in smooth video delivery.

In this paper, we propose a framework for streaming video from multiple mirror sites simultaneously to a single receiver in order to achieve higher throughput, and to increase tolerance to loss and delay due to network congestion. Our solution combines approaches from different perspectives including system architecture and transport protocols. From systems perspective, we expand the edge architecture to allow simultaneous video streaming from multiple mirror sites as shown in Figure 2. This is in contrast with the edge architecture where only one server is responsible for streaming video to its nearest clients. From protocol perspective, we use a TCP friendly protocol to coordinate simultaneous transmission of video from multiple mirror sites to a single receiver effectively. Although, this work focuses primarily on video streaming, our protocol and architecture can accommodate other multimedia components such as audio as well.

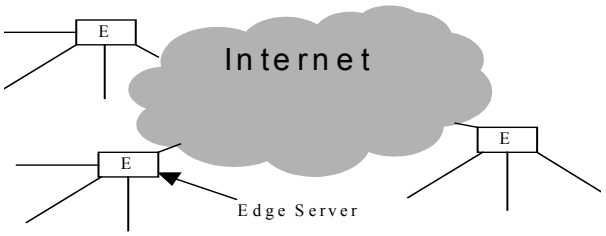

Figure 1: *Edge Architecture*

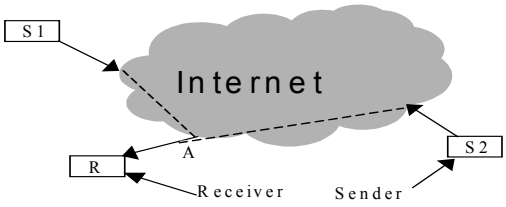

Figure 2: *Distributed Video Streaming*

1.1 Related Work

There have been other works dealing with simultaneous downloading of data from multiple mirror sites. If the data is not delay sensitive, it is possible to use multiple TCP connections to different sites, with each TCP connection downloading a different part of the data. For example, the authors in [7] use Tornado codes to download data simultaneously from multiple mirror sites. More recently, Digital Fountain has used an advanced class of linear-time codes to enable the receivers to receive any N linear-time coded packets from different senders, so as to recover N original data packets. Another example is multiple description coding of video, in which a video source is partitioned into multiple descriptions [8], each one being sent along a different route in the network. This approach assumes that the visual quality of the video degrades gracefully as the number of received descriptions decreases due to network congestion on different routes.

1.2 Assumptions

To successfully stream video from multiple senders, we assume that the available aggregate bandwidth from all the senders to the receiver exceeds the required video bit rate. As an example, consider a network configuration shown in Figure 2. The available bandwidth from router A to receiver R, sender S1 to router A, and sender S2 to router A is assumed to be 2Mbps, 0.8Mbps, and 0.4Mbps, respectively. Router A can be an edge router of a particular network domain of a university or a company, and is assumed to be the only router receiver R is connected to. In this scenario, it is not possible to stream a 1Mbps video from sender S1 to receiver R since the available bandwidth for streaming video from S1 to R is only 0.8Mbps. However, it is possible to stream a 1Mbps video simultaneously from both senders S1 and S2 to receiver R since the aggregate bandwidth from both senders S1 and S2 to receiver R is 1.2 Mbps which exceeds the required bit rate of 1Mbps. On the other hand, if the available bandwidth from router A to receiver R is only 0.9Mbps, then the link between A and R becomes a bottleneck, and video streaming at 1Mbps becomes infeasible both for multiple sender and single sender scenarios.

We also assume that the routes from a client to the senders do not share the same congestion link. If there is congestion on a shared link between two senders, then changing the sending rate of one sender may adversely affect the traffic conditions of the other one. In this work, we assume that changing the sending rate of one sender does not affect the route conditions of others. As will be discussed in Section 2, this assumption prevents potential oscillations and instabilities in the sending rates computed by our rate allocation algorithm.

Based on the above assumptions, in streaming situations when the bottleneck is in the last hop, e.g. due to the physical bandwidth limitation of dial-up modem, our proposed distributed streaming approach is of little use. Indeed if a client is connected to the Internet through a low bandwidth connection, it is preferable to download the entire video in a non-real time fashion before playing it back. Our premise is that, asymptotically, as broadband connection to the Internet such as DSL or cable modem becomes prevalent, the limiting factor in streaming is packet loss and delay due to congestion along the streaming path, rather than the physical bandwidth limitations of the last hop.

Finally, there has been work on detecting the shared congestion points of different routes [9] based on the correlation of packet loss and delay between routes. These correlations can be used ahead of time to improve the performance of our approach.

1.3 Scope

Our contribution can be divided into three parts. First, we propose a protocol that allows simultaneous streaming of video from multiple mirror sites to a single client in a TCP-friendly manner, yet with smooth sending rates so as to reduce jitter. Next, we propose an algorithm that runs on the receiver to specify the sending rate for each sender in order to minimize the total loss rate. Finally, we suggest a distributed algorithm that runs on each sender to partition packets so as to minimize the probability of packets arriving late.

The rest of our paper is organized as follows. In Section 2, we describe our transport protocol, rate allocation, and packet partition algorithms. We show that under certain assumptions, our rate and packet allocation algorithms are optimal in terms of packet loss and delay. Next, we describe the simulation setup and discuss results in Section 3. Finally, we conclude and provide possible extensions in Section 4.

2 Protocol Overview

2.1 Transport Protocol

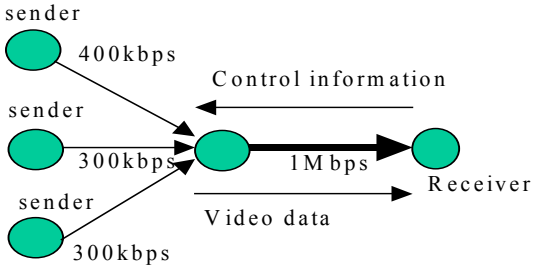

Figure 3: *A scenario of distributed video streaming*

Our transport protocol is a receiver-driven one in which, the receiver coordinates transmissions from multiple senders based on the information received from the senders. Each sender estimates and sends its round trip time to the receiver. The receiver uses the estimated round trip times and its estimates of sender's loss rates to calculate the optimal sending rate for each sender. When the receiver decides to change any of the sender's sending rates, it sends an identical control packet to each sender. The control packet contains the synchronization sequence number and the optimal sending rates as calculated by the receiver for all senders. Using the specified sending rates and synchronization sequence number, each sender runs a distributed packet partition algorithm to determine the next packet to be sent. Figure 3 shows the block diagram of an example of a system deploying our approach.

2.2 Bandwidth Estimation

In our protocol, the receiver estimates available bandwidth for each sender based on the TCP-friendly rate control algorithm (TFRC) proposed in [6]. TFRC protocol is designed to be fair with TCP traffic, and results in less fluctuation in sending rate than TCP does. It calculates the available bandwidth according to the following equation:

$$B = \frac{s}{R\sqrt{\frac{2p}{3}} + T_{rto}(3\sqrt{\frac{3p}{8}})\,p(1+32p^2)} \tag{1}$$

where B denotes the current available TCP-friendly bandwidth between each sender and the receiver, T_{rto} is TCP time out, R is the estimated round-trip time in seconds, p is the estimated loss rate, and s is the TCP segment size in bytes. The estimated round trip time is computed using the moving average of round-trip times over a fixed time interval. Similarly, the estimated loss rate is the ratio of number of lost packets over the total number of packets sent during a fixed time interval. The estimated bandwidth B forms an upper bound for the TCP-friendly sending rate.

2.3 Rate Allocation Algorithm

In our protocol, the receiver computes the optimal sending rate for each sender based on its loss rate and estimated available bandwidth. The problem of allocating optimal sending rate to each sender can be stated as follows. Let N be the total number of senders, and $L(i,t)$ and $S(i,t)$ be the estimated loss and sending rates, respectively for sender i over an interval $(t,t+\Delta)$. Our goal is to find $S(i,t)$, $i=\{1,...N\}$, that minimize the total packets loss during interval $(t,t+\Delta)$ given by

$$F(t) = \sum_{i=1}^{N} L(i,t)S(i,t)$$

$$\text{subject to} \quad \begin{cases} 0 \le S(i,t) \le B(i,t) \\ \sum_{i=1}^{N} S(i,t) = S_{req}(t) \end{cases}$$

where $S_{req}(t)$ is the required bit rate for the encoded video during the interval $(t,t+\Delta)$, and $B(i,t)$ is the TCP-friendly estimated bandwidth for sender i during the interval $(t,t+\Delta)$. This optimization problem can be solved using the following algorithm. At time t, we sort the senders according to their estimated loss rates from lowest to highest. We start with the lowest loss rate sender and assign its sending rate to be its TCP-friendly estimated bandwidth as described in equation (1). We then continue to assign to each sender its available bandwidth, beginning with the ones with lower loss rates and moving to the ones with higher loss rates, until the sum of their available bandwidths exceeds the bit rate of the encoded video. As an example, Figure 4 shows the rate allocation for four senders with the loss rates increasing from left to right. The height of each cylinder shows the available TCP-friendly bandwidth for that sender. As seen, since sender four has the highest loss

rate and since sum of available bandwidth from sender one, two, and three is sufficiently large, sender four can afford to operate at "below capacity."

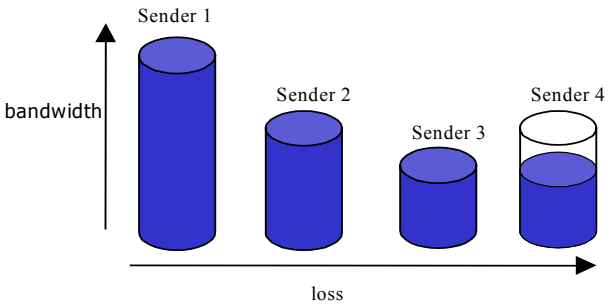

Figure 4: *Sending rate allocation for four senders*

Intuitively, the algorithm assigns more packets to senders with lower loss rates and fewer packets to the senders with higher loss rates. At the same time, each sender *i* also satisfies the constraints $S(i,t) \le B(i,t)$ in order to share bandwidth fairly with other TCP traffic. The algorithm minimizes *F(t)*, the number of lost packets during the interval $(t, t+\Delta)$, given instantaneous feedback, and assuming that the estimated loss rate and TCP-friendly available bandwidth are accurate. The proof for this is included in the Appendix.

In practice, the network delay introduces errors into the estimated loss rate and bandwidth, making the algorithm approximately optimal. This algorithm also assumes that changing the sending rate of one sender does not affect the loss rates of other senders. This is a reasonable assumption if senders are located strategically across the network. On the other hand, if the assumption is not true, then sender's estimated loss rates may fluctuate after each rate assignment, leading to oscillating sending rates. Also, the estimated bandwidth $B(i,t)$ using equation (1) can vary rapidly, even though the average bandwidth of other traffic remains constant. This is due to bandwidth overshooting and undershooting of TCP connections, which result in large fluctuations of the estimated bandwidth of TFRC connections. If $B(i,t)$ changes rapidly, $S(i,t)$ needs to be recomputed more frequently, and more control packets have to be sent from the receiver to all the senders, resulting in inefficiencies. Therefore, we use a systolic window to constrain the minimum interval during which, the sending rates must remain constant. The idea of systolic window is as follows. We periodically sample the estimated bandwidth $B(i,t)$ at a fixed interval ϕ. If $B(i,t)$ is greater than $S(i,t)+w$, where w is a small percentage of $S(i,t)$, then we add one to the variable *count*. At each sampling time, if the estimated bandwidth is smaller than $S(i,t)-w$, then we subtract one from the variable *count*. Otherwise, the variable *count* remains the same. The parameter *count* can be interpreted as the number of times that the sampled bandwidth is outside the window. If at some point the variable *count* is greater than γ, a fixed threshold, or smaller than $-\gamma$, then the rate allocation algorithm is run to update the sending rates. The parameters *w*, ϕ, γ, and *count* determine the minimum interval during which $S(i,t)$ must remain constant, and the responsiveness of $S(i,t)$ to network conditions. If a protocol responds too slowly to decreasing network available bandwidth, it occupies more than its fair share of the bandwidth, and vice versa. Our simulations in Section 3 show that on average, our algorithm based on systolic window shares bandwidth fairly with other TCP traffic.

2.4 Packet Partition Algorithm (PPA)

After receiving the control packet from the receiver, each sender immediately decides the next packet in the video stream to be sent, using the packet partition algorithm. All the senders simultaneously run this algorithm to ensure that no sender sends the same video packet, and also to minimize the probability of packets arriving late at the receiver due to network jitter. The algorithm can be described as follows.

Each sender receives control packet from the receiver through a reliable protocol whenever the receiver determines there should be a change in any of the sending rates. The format of the control packet is shown Figure 5. For simplicity, we do not show the IP header and sequence number.

D1	D2	D3	D4	D5	S1	S2	S3	S4	S5	Sync

Figure 5: *Format of the control packet*

S1-S5 are two-byte fields to specify the sending rate in packets/second for each sender. Packet size is constant for all senders. D1-D5 are one-byte fields to denote the estimated delay from each sender to the receiver. This delay is expressed in multiples of 2 milliseconds interval. The Sync field is the starting sequence number that all senders use in the packet partition algorithm to determine the next packet to send, immediately upon receiving the control packet. The entire copy of the video is assumed to reside at all the senders.

To describe the partition algorithm in detail, we use the notation in Table 1:

k'	Sequence number sync in the control packet which all senders use to initialize the packet partition
$T_{k'}$	Time at which control packet with sequence k' is sent at the receiver
N	Number of senders
P	Packet size
$T_{k'}(k)$	Playback time for k^{th} packet with respect to $T_{k'}$
$\sigma(j)$	Sending interval between packets for sender j
$n_{j,k}$	Number of packets already sent by sender j since packet k', and up to packet k
$D(j)$	Estimated delay from the sender j to the receiver
$S(j)$	Sending rate for sender j

Table 1: *Notations*

If the reference time, $T_{k'}$, is conceptually chosen to be the departure time of the control packet from the receiver, the estimated arrival time of the k^{th} packet sent by sender j is $n_{j,k}\sigma(j) + 2D(j)$. This is because it takes $D(j)$ for the control packet to arrive at the sender j, $n_{j,k}\sigma(j)$ for the k^{th} packet to be sent by sender j, and $D(j)$ for it to arrive at the receiver. Since $T_{k'}(k)$ is also the playback time of the k^{th} packet with respect to $T_{k'}$, the expression $A_{k'}(j,k) = T_{k'}(k) - [n_{j,k}\sigma(j) + 2D(j)]$ can be interpreted as the estimated time difference between arrival and playback time of the k^{th} packet, if sender j is its originator. If $A_{k'}(j,k)$ is positive, k^{th} packet is on time, otherwise, k^{th} packet is late. Hence, maximizing $A_{k'}(j,k)$ is equivalent to minimizing the probability that the k^{th} packet is late.

The basic idea in our packet partition algorithm is that among all senders $j = 1, \ldots N$, the one that maximizes $A_{k'}(j,k)$ is assigned to send k^{th} packet. Specifically, each sender computes $A_{k'}(j,k)$ for each packet k for itself and all other senders, and only sends k^{th} packet if it discovers that $A_{k'}(j,k)$ is at a maximum for itself. If $A_{k'}(j,k)$ is not at a maximum for sender i, it will increase k by one, and repeats the procedure until it finds the packet k^* for which $A_{k'}(j,k)$ is at a maximum among all other senders.

Each sender effectively keeps track of all the values of $A_{k'}(j,k)$ for all N senders and updates $A_{k'}(j,k)$ every time a packet is sent. The values of $A_{k'}(j,k)$ evolve in the same way at all senders even though they are computed at different locations. The reasons for this are that (a) all the senders receive the same control packet from the receiver, (b) only use the information in the control packet to update $A_{k'}(j,k)$ and (c) all use the same equation to do so. Therefore there is no need for information exchange among the senders in order to determine who needs to send the next packet. Nor is there a need for synchronized clock at various senders, as all the needed information is in the control packet.

We now argue that all the needed information to compute $A_{k'}(j,k)$ is in the control packet. To begin with, the estimated delays $D(j)$ are explicitly in the fields D1-D5 of the control packet. Secondly, $\sigma(j) = P/S(j)$ is easily computed from fields S1-S5 in the control packet. As for $T_{k'}(k)$, since it does not in any way affect the value of j that maximizes $A_{k'}(j,k)$, for convenience, it can arbitrarily be set to any value, say zero, by all users for all packets k. As for $n_{j,k}$, we have $n_{j,k} = 0$, for computing $A_{k'}(j,k')$ where k' is the synchronization sequence number as specified in the sync field of the control packet. Therefore for packet k', all senders can determine the sender j^* with the largest $A_{k'}(j,k')$ chosen to send packet k'. Each time sender j is chosen to send a packet, $n_{j,k}$ is incremented by one at all sender locations. Since all senders are initialized to start computing $A_{k'}(j,k)$ for packet k' onwards, and they all receive identical information as supplied in the control packet to update $A_{k'}(j,k)$, the values of $A_{k'}(j,k)$ at all the senders evolve the same way, and as such the decision as to which sender maximizes $A_{k'}(j,k)$ for $k > k'$ is identical at all senders, and hence no duplicate packets are sent.

To illustrate the algorithm, we show a simple example in which, there are two senders, each sending packets at equal rate. As shown in Figure 6, the Sync sequence number is 10. The top line with vertical bars denotes the playback time for the packets. The middle and the bottom lines indicate the time to send packets for senders 1 and 2, respectively. In this scenario, packet 10 will be sent by sender 1 since the estimated difference between playback and its receive time for packet 10 is greater than that of sender 2. Next, packet 11 will be sent by sender 2 for the same reason.

The synchronization among senders would not break down if a control packet is late. In our implementation of a reliable protocol for the control packets, a batch of 5 identical control packets are sent with 5 milliseconds spacing between them whenever the receiver determines there should be a change in sending rates. If none of the control packets is acknowledged within two round-trip time from a particular sender, a new batch of control packets are sent to that sender until at least one of the control packets is acknowledged. Assume that the control packets are lost independently and the loss probability is p, then probability that a particular sender does not receive the control packets after $2n$ round-trip time is p^{5n}. Even if p is as large as 0.1, the probability that a control packet is late more than 2 round-trip time is 10^{-6}. This loss probability is very small, considering that control packets are rarely sent. Even when the control packet is late for one particular sender, that sender simply sends packets behind other senders by 2 round trip time, which can be easily absorbed by using a receiver buffer.

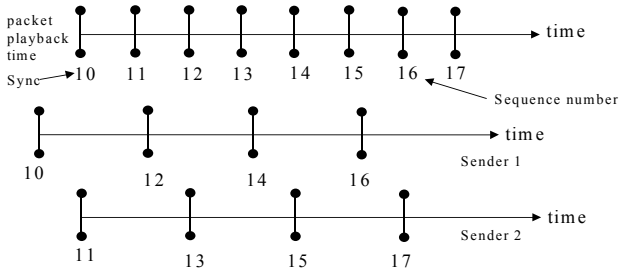

Figure 6: *Illustration of packet partition algorithm*

2.5 Choice of Synchronization Sequence Number

An important consideration in the packet partition algorithm (PPA) is the proper choice of synchronization sequence number, k', at the receiver for each control packet. At any point in time, the receiver knows the sequence number for the last packet sent by each sender. Thus, an obvious strategy is to choose k' more or less to be around the sequence numbers that the senders are actually sending when they receive the control packets. However, it is possible that, due to a variety of practical factors, senders lag each other by significant amount, making the "optimal" choice of k' nontrivial.

As an example, consider Figure 7, where two senders are assumed to send packets at equal rate, with sender one having a shorter round trip time (RTT) to the receiver than that of sender two. The already sent packets by either sender are denoted by the crosses, while the packets to be sent by circles. Suppose the receiver sends out control packets which arrive at sender 1onebefore sender two. Thus, by the time the control packet arrives, sender one's most recently sent packet is 6, and that of sender two is 10. If k' is chosen to be 11, then sender 1 will have to skip packets 8 and 10, resulting in loss. On the other hand, if k' is chosen to be 8, then senders one and two, will run the PPA beginning from packet 8 in order to determine who sends what. In practice, if k' is significantly smaller than the packet sequence number a particular sender is processing at the time it receives the control packet, it is possible that some packets are sent twice, i.e. are duplicated at the receiver.

Based on the above discussion, choosing tool large of a value for k' may result in packet loss, and choosing too small of a value might result in duplication. To avoid excessive packet loss, and hence lower visual quality, our chosen strategy is for the receiver to set $k' = \min_j \{k''(j)\}$, where *k''(j)* is the estimated sequence number for the latest packet sender *j* has just sent, before receiving the control packet. Since the receiver knows about the last packet received from each sender, and the RTT between itself and each sender, it can arrive at a fairly accurate estimate for *k''(j)*. In particular, a reasonable estimate for *k''(j)* is $k^*(j) + 2S * D(j)$, where $k^*(j)$ is the sequence number for the last packet receiver has received from sender *j*, and *S* is the total sending rate in packets per second. This estimate of *k''(j)* is reasonable since $2S * D(j)$ is the approximate number of packets sent by all senders during round trip time of sender *j*, i.e. during $2D(j)$ interval.

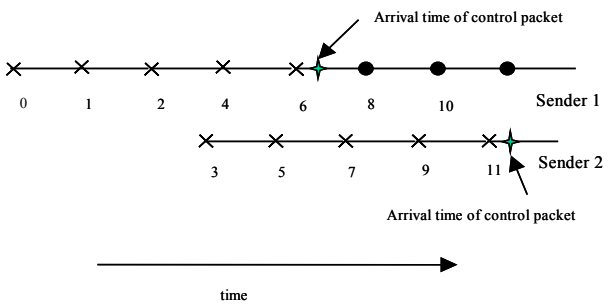

Figure 7: *Illustration of choosing* k'

In addition to the variation in the delays from senders to receivers, other factors can contribute to significant lag amongst the senders. Examples of these include system resource issues at the video servers at the senders, and inaccuracies in estimates.

Clearly, one strategy is to have a large enough receiver buffer to absorb the lag among the senders; however, if the lag is not corrected for long periods of time, say hours, the required buffer size could grow in an unbounded fashion. One way to avoid excessively large required receiver buffer size, is to send control packets with appropriate k' so as to "synchronize" the senders. For example, if $k' = \text{median}_j\{k''(j)\}$ then senders with $k''(j) < k'$ will have to send packets at a faster rate temporarily so as to "catch up" with the other senders, in order to avoid packet loss. Alternatively by choosing $k' = \min_j\{k''(j)\}$, we are effectively slowing down all the senders so as the one, most far behind can catch up.

3 Simulations and Results

We now demonstrate the effectiveness of our approach by showing that simplistic distributed video streaming without using our protocol results in lower video quality than would be achieved otherwise.

3.1 NS Simulations

The first scenario involves a single receiver simultaneously receiving video from two senders using our protocol. Via the same network configuration, in the second scenario, receiver also receives video from two senders but without using our protocol to adapt to the channel conditions. The network configuration is shown in Figure 8. The delays from senders one and two to the receiver are 40 and 50 milliseconds, respectively, the link capacities from senders one and two to the router R are 60Mbps and 30Mbps, respectively, and the local link from router R to the receiver is 240Mbps. Senders one and two simultaneously stream a video to the receiver. The video is a MPEG-1 video sequence taken from MPEG-7 test suite which is then decoded and recoded using H.263 encoder with error-resilient option at 880kbps. At the receiver, we use a simple error-resilient technique to combat packet losses. Basically, the error-resilient technique replaces the lost group of block (GOB) of the current frame with GOB of the previous frame, and copies the motion vectors of the lost GOB from the GOB above it. The parameters for our experiment are shown in Table 2.

To provide a compromise between responsiveness to network conditions and oscillations of sending rates, we find that the parameters shown in Table 1 provide a reasonable trade-off for many experiments using the same network topology but with different link rates and number of TCP sources. Using these parameters, it takes about 20s to 30s for our protocol to respond to network conditions.

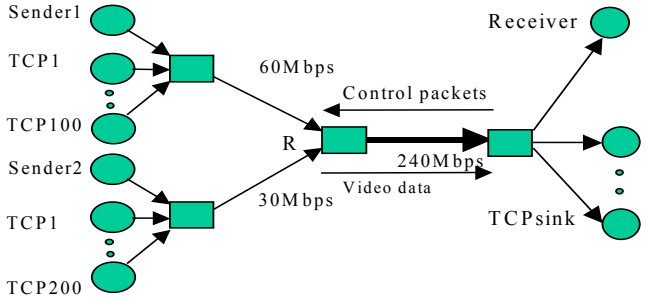

Figure 8: *Simulation configuration*

w (width of the systolic window)	0.1*S(i)
ϕ (sampling interval for the systolic window)	100ms
γ (count threshold)	40
PWIN (time window for estimating loss)	128 RTT
Packet size	500 bytes

Table 2: *Simulation parameters*

At time t = 0s, 100 TCP sources that share the same route with sender one, and 200 TCP sources that share the same route with sender two, start transmitting data. At t = 2s, the receiver starts sending control packets to coordinate the video transmission from senders one and two. Initially, the receiver does not know the fair bandwidth for each sender, so 880kbps is divided equally between the two senders. Figure 9 shows the estimated loss rate for each sender. At t ≈ 25s, both senders begin to send data at the rates according to the equation (1) based on their loss rates and round-trip times as shown in Figure 10. During this time, the average bandwidth of the sender one is 76.7 Kbytes/s, which is close to the fair bandwidth of 77864 bytes/s per connection obtained by dividing 60Mbps among 101 connections. At t = 200s, 25 of 100 TCP sources that share the same route with sender one stop sending data; our protocol responds by increasing sender one's rate by approximately a quarter of its current rate, and reducing the sender two's rate such that the total bandwidth is 880kbps. At t = 400s, 10 new

TCP sources that share the same route with the sender one start and stop at random times, varying the available bandwidth for sender one. As shown in Figure 10, our algorithm adjusts the sending rates for both senders appropriately. Figure 11 shows the total throughput and the throughput of each sender. As seen, the variations in Figure 11 reflect closely those in Figure 10.

In scenario one, both senders use our adaptive protocol to adjust their sending rates accordingly so as to be TCP friendly, and at the same time, to reduce the overall loss rate. In scenario two, the sender two is not TCP-friendly since it sends more than the average bandwidth of other TCP sources while sender one under-utilizes its available bandwidth. Figure 12 shows the throughput for each sender in scenario two, and Figure 13 shows the loss rate ratio of scenario two over that of scenario one. As seen, the loss rate of scenario two can be 2.5 times larger than that of scenario two.

Next, we compare the mean squared error (MSE) of pixel values between the sent frame and the received frame. Since there are quite a large number of frames, we further average the MSE for every video frame over a period of 5s. Higher MSE represents lower fidelity of the video due to lost packets. As shown in Figure 14, most of the time the MSE for scenario one is lower than that of scenario two. The average MSE for the entire simulation for scenarios one and two are 13.70dB and 16.16dB, respectively. The difference in MSE between the two scenarios is most significant from t = 200s to t = 400s. During this period, there are fewer active TCP sources, and therefore, sender one has more available bandwidth, triggering our protocol to allocate more packets to sender one. Since sender one has lower loss rate, the overall loss rate decreases, resulting in smaller MSE. Beyond MSE arguments, it is worthwhile to mention that while our protocol allows the distributed senders to compete for bandwidth fairly with existing TCP connections, in scenario two, TCP connections sharing bandwidth with distributed senders suffer unfairly.

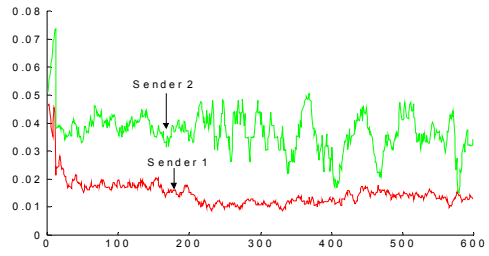

Figure 9: *Loss rates of two senders in adaptive scenario one*

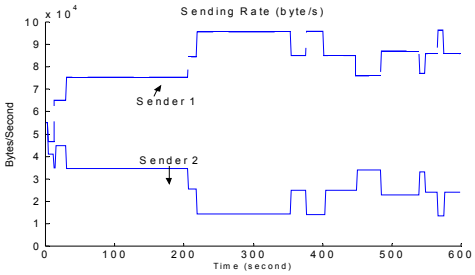

Figure 10: *Sending rates of two senders in adaptive scenario one*

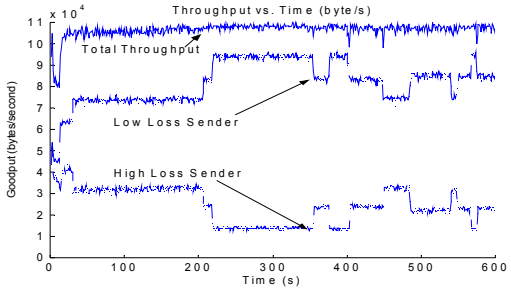

Figure 11: *Throughputs of two senders in adaptive scenario one*

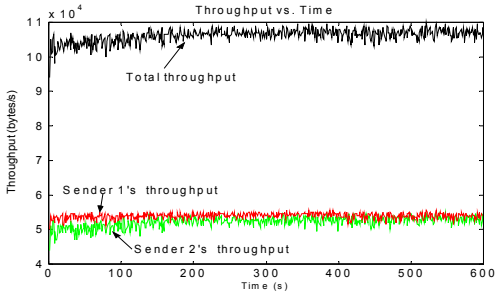

Figure 12: *Throughputs of two senders in non-adaptive scenario two*

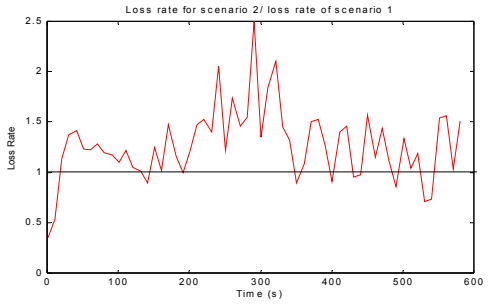

Figure 13: *Loss rate of scenario two over that of scenario one*

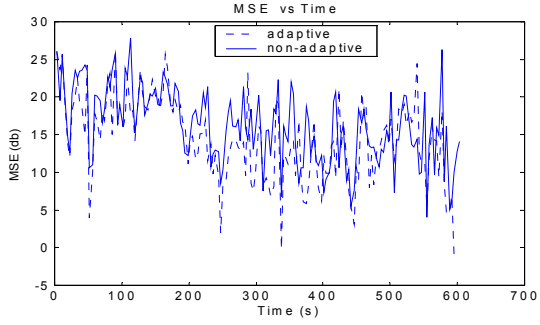

Figure 14: *MSE for adaptive and non-adaptive*

3.2 Internet Experiments

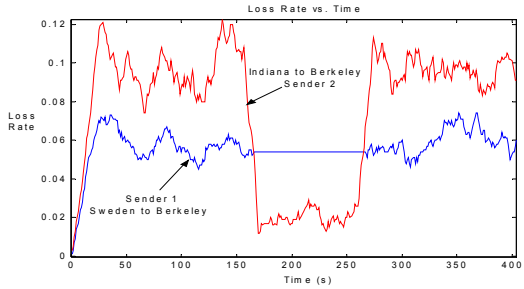

Figure 15: *Loss rate vs. time, streaming from Sweden and Indiana to Berkeley*.

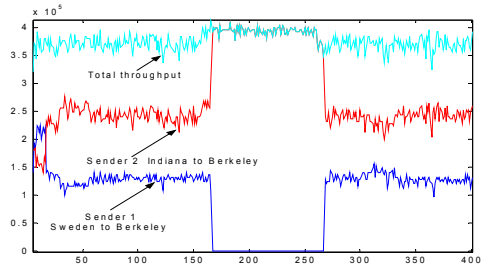

Figure 16: *Throughput vs. time for simultaneous streaming from Sweden and Indiana to Berkeley.*

We have also implemented our protocol for experiments over the Internet. Currently, our protocol supports up to 5 simultaneous connections with a reordered buffer of 100 Kbytes. Reordered buffer is used to reorder the packets coming from different connections. In our Internet experiment, packets of 500 bytes are sent simultaneously at an aggregate rate of 400 kbps from Sweden and Indiana to UC Berkeley. Since there is virtually no congestion from Sweden and Indiana to UC Berkeley, packets are artificially dropped to simulate the congestion effect. Figure 15 shows the initial simulated loss rates of 5% and 10% for senders at Sweden and Indiana, respectively. Initially, the receiver does not know loss rates of two senders, so the total sending rate is divided equally between two senders. After 30s, senders at Sweden and Indiana start sending packets at roughly 260 kbps and 140 kbps, respectively. Figure 16 shows the throughputs for both senders. In this experiment, the sender at Indiana has higher loss rate than that of sender at Sweden, and yet, it sends packets at a higher rate. The reason for this behavior is that the round trip time between Berkeley and Sweden is about 160 milliseconds, while that of Berkeley and Indiana is only 60 milliseconds. Since our rate algorithm assigns rate which is smaller than or equal the sender's fair bandwidth, the assigned sending rate for sender at Sweden is 140 kbps, and the leftover rate of 260kbps is assigned to sender at Indiana.

Between t ≈ 105s and 250s, the loss rate of the sender at Indiana drops from 10% to 2%. Approximately 10s later, our rate allocation algorithm reassigns all 400kbps to Indiana sender since it has lower loss rate, and its fair estimated bandwidth is more than 400kbps. During t ≈ 115 to 260s, the overall loss rate is 2% rather than 2%(260/400) + 5%(140/400) ≈ 3.05% which would have been the loss rate had the sending rates not been adapted to channel conditions properly.

4 Conclusions and Future Work

In this paper, we have proposed a framework for simultaneously streaming video from multiple senders to a single receiver in order to achieve higher throughput, and to increase tolerance to loss and delay due to congestion. We exploit a TCP-friendly protocol and propose rate allocation and packet partition algorithms to minimize the packet loss and jitter. We have shown that distributed video streaming using our protocol results in lower distortion than distributed video streaming in a simplistic, non-adaptive fashion. Our rate allocation and packet partition algorithms have not made use of the relative importance of different bits in the video such as motion vectors and texture bits. We speculate that the MSE will be even smaller if the protocol can be extended to allocate video bits based on their importance. Also, we are incorporating FEC techniques into our future distributed video streaming framework.

5 References

[1] W. Tan and A. Zakhor, "Real-time Internet video using error resilient scalable compression and TCP-friendly transport protocol," *IEEE Transactions on Multimedia*, June 1999, vol. 1, p.172-86.

[2] G. De Los Reyes, A. Reibman, S. Chang, and J. Chuang, "Error-resilient transcoding for video over wireless channels," *IEEE Journal on Selected Areas in Communications*, vol. 18, p.1063-74, June 2000.

[3] J. Robinson and Y. Shu, "Zerotree pattern coding of motion picture residues for error-resilient transmission of video sequences," *IEEE Journal on Selected Areas in Communications*, vol. 18, p.1099-110, June 2000.

[4] H. Ma and M. Zarki, "Broadcast/multicast MPEG-2 video over wireless channels using header redundancy FEC strategies," *Proceedings of The International Society for Optical Engineering*, vol. 3528, p.69-80, Nov. 1998.

[5] W. Tan and A. Zakhor, "Error control for video multicast using hierarchical FEC," *Proceedings of 6th International Conference on Image Processing,* vol. 1, p.401-5, Oct. 1999.

[6] S. Floyd, M. Handley, J. Padhye, and J. Widmer, "Equation-based congestion control for unicast applications," *Applications, Technologies, Architectures and Protocols for Computer Communication*, p.43-56, Oct. 2000.

[7] J. Byers, M. Luby, and M. Mitzenmacher, "Accessing multiple mirror sites in parallel: using Tornado codes to speed up downloads," *Proceedings of Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies*, vol. 1, p.275-83, Mar. 1999.

[8] A. Reibman, H. Jafarkhani, Y. Wang, M. Orchard, and R. Puri, "Multiple description coding for video using motion compensated prediction," *Proceedings of International Conference on Image Processing*, vol. 3, p.837-41, Oct. 1999.

[9] D. Rubenstein, J. Kurose, and D. Towsley, "Detecting shared congestion of flows via end-to-end measurement," *International Conference on Measurement and Modeling of Computer Systems*, p.145-55, June 2000.

[10] S. Seshan and M. Stemm, "SPAND: Shared Passive Network Performance Discovery," *Usenix Symposium on Internet Technologies and Systems*, p.135-46, Dec. 1997.

[11] C. Labovitz, G. Malan, and F. Jahanian, "Internet routing instability," *IEEE/ACM Transactions on Networking*, vol. 6, p.515-28, Oct. 1998.

[12] V. Paxson, "End-to-end routing behavior in the Internet," *IEEE/ACM Transactions on Networking*, vol. 6, p.601-15, Oct. 1997.

[13] S. Deering, D. Estrin, D. Farinacci, V. Jacobson, C. Liu, and L. Wei, "The PIM architecture for wide-area multicast routing," *IEEE/ACM Transactions on Networking*, vol. 4, p.153-62, April 1996.

Appendix

In this appendix, we will show the optimality of the rate allocation algorithm in Section 2.3.

Statement: Consider N senders with the loss rate during the interval $(t, t+\Delta)$ for the i^{th} sender denoted by $L(i,t)$ and its available TCP-friendly bandwidth by $B(i,t)$. Suppose we want to choose a subset of the senders to stream a video with total bit rate of $S_{req}(t)$. Without loss of generality, assume the senders are ordered in such a way that for $i < j$, we have $L(i,t) < L(j,t)$. Then to minimize the total loss rate given by

$$F(t) = \sum_{i=1}^{N} L(i,t)S(i,t)$$

$$\text{subject to} \quad \begin{cases} 0 \le S(i,t) \le B(i,t) & \text{(A.1)} \\ \sum_{i=1}^{N} S(i,t) = S_{req}(t) & \text{(A.2)} \end{cases}$$

We must choose $S(i,t) = B(i,t)$ for $i = 1,...M$ where M is the smallest integer such that $\sum_{i=1}^{M} B(i,t) \ge S_{req}(t)$ and choose $S(i,t) = 0$ for $M < i \le N$.

Proof: We prove the above statement by contradiction. Suppose there exists a rate allocation with M' senders in which $F(t)$ is at minimum, say $F'(t)$, for some $1 \le i' < M'$, but $S(i',t) \ne B(i',t)$. Due to constraint inequality shown in (A.1), this implies that $S(i',t) < B(i',t)$. It is then possible to show that by allocating some of the bit rate from sender M' to i', we can achieve smaller loss rate than $F'(t)$. Specifically, if $\Omega = \min[S(M',t), B(i',t) - S(i',t)]$ is the reallocated bit rate from sender M' to i', then the resulting loss rate will be given by

$$F^*(t) = \sum_{k=1}^{M'-1} L(k,t)S(k,t) + L(i',t)\Omega + L(M',t)[S(M',t) - \Omega])$$

$$= \sum_{k=1}^{M'} L(k,t)S(k,t) - [L(M',t) - L(i',t)]\Omega$$

Since the first term in the above summation is $F'(t)$, and since the second term is always a positive quantity, we have $F^*(t) < F'(t)$. Clearly, this contradicts the fact that $F'(t)$ is the minimum. QED.