A PROTOCOL FOR SIMULTANEOUS REAL TIME PLAYBACK AND FULL QUALITY STORAGE OF STREAMING MEDIA

A Thesis

presented to

the Faculty of the Graduate School

at the University of Missouri-Columbia

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

by

MICHAEL R. SULLIVAN

Dr. Wenjun Zeng, Thesis Supervisor

DECEMBER 2010

The undersigned, appointed by the dean of the Graduate School, have examined the thesis entitled

A PROTOCOL FOR SIMULTANEOUS REAL TIME PLAYBACK AND FULL QUALITY STORAGE OF STREAMING MEDIA

presented by Michael R. Sullivan,

a candidate for the degree of master of science,

and hereby certify that, in their opinion, it is worthy of acceptance.

Professor Wenjun Zeng

Professor Jeffrey Uhlmann

Professor Zhihai He

ACKNOWLEDGEMENTS

I'd like to thank my thesis advisor Dr. Wenjun Zeng and by extension Dr. Heather Yu from Panasonic Research for their discussions that motivated this research. I would also like to thank Greg Heckenberg, Ian Roth, and Ryan Calhoun for their input during this research.

The lengthy span of time between when I finished my class work and when I defended my thesis should provide a cautionary tale for students considering taking a job before finishing their advanced degree. It is far too easy to get fully engrossed in a fun, exciting, and paying job, which makes it difficult to go back and finish your research. For their encouragement to finish this research, I'd like to thank Dr. Zeng, Dr. Uhlmann, and Dr. Pal, as well as my Mom and Dad.

I'd like to thank my sister Maggie, brother Tim, and all of my friends, family & colleagues for making life fun.

Lastly I'd like to thank my wife Susan for always supporting me, making delicious food & art, and putting up with my many nerdy conversations and endeavors.

This research is supported in part by a grant from the University of Missouri System Research Board, and in part, by NSF grant CNS-0423386.

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

A Protocol for Simultaneous Real Time Playback and Full Quality Storage of Streaming Media

Mike Sullivan

Dr. Wenjun Zeng, Thesis Supervisor

ABSTRACT

This research introduces the new problem of simultaneous streaming of a single media bitstream to multiple devices with different Quality of Service (QoS) requirements. In particular, the research addresses simultaneous streaming of a single video stream for both real time playback and full quality storage, where the QoS requirements of the two targets are different. We design a joint streaming protocol to fully exploit the available bandwidth to deliver both real-time and retransmitted packets simultaneously, as bandwidth allows. Preliminary results show that the proposed joint streaming protocol can simultaneously address the requirements of both real-time playback and less time-critical, higher quality storage of streaming media.

We published portions of this research at the 2005 IEEE International Conference on Communications. [9]

Chapter 1

Introduction

It has been envisioned that in the future home, a wired and wireless interoperable network of Personal Computers (PC), Consumer Electronics (CE) and mobile devices enables a seamless environment for sharing and growing new digital media and content services. We envision that a desired and important application scenario in home networking is to be able to stream (potentially live) video from the Internet in real-time to an in-home display device, where some loss of video data is tolerable in exchange for stringent real-time performance, and to, at the same time, be able to store a perfect copy of the video on the in-home hard disk for archival purposes where the delay constraint is less of a concern. It is possible, using existing protocols, to use two separate streams to acquire both quality levels of video. However, streaming the video bitstream for both real-time playback and in-home storage concurrently in two separate processes is not an efficient solution because it does not make use of the high correlation between the two datasets. Using two concurrent bitstreams may not even be viable due to limited available bandwidth, especially for live streaming. Data delivered on time to the home service gateway should instead be shared by both purposes.

This research introduces the new problem of simultaneous streaming of a single video stream to multiple devices with different Quality of Service (QoS) (i.e., delay, loss, quality, resolution, etc.) requirements. In particular, the research addresses simultaneous streaming of a single video bitstream for both real time playback and full quality storage, where the QoS requirements of the two targets are different. This problem requires that the in-home service gateway be designed in such a way that it can communicate with the remote multimedia-streaming server on the Internet to facilitate retransmission of lost packets for in-home storage, even though some of the retransmitted data may not be available in time for real-time display/playback. Deciding how and when to retransmit the lost data is a very challenging issue because that data will consume some bandwidth, and therefore has the potential to impact the real-time streaming performance. We design a joint streaming protocol (JSP) to fully exploit the available bandwidth to deliver both real time and re-transmitted packets simultaneously, as bandwidth allows. In JSP, mixed unreliable and reliable channels are established and coordinate with each other efficiently and seamlessly. In order to successfully retransmit the lost data without overloading the system, precise bandwidth estimation is necessary. The research implementation uses a packet-pair algorithm [1] that measures the available bandwidth across the delivery path. Using NS-2 network simulator [6], the proposed joint streaming protocol is shown to effectively address the requirements of both real-time playback and less time critical higher quality storage of streaming media.

Chapter 2 provides a high level overview of the problem and describes the elements of the proposed joint streaming protocol. Simulation results and statistical analysis are presented in Chapter 3 to illustrate the effectiveness of the proposed protocol. Chapter 4 draws the conclusion and discusses potential future work.

Chapter 2

The Joint Streaming Protocol

2.1 Problem Overview

Figure 2-1 illustrates the generic problem the JSP is intended to address. The streaming server sends a single multimedia bitstream simultaneously to a set of client devices that share much of the same communication links but that are potentially used for applications with different QoS requirements. Some of the QoS parameters include delay, packet loss tolerance level, media resolution and quality. The objective is to exploit the common set of properties these different client applications share, such as shared links, shared bandwidth, and shared media to efficiently meet the different QoS requirements of each client application. In general, each client device can also contribute resources to assist the QoS performance achievement of other client devices, e.g., by performing media transcoding and adaptive forwarding. Links along the delivery path may have different kinds of characteristics, e.g., wired vs. wireless. In such a scenario, a joint design of the streaming protocol is necessary to achieve efficient concurrent delivery of the multimedia data to multiple client devices.

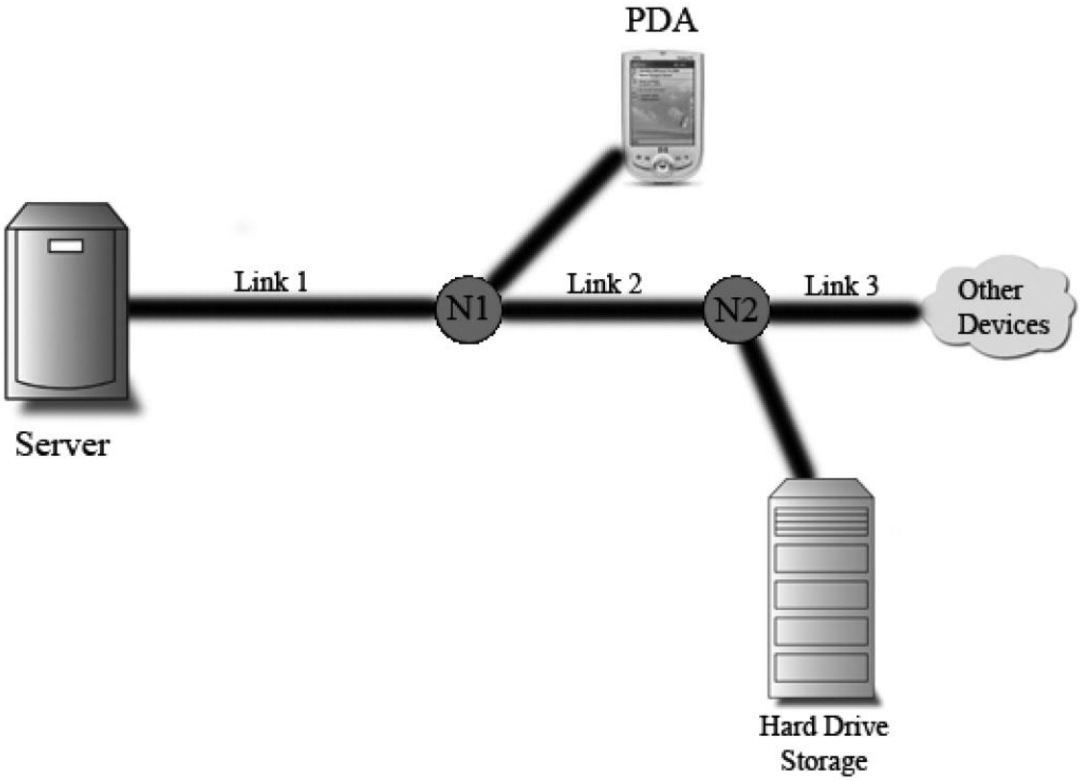


Figure 2-1. A generic scenario for streaming a single bitstream to multiple client devices with different QoS requirements.

In this paper, we focus on a special case of this generic scenario: simultaneously streaming a single video bitstream for both real time playback in a display device and full quality storage in a storage device, where the QoS requirements for the two applications are quite different but they share the same delivery path. We design an efficient joint streaming protocol to address the requirements of both applications.

2.2 Joint Streaming Protocol

Our goal is to design a joint streaming protocol to simultaneously support the QoS requirements of both real time playback and higher quality storage of streaming media.

Real Time Streaming Protocol (RTSP) [4] and Real Time Transport Protocol (RTP) [5] are widely accepted standards for real-time multimedia streaming applications. RTP/RTCP (Real-Time Control Protocol) is designed to work on top of the unreliable UDP protocol to address the real-time requirement of streaming media. This research extends the RTP/RTCP protocol [5] to achieve the goal of addressing two different QoS requirements simultaneously, by allowing retransmission of lost packets for storage when extra bandwidth is available.

RTCP packets currently only send statistical information regarding packet loss. In order to properly inform the server of which packets need to be re-sent, RTCP packets are extended to include more specific information about each lost packet. We call our extended RTCP packets JSCP (joint streaming control protocol) packets.

The server uses a priority queue to manage the re-send packets, given a constraint on the server buffer size. The algorithm used to determine the priority of the packets to be re-sent could be tailored for specific purposes. In the case of a video stream, Intra-frames are assigned a higher priority than Inter-frames. The server re-sends these older packets along with the real-time packets as bandwidth and server buffer size allow.

The three major modules of the proposed JSP are bandwidth estimation, server functionalities, and joint streaming control protocol.

2.3 The Available Bandwidth Estimation

The two metrics most commonly used to determine a path’s bandwidth are the capacity and the bandwidth available along the path [3]. The capacity is also commonly referred to as the bottleneck bandwidth. A path’s bottleneck bandwidth is the maximum amount of data that can fit across the link in the path with the smallest capacity. In Figure 2-2, the bottleneck bandwidth of the path from the server node to the receiver node is equal to the bottleneck bandwidth of Link 1. A path's available bandwidth is defined by the amount of bandwidth available across the link with the least available bandwidth. In Figure 2-2, the available bandwidth of the path from the server node to the receiver node is equal to the available bandwidth of Link 3. As illustrated, the link with the least available bandwidth (Link 3) is not necessarily the same link as the one with the smallest capacity (Link 1).

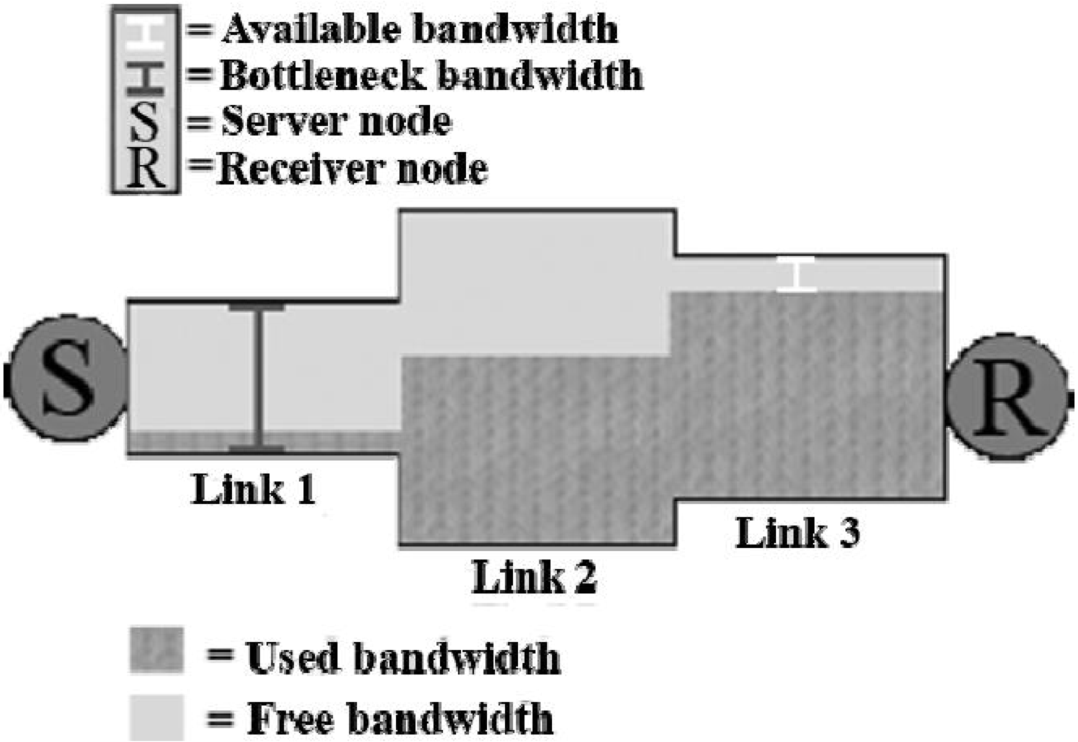


Figure 2-2. An illustration of the difference between available and bottleneck bandwidth values. Link 1's bottleneck bandwidth is the bottleneck bandwidth of the path. Link 3's available bandwidth is the available bandwidth of the path.

The available bandwidth estimation is critical to the performance of the proposed JSP protocol. Poor bandwidth estimation can lead to one of two problems. An underestimate of available bandwidth would lead to poor utilization of the network, which would result in the server attempting to re-send too few packets, thus the archival quality may suffer. An overestimate of available bandwidth leads to overloading of the network. This is very undesirable because the re-send packet stream should never interfere with the real-time stream. If the network is overloaded, more of both real-time packets and re-send packets will be dropped across the path. That scenario violates the goals of the JSP because it is both detrimental to the quality of the real-time playback, and it starts a vicious cycle that increases the number of re-send packets necessary to achieve full archival quality. Therefore, in general, conservative bandwidth estimates are favored over aggressive ones.

Many bandwidth measurement techniques [3] have been developed for estimating path capacity and available bandwidth. Stanford's *nettimer* describes a set of packet-pair estimation algorithms, which measure the available bandwidth of the bottleneck link when fair queuing is used in the routers, as analyzed in [1]. Other techniques such as Pathload [11] and pathChirp [12] use the principle of self induced congestion to determine the available bandwidth. Pathload uses a series of closely spaced packets, sometimes called TOPP (trains of packet pairs) to converge on the available bandwidth more quickly than the packet pair algorithms, but this comes at the expense of heavy bandwidth usage. Rice University's pathChirp uses packet trains with rapidly increasing probing rates, which allows for quick convergence of the available bandwidth without the heavy bandwidth overhead of the TOPP techniques.

The initial implementation of JSP used the ROPP (Receiver Only Packet-Pair) method [1] for estimating the bandwidth. In ROPP, two consecutive packets of the same size are sent to the receiver, which sends back an acknowledgement packet containing the arrival time of the received packets. Upon receipt of the acknowledgement packet, the server calculates the delay and estimates the available bandwidth. Among the packet pair based bandwidth estimation techniques, the ROPP technique is more accurate than the SBPP (Sender Based Packet Pair) technique, and almost as accurate as the most robust RBPP (Receiver Based Packet Pair) technique [1].

Other bandwidth estimation techniques including Rice University's pathChirp were also used during the data collection and testing of this research. More recently, passive techniques to measure available bandwidth have emerged [8]. These techniques could further improve the efficiency of the JSP by providing effective measurement of the available bandwidth without requiring the overhead of the dedicated bandwidth estimation packets used in the current implementation.

2.4 Server Functionalities

The JSP server performs three main functions. It controls the streaming of the real-time packet stream, performs the bandwidth estimation used to determine the appropriate re-send timing, and it performs the buffer management to support the re-sending of lost packets.

JSP calls for two buffers, one for the real time stream, and another to hold lost packets that need to be re-sent when bandwidth allows. Conceptually, the re-send buffer is placed behind the real time stream buffer in Figure 2-3, indicating that the re-send packets have lower priority than the real-time packets.

In Figure 2-3, one can see that the real-time packets, denoted RT-number, leave from the front of the buffer in the order they were received. The packets in the re-send buffer, however, are ordered for re-transmission based on both age and packet importance. For example, in the real-time stream, “OLD I 7” packet was generated after “OLD B 3” (it’s a newer packet), but in the re-send buffer, it will be sent earlier because as an intra-frame, it has higher priority. This insures that the server delivers as much important information as possible to the receiver, given constrained server buffer size.

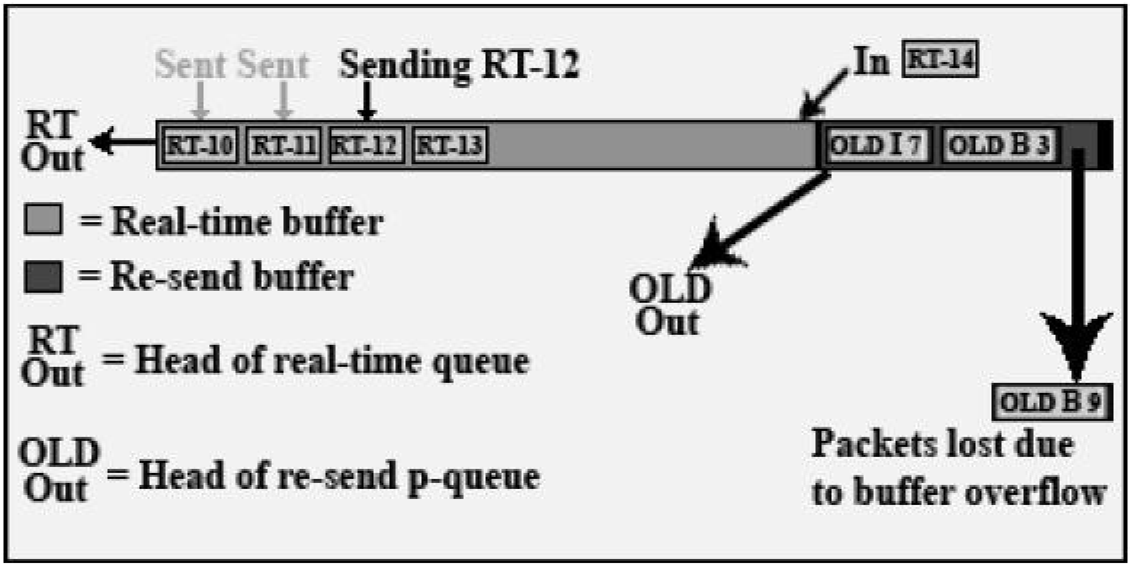


Figure 2-3. A conceptual illustration of the buffer management in a JSP server. P-queue stands for priority queue and OLD packets refer to packets that need to be re-sent. “OLD I 7” is a high priority I-frame packet, and “OLD B 3” is a low priority B-frame packet.

The available bandwidth estimate must be updated frequently enough that it reflects the current status of the network, but in the current implementation, each update involves the overhead of sending dedicated estimation packets. These extra packets, if sent too often, would overwhelm the network and cause unnecessary congestion. In the simulation, we found refreshing the estimation once every thirty real-time stream packets to be sufficient. A more volatile network topology may require more frequent re-estimation.

Unlike many implementations of RTP, a JSP server cannot immediately discard streamed packets. Instead, it must hold onto the packets until it receives a JSCP packet.

The control packets are used to tell the server which packets need to be re-sent. Section 5 of Chapter 2 describes the details about the contents and design of the control packet. Upon receipt of a control packet, the server takes the sent packets (marked with a gray “Sent” in Figure 2-3) and either discards them if successfully received or moves them to the re-send buffer if they were not received.

Because the server must wait for receipt of a JSCP before discarding sent packets, the performance of JSP is sensitive to the server’s real-time buffer size and JSCP packet frequency. If the buffer is too small and the JSCP packets are sent from the client too infrequently, the server will be forced to discard sent packets before identifying whether the client received the data. This could prevent a data stream from being fully reconstructed.

Packet retransmission depends on the result returned from the bandwidth estimation algorithm. The retransmission rate is determined by an increasing staircase function, which takes the estimated available bandwidth as its input. JSP chooses this level of indirection to avoid the potential negative impact of overestimation errors in the bandwidth estimation algorithm. If we had implemented a more robust estimation algorithm then we could base the retransmission rate directly off of the estimation result, with an offset to allow for local variation in the actual bandwidth. In many simulation runs, the remaining bandwidth capacity allows for all lost packets to be re transmitted before termination of the real-time stream.

2.5 Joint Streaming Control Protocol

The joint streaming control protocol is based on the real time control protocol (RCTP). The RTCP includes statistical information about the nature of the packet loss observed on the client side. When the RTP server receives the RTCP packet, it can take some sort of action such as transcoding the bitstream to better match the available bandwidth.

The JSCP also contains information about the packet loss, but as opposed to sending only the statistics about the loss, the JSCP packet stores the identity of the missing packets. This allows the JSP server to purge the packets that have been received by the client from the real-time buffer and move the lost packets into the re-send buffer.

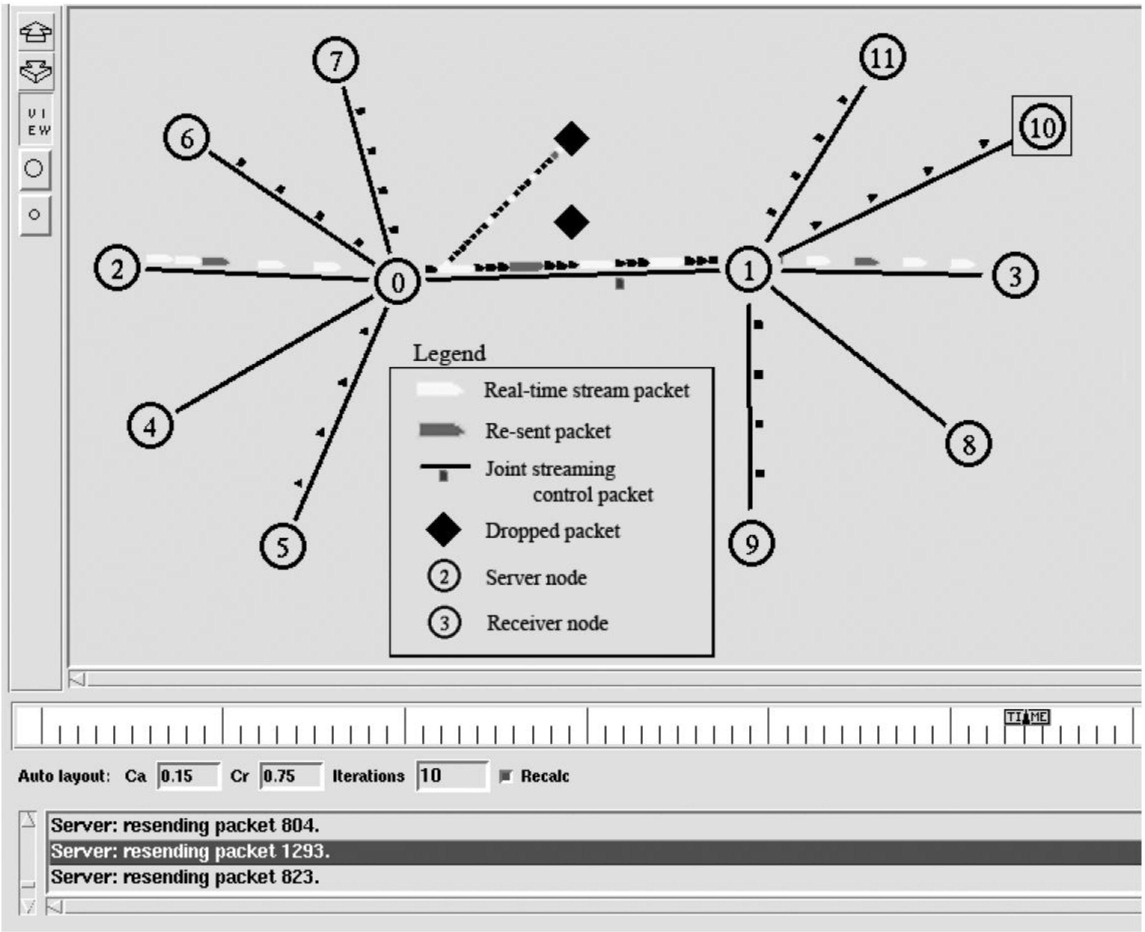
Because we are addressing a unicast problem, the requirements on the frequency and amount of information that can be sent back to the server are not as stringent when compared to a multicast scenario. This will allow us to send JSCP packets back to the server more often if needed to address the server buffer size constraint. In the current implementation, the JSCP packet size grows proportionately to both the number of lost real-time packets and the sequence number size of the lost packets. If overhead became a concern, the JSCP packets could be encoded and compressed. For example, if one were to use the sequence number of the lowest lost packet as a base, one could then define all other lost packets as an offset from that base. The packet size would be reduced considerably.

Chapter 3

Results and Analysis

3.1 Simulation Results

We use ns-2 [6] to simulate realistic network topologies in which JSP might be used. Figure 3-1 is one of the topologies used for testing. We created topologies that would stress the bandwidth estimation algorithm and that reflect realistic traffic scenarios.



Auto layout: Ca 0.15 Cr 0.75 Iterations 10 ☒ Recalc

```
Server: resending packet 804.
Server: resending packet 1293.
Server: resending packet 823.
```

Figure 3-1. Screenshot from an NS-2 simulation. Node 2 is the server sending the video stream using the JSP protocol. Node 3 is the receiver. The link between nodes 0 and 1 is the bottleneck link.

The topology pictured in Figure 3-1 is a wired unicast network with competing traffic flows that start and stop flowing at various points during the simulation. All links are 10 Mbps and use drop-tail fair queuing. In the figure, the nodes are represented by black numbered circles, and the links are represented by solid black lines. The bitstream of interest is flowing from node 2 to node 3. All traffic in the simulation passes across the link that connects nodes 0 and 1. The competing traffic is CBR (constant bit rate) and is represented in the figure by black triangle shaped bitstreams flowing from nodes 4, 5, 6, and 7 to nodes 8, 9, 10, and 11, respectively. They consist of 500 byte packets sent at an interval of one every 2 milliseconds. The queue at bottleneck node 0 is represented by the line of packets extending up and to the right of node 0, and the dropped packets are the large diamond shapes falling off the end of the queue.

The real-time video stream consists of 1500 byte packets, sent at an interval of one every 2.5 milliseconds. The JSCP packet is being sent from the receiver at node 3 to the server at node 2 once every 50 data packets. The server parses the control packets to determine which packets were lost in transmission, then it prioritizes the re-send packets based on that information. The re-sent packets flow from the server to the receiver as bandwidth allows. No bandwidth estimation packets are pictured in this screenshot.

Figure 3-2 shows the total number of lost real-time packets (which does not account for lost re-sent packets) when JSP and RTP are used respectively for the same topology and setting described above. The figure suggests that the JSP re-sends the lost packets at the appropriate time without negatively impacting the real-time playback performance, i.e., without increasing the real-time packet loss rate.

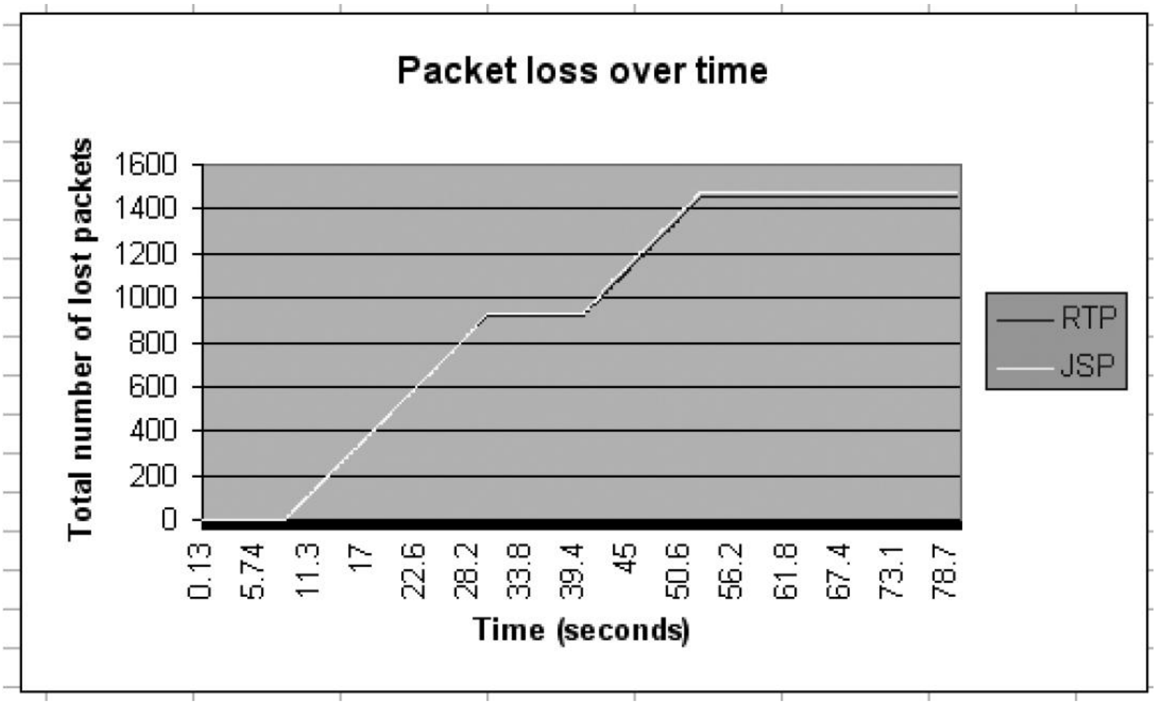


Figure 3-2. Total number of lost real-time packets for JSP and RTP.

Figure 3-3 shows the aggregate bitrate sent by the JSP server's real-time packet stream, it's retransmission process, and by the background traffic. The staircase of the background traffic indicates the number (up to 4) of background servers sending data. At some time intervals, the aggregated rate sent by the combination of background traffic and the real-time JSP stream exceeds the bottleneck bandwidth of 10 Mbps, causing loss of real-time packets. Those lost packets are re-sent only when extra bandwidth is available, to avoid negatively impacting the real-time streaming performance.

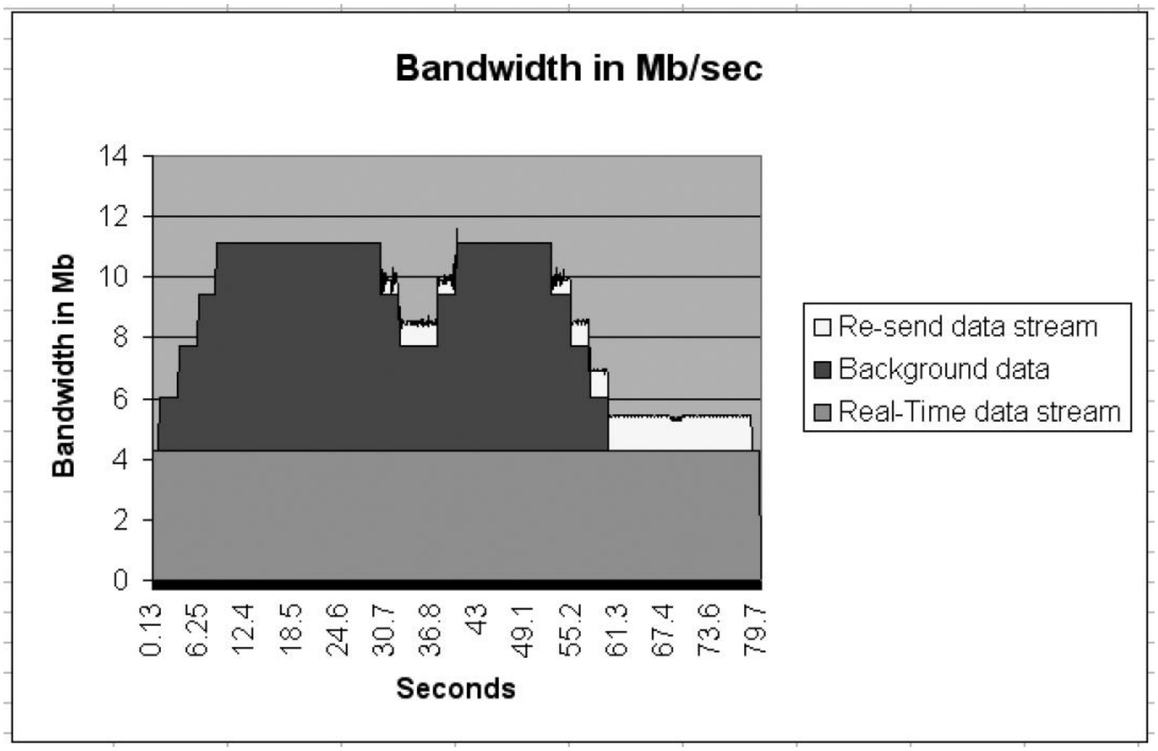


Figure 3-3. Aggregated transmission bitrate.

3.2 Statistical Results

Statistical analysis of a real world video stream confirms the simulation results and shows the potential utility of the JSP. The video analyzed in Figures 3-4 and 3-5 shows marked spikes in data rate depending on the video content. If transmitting over a bandwidth constrained link, a real-time player would have to drop frames to keep up in these periods of high data generation. As we constrain the link capacity, the re-send priority queue grows in size, but even in a very bandwidth constrained example JSP would be able to re-transmit a meaningful amount of data.

To collect these statistical results, an mp4 video stream is generated using *gstreamer*[10] to encode a Linux desktop. The 1600 x 1200 desktop was encoded at 10 frames per second with default options. In this mp4 stream, the data rate is variable because both the intra-frames and the inter-frames grow in size when the video stream content is complex and changing rapidly.

In Figure 3-4, the “Data Generated” line shows the frame data generated by the mp4 stream, while each of the “Data Sent” lines shows the data sent along constrained capacity links in three scenarios where the link capacity differs. In scenario one, the link capacity is 1.0 mebibyte (MiB or 1024*1024 bytes) per second, while in scenarios two and three the link capacity is 0.5 MiB per second and 0.3 MiB per second, respectively. For this analysis, to calculate the data generated per second, the video frame data has been bundled into one second chunks. For each one second chunk, the data value is the sum of the frame-sizes for each frame generated during that one second time-slice. In Figure 3-4, the “Data generated” line shows the variability of the video stream’s data rate over the 633 second capture. At periods of high data generation, the chart also shows how each “Data Sent” line levels off at the link capacity, which shows the maximum data that can be transmitted on that link per time-slice.

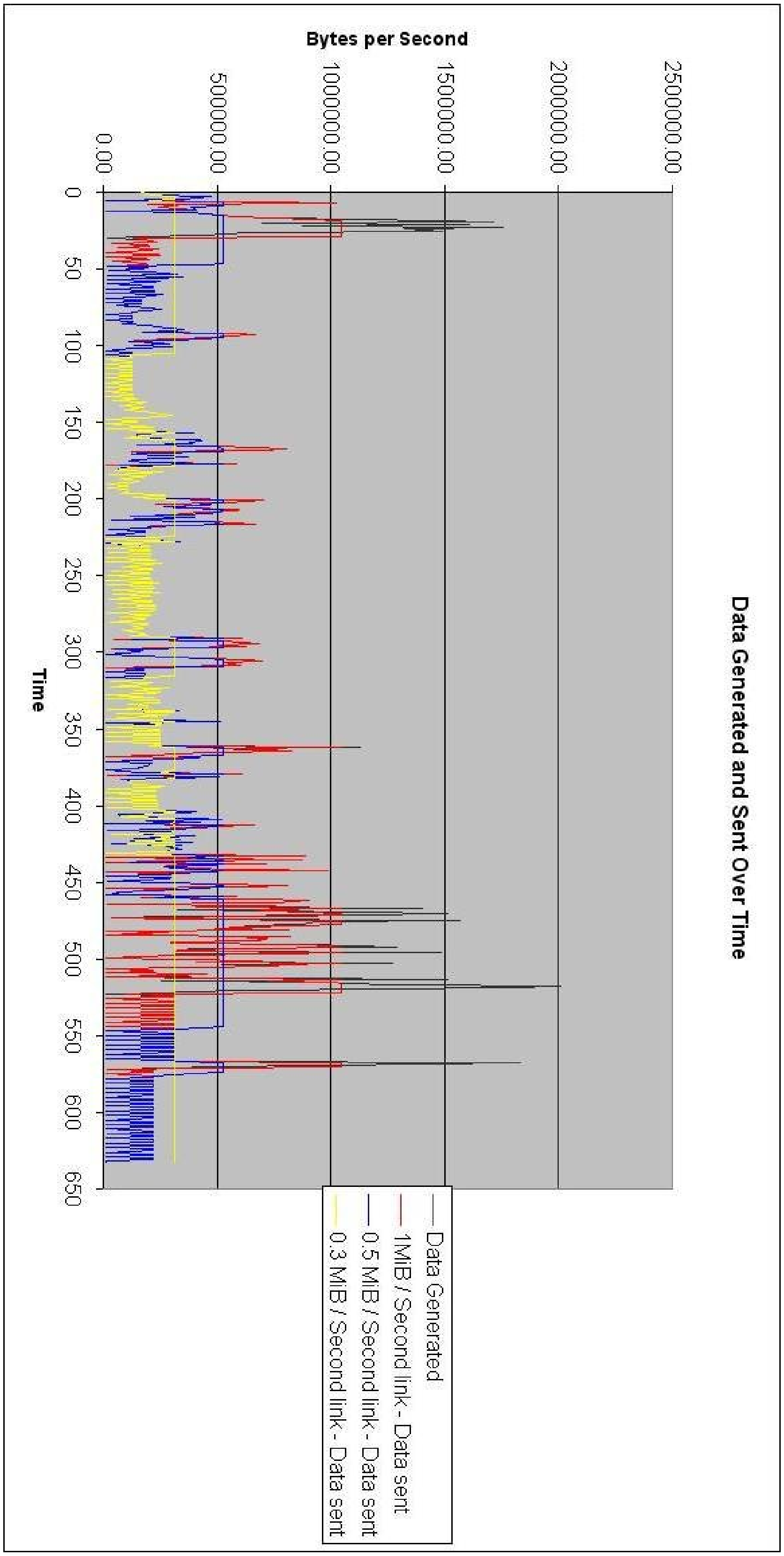


Figure 3-4. An mp4 video streamed over 1.0 MiB, 0.5 MiB, and 0.3 MiB links.

Figure 3-5 shows the size of the re-send priority queue over time in the same three scenarios from Figure 3-4. For this analysis, the size of the priority queue is calculated per time-slice based on excess data generated and excess link capacity. The value of excess data generated is calculated by subtracting the link capacity from the data generated. A positive value is representative of frame data in excess of link capacity, which the network can’t transmit on time, so for this analysis it is assumed that the excess data will need to be saved and retransmitted. For each time-slice, if the priority queue is greater than zero, and there is excess link capacity, there is room to retransmit the older frame data held in the priority queue. That retransmission is represented by subtracting the excess link capacity from the size of the priority queue. These idealized calculations are not based on transmission using the JSP protocol, but the scenarios are valid for conceptual analysis, to show that the protocol would have utility when applied to real-world data. Figure 3-5 shows that the PQueue buffer grows in periods of excess data generation, and shrinks as the excess link capacity allows for retransmission of the frame data lost during those periods of excess data generation.

In each scenario, the excess link capacity would allow the JSP to retransmit a significant amount of video data. In the 1.0 MiB per second scenario, during three percent (3%) of the time-slices, the video stream generated more data than the link could handle. The JSP would be able to reconstitute the full quality video by retransmitting 8.12 MiB of data. In the 0.5 MiB per second scenario, twelve percent (12%) of the time the video stream generated more data than the link could handle. The link would still have enough excess capacity to allow JSP to reconstitute of the full quality video by retransmitting 28.7 MiB. In the 0.3 MiB per second scenario, the capacity is sufficiently constrained such that the JSP would not have time to retransmit every dropped frame. In this example, over twenty one percent (21%) of the time the video stream generated data in excess of link capacity, and it would only be possible to re-send eighty five percent (85%) or 41.7 out of 49.2 MiB of the dropped data. This capacity starved scenario highlights why it is so important to prioritize the retransmission frame buffer in order to re-send the most important information first.

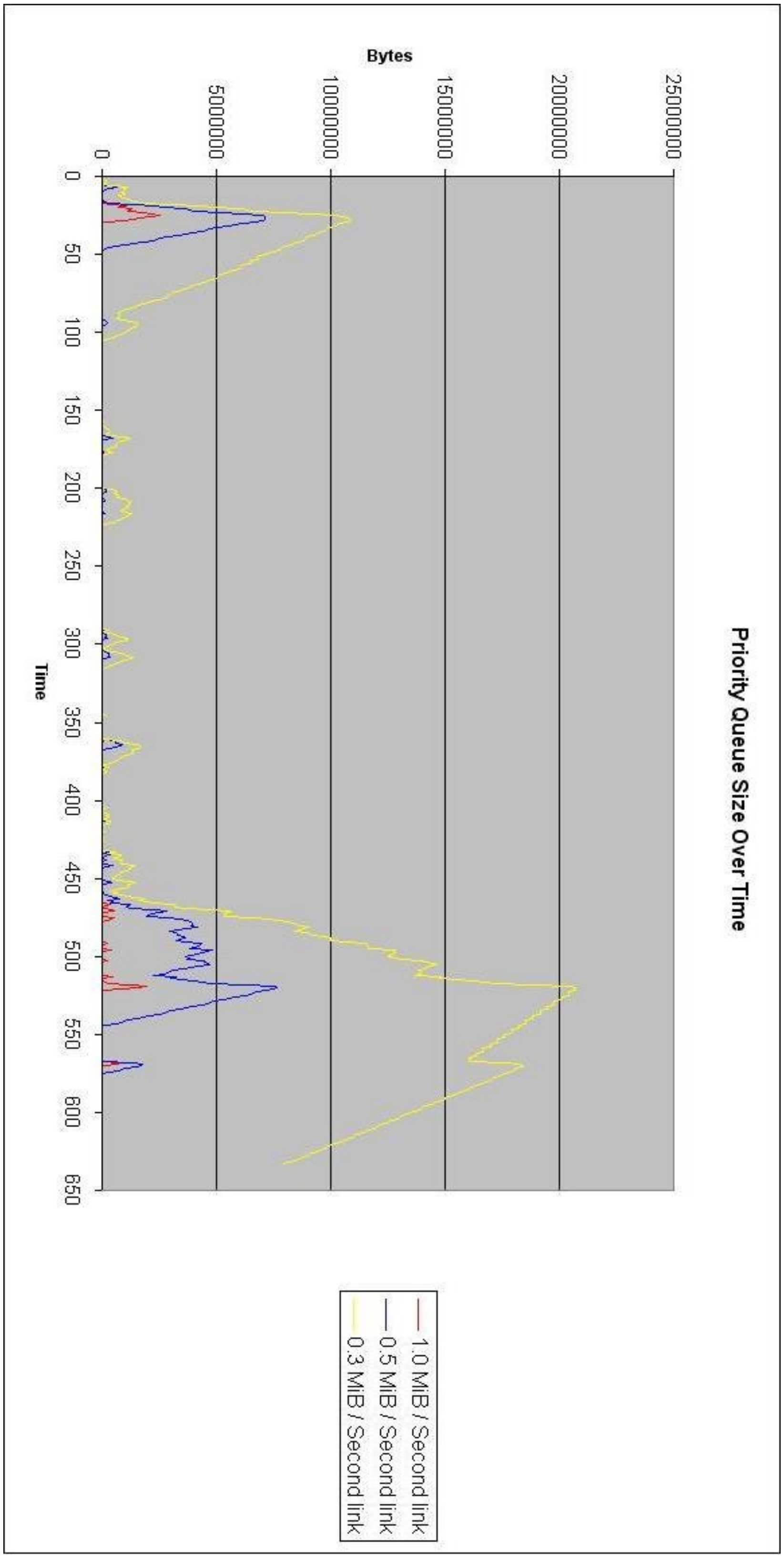


Figure 3-5. An mp4 video streamed over a .5MiB link. The re-send PQueues generated by an mp4 video when streamed over 1.0 MiB, 0.5 MiB, and 0.3 MiB links. The PQueue buffer grows in periods of excess data generation, and shrinks as the link capacity allows for retransmission of the frame data that the client didn't receive in time for the real-time stream.

Chapter 4

Conclusion and Future Work

This thesis introduces the problem of joint streaming for applications with different QoS requirements. A joint streaming protocol is proposed to addresses the QoS requirements of both real-time playback and higher quality storage of streaming media. The proposed joint streaming protocol is an important new solution to a problem that is of increasing importance in a multitude of application spaces. Initial results show that the proposed joint streaming protocol has the potential of effectively addressing this specific case by simultaneously supporting two different QoS requirements.

The current work is still preliminary. Future work could include designing and incorporating a more accurate and robust bandwidth estimation scheme. The packet pair algorithm from [1] works adequately in the experimental setting, but a more accurate and robust mechanism for available bandwidth estimation such as those proposed in [3][7][8] should be implemented. The passive technique described in [8] seems most promising because it would improve the efficiency of JSP by providing effective measurement of the available bandwidth without the overhead of dedicated bandwidth estimation packets.

The server buffer management should also be treated more rigorously to accommodate the requirements from the applications, e.g., by explicitly accounting for the delay constraint of real-time playback. A future implementation could also write the tail of the re-send buffers into secondary memory. This could relax the correlation between server buffer size and JSCP packet spacing, which is an issue even if no packets are lost in the real time stream. It could also allow for better handling of volatile network situations, by avoiding the truncation of oldest and least important packets, which could be dropped in the current implementation due to server buffer size constraints and preclude retransmission of a full quality archival stream.

To address the more generic scenario described in Section 1 of Chapter 2, an extension of the current work and a more rigorous treatment of the problem could be investigated.

References

[1] Kevin Lai and Mary Baker, “Measuring Link Bandwidths Using a Deterministic Model of Packet Delay,” In *Proceedings of ACM SIGCOMM*, August 2000.

[2] Vern Paxson. *Measurements and Analysis of End-to-End Internet Dynamics*. PhD thesis, University of California, Berkeley, April 1997.

[3] R.S.Prasad, M.Murray, C. Dovrolis, and K.C.Claffy, “Bandwidth Estimation: Metrics, Measurement Techniques, and Tools, *IEEE Network*, Nov. 2003. http://www.cc.gatech.edu/~dovrolis/Papers/NetDov0248.pdf.

[4] H. Schulzrinne, A. Rao, and R. Lanphier, “Real Time Streaming Protocol (RTSP),” IETF RFC 2326, April, 1998.

[5] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson, “RTP: A transport protocol for real-time applications,” IETF RFC 1889, January 1996.

[6] “The network simulator ns-2,” http://www.isi.edu/nsnam/ns/

[7] S. Floyd, et. al, “Equation-based congestion control for unicast applications,” *ACM SIGCOMM*, Sept. 2000.

[8] Zhang Jian-hui, Wang Bin, “Performance Analysis of Available Bandwidth Estimation Algorithm Based on EWMA and Kalman Filter,” 2009 International Conference on Multimedia Information Networking and Security, Nov. 2009.

[9] Mike Sullivan, Wenjun Zeng, “A Protocol for Simultaneous Real Time Playback and Full Quality Storage of Streaming Media,” 2005 IEEE International Conference on Communications, May 2005.

[10] “gstreamer the open source multimedia framework,” http://www.gstreamer.net/.

[11] M. Jain, C. Dovrolis, "Pathload: A measurement tool for end-to-end available bandwidth," Proc. Passive and Active Measurements (PAM) Workshop, Mar. 2002, pp. 14-25.

[12] V. Ribeiro, R. Riedi, R. Baraniuk, J. Navratil, L. Cottrell, “pathChirp: Efficient Available Bandwidth Estimation for Network Paths,” In Passive and Active Measurement Workshop, 2003.