ADAPTIVE LIVE STREAMING OVER ENTERPRISE NETWORKS

D. S. Turaga[1], A. Abd El Al[2], C.Venkatramani[1], O. Verscheure[1]
[1]IBM T.J. Watson Research Center, Hawthorne. [2]City University of New York
{turaga, chitrav, ov1}@us.ibm.com aabdelal@ieee.org

ABSTRACT

In this paper we present multimedia adaptation strategies for live video streams, at the streaming server, where we switch among several versions of the coded multimedia to match the available network bandwidth accurately, and meet client delay constraints. We estimate information about the available network bandwidth, at the server, by monitoring the application buffer and then decide to adaptively switch up or switch down the transmitted bit-rate. We use a piecewise linear model for the network bandwidth to estimate the current and future server buffer drain delay, and derive the transmission rate to minimize client buffer starvation. We implement these strategies in an enterprise streaming system and verify the performance, in terms of the network fidelity and received video quality, using both real as well as simulated network traces.

1. INTRODUCTION

The streaming of multimedia over enterprise networks is gaining momentum. While enterprise multimedia streaming faces many challenges similar to internet multimedia streaming, there are additional requirements and advantages specific to it. Firstly, an enterprise owns and can control the network, and thus issues of multimedia server placement, content distribution, enforcement of network policy all are specific to the enterprise network, and can differ significantly from the internet case. Secondly, since data in enterprise networks is commercially valuable, care needs to be taken to ensure that the multimedia streams are "friendly" to other streams, go through firewalls, and can be adapted to account for varying bandwidths and user requirements. In this paper we present server multimedia adaptation algorithms for Adaptive Rich Media Streaming (ARMS) over enterprise networks using TCP transport. TCP has been viewed as unsuitable for real-time traffic due to its lack of throughput guarantees and insistence on reliability [1] with an additive increase multiplicative decrease (AIMD) flow control. Recent work [2] has shown that TCP generally provides good streaming performance when the achievable TCP throughput is roughly twice the media bitrate, with only a few seconds of startup delay Of course, with TCP, the transport is ensured to be friendly to other flows sharing the same network, unlike the TCP-friendly implementations of other protocols, that cannot achieve friendliness at all time-scales. And more importantly, in many situations streaming over TCP is unavoidable, such as when the client machines are located behind network firewalls permitting only inbound HTTP traffic.

Multimedia adaptation has been studied for internet applications, and the adaptive control schemes can be classified into receiver-driven, sender-driven and transcoder-based. Receiver driven schemes allow receivers individually to tune the received transmission according to their needs and capabilities. Mehra and Zakhor [8] modify the TCP protocol at the receiver end to provide video streams a nearly CBR connection over a bandwidth limited access link. Hsiao *et al* [3] present Receiver-based Delay Control (RDC) in which receivers delay TCP ACK packets based on router feedback to provide constant bit rate for streaming. While receiver buffers can be used for smoothing out rate fluctuations, buffering is limited by the end-to-end latency limit.

A majority of the sender-driven algorithms may be grouped under quality adaptation schemes. Quality adaptation techniques can further be classified into on-the-fly encoding, adding/dropping layers, and switching among multiple encoded versions. Kanakia *et al* [5] estimate the buffer occupancy and the service rate received by the connection at the bottleneck queue through periodic feedback messages from the network. These estimates are used to control the transmission rate of each video frame on-the-fly by adjusting the encoder quantization factor. However, in general, on-the-fly encoding is CPU intensive and thus regarded as unsuitable for streaming live video. This is especially true when the sender has to service many different clients with different bandwidth requirements. In the adding/dropping layers scheme, the video stream is partitioned into several layers using scalable coding schemes such as MPEG-4 FGS or interframe wavelet video encoding. Video streaming applications can add or drop enhancement layers to adjust the transmission rate to the available bandwidth. In the switching-versions scheme, the video is encoded at different rates, and therefore different quality levels, and each of these versions is made available to the streaming server as an independent stream. The server detects changes in available bandwidth and switches among the input streams, in order to adapt the transmission rate to the available bandwidth. Quality adaptation that is based on multiple encoded versions has been shown to provide better viewing quality than adding/dropping layers, due to the layering overhead [4].

Besides quality adaptation, scheduling algorithms may also be used to improve the multimedia streaming

0-7803-9332-5/05/$20.00 ©2005 IEEE

adaptation. Saparilla and Ross [6] prefetch portions of the video into the client buffer as bandwidth is available. Transcoder-based schemes [9] decode and re-encode the video at gateways placed at appropriate locations to deliver different levels of quality to clients with different bandwidths.

In this paper we focus on sender-driven quality adaptation, for live video streams, to minimize any overheads at the client. In particular, we focus on quality adaptation using stream switching, as it has been shown to provide better viewing quality than adding/dropping layers, due to the layering overhead [4]. We introduce a mechanism for adaptive stream switching for live video that does not require either modifications to the network transport protocol at the sender or at the receiver, or support from the network infrastructure. The mechanism detects the variations of the network bandwidth through monitoring the application buffer occupancy, and accordingly adapts the video quality to ensure that the client buffer does not underflow and that the adaptation affects the perceptual quality at the client minimally.

This paper is organized as follows. We describe our system architecture in Section 2, and the adaptation mechanisms, including the network bandwidth measurement, switching down and switching up strategies in Section 3. We finally present results in Section 4 to validate these algorithms. We include a brief discussion on improving these results by modifying TCP buffers in Section 5, and conclude in Section 6.

2. SYSTEM ARCHITECTURE

We consider the *live* streaming of *adaptive* media streams over enterprise networks with *reliable* transport. Our scenario is illustrated by Figure 1. Multiple quality streams feed the adaptive media server. These streams may be independent encodings of the media at different bit-rates, resolutions, frame rates etc. Also, additional streams may be derived from each independent encoding by discarding parts of the stream. For e.g., from a stream with a GOP structure, IBPBP we can derive two additional sub-streams corresponding to lower frame-rates by: a) dropping all B and P frames b) dropping all B frames. The server selects one of those streams $r(t)$ and injects it into the server buffer. The server buffer is drained as fast as the reliable network connection permits, i.e. $x(t)$. The network output $x^*(t)$ is fed into the client buffer. The media player starts emptying the client buffer at the encoding rate $r(t)$ after a playback delay of D units of time.

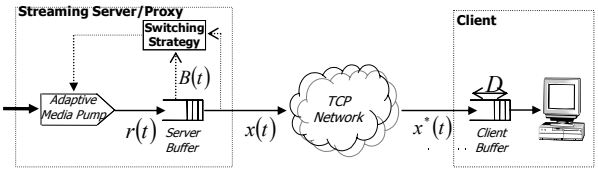

Figure 1: Streaming System Architecture

In this paper we present an optimized switching strategy at the streaming server that maximizes perceived quality at the client under varying network bandwidth conditions, given *N* constant bit rate (CBR) input media streams. We measure quality in terms of the achieved average bit rate, variations in the quality at the client, and data loss due to client buffer starvation which can result in frame drops or freezes in the video display, to allow for re-buffering.

3. STREAM SWITCHING STRATEGIES

A key requirement of our system is the estimation of network bandwidth. In the case of TCP-based streaming, this is not straightforward since TCP hides the network congestion status from the application. In this work we estimate the current available channel rate $\tilde{x}(t_k)$ at sampling instant t_k as the rate at which the server buffer empties. Clearly this estimation is highly variable (noisy measurements). Thus we calculate $x(t_k)$ as its exponential average; i.e. $x(t_k) = \rho x(t_{k-1}) + (1-\rho)\tilde{x}(t_k)$.

Consider that we have *N* available video streams (different encodings or derived sub-streams) with corresponding bit-rates V_j ($j = 1,2,\ldots,N$) and we make the decision to switch at discrete time instances t_k. In this discussion we have assumed that the stream bit rates do not change with time. This is true when the encoder has effective rate control, and all the data from the encoder reaches the server. When these assumptions are violated, we need to estimate these stream bit-rates. We present different strategies to make this decision:

3.1. Switching Down Strategies

Let *R, X, X** be the cumulative values of the flows *r, x and x**. That is, $R(T) = \int_0^T r(t)dt$ denotes the number of bits of the media flow $r(t)$ in *[0, T]*.

3.1.1. Instantaneous Decision Strategy

Client buffer starvation is avoided if $X^*(t) \geq R(t-D)$, for all time *t*. Thus, the condition on the media stream feeding the server buffer can be written as $R(t) \leq X^*(t+D)$. In general $X^*(t) = X(t) - tcpbuf(t)$, where *tcpbuf* corresponds to the TCP buffers as discussed in Section 5. We assume the number of packets in-flight is small in comparison to the server buffer size. Therefore, $X^*(t) = X(t)$ and the no-starvation condition becomes $R(t) \leq X(t+D)$. So, by monitoring the evolution of the network channel using $X(t)$ and by predicting the amount of data the network will remove from the server buffer in the time interval *(t, t+D]*, we can derive an optimized switch-down strategy. This prediction requires a model of the network channel bandwidth process; e.g. Gaussian autoregressive [1], fractional Brownian motion [2]. Here we adopt a very

simple, deterministic piecewise-linear model with $X(t_k + t^*) = X(t_k) + t^* x(t_k)$.

Note that our work can easily adopt a different network model. We illustrate the switch-down strategy in Figure 2.

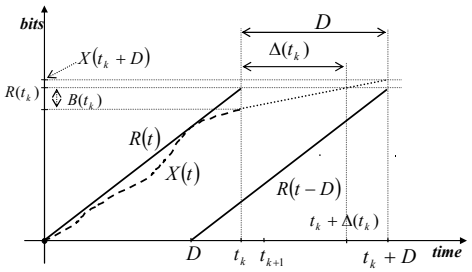

Figure 2: Decision for stream switching

We may equivalently formulate this decision strategy in terms of the buffer drain delay $\Delta(t_k)$, defined as $\frac{B(t_k)}{x(t_k)}$ under our simple network model assumption (see Figure 2). Hence, in order to prevent client buffer starvation, we want $\Delta(t_k) < D$. Hence, whenever we observe $\Delta(t_k) > \alpha D$, we reduce the input rate to the largest available rate smaller than $x(t_k)$. The conservative factor α ($0 < \alpha < 1$) is introduced to account for possible variations in the input and output rates during sampling interval $[t_k, t_{k+1})$. Hence, we select $r(t_k) = \max_{\substack{j=1,\ldots,N \\ V_j < x(t_k)}} \{V_j\}$. The factor α should be selected based on the expected variations in the rates.

3.1.2. Look Ahead Decision Strategy

Variations and inaccuracies in the measured rates can lead to the instantaneous strategy being too conservative, and switching down too aggressively, thereby under-utilizing the available bandwidth. In order to avoid this, we additionally look at the rate of change of $\Delta(t)$. If the measured output rate does not change significantly (especially as compared against the difference in rate between the current input rate V_j and the next lower rate V_k) and we observe that $\Delta(t)$ is decreasing, then we would like to wait before switching further down. Since $\Delta(t_k) = \frac{R(t_k) - X(t_k)}{x(t_k)}$, we may compute the rate of change $\Delta'(t_k) \approx \frac{r(t_k) - x(t_k)}{x(t_k)}$ (assuming that $x(t_k) \approx x(t_{k-1})$). Using this we may estimate the buffer drain delay in the future as $\Delta(t_{k+1}) = \Delta(t_k) + \Delta'(t_k)(t_{k+1} - t_k)$. If $\Delta(t_{k+1}) < \beta D$, where β ($0 < \beta < 1$) is another conservative factor introduced to account for possible variations in the input and output rates, we should not switch down any further, otherwise we should continue to switch down. Using the above equation, we can also determine the input rate to switch down to as $r(t_k) \le \frac{\beta D x(t_k) - B(t_k)}{(t_{k+1} - t_k)} + x(t_k)$. This strategy can avoid unnecessarily aggressive reductions and stream switches in the input rate by sometimes borrowing from, and sometimes provisioning for the future. However, it also makes assumptions that the output rate does not change significantly over the interval $[t_k, t_{k+1})$. Hence, when the timescale of network variations is smaller than the sampling interval (i.e. the network conditions change rapidly) the instantaneous decision is likely to outperform the look-ahead decision, and vice-versa.

3.1.3. Combined Decision Strategy

We may combine the benefits of these two decision strategies, and the algorithm may be written as:

1. Compute buffer fullness $B(t_k)$ and estimate $x(t_k)$. Set $r(t_k) = r(t_{k-1})$.
2. Estimate $\Delta(t_k)$ and $\Delta(t_{k+1})$
3. If $\Delta(t_k) > \alpha D$ and $\Delta(t_{k+1}) > \beta D$
Determine upper bound on input rate

$$r^{Max} = \max\left\{\left(\frac{\beta D x(t_k) - B(t_k)}{(t_{k+1} - t_k)} + x(t_k)\right), x(t_k)\right\}$$

Select $r(t_k) = \max_{\substack{j=1,\ldots,N \\ V_j < r^{Max}}} \{V_j\}$

Else
Test whether we can switch up
4. Wait until next sampling interval. Goto Step 1.

Figure 3. Combined switching down decision strategy

3.2. Switching Up Decision Strategy

We cannot switch up the streaming rate rapidly as it can actually create congestion in the network and thereby lead to oscillations between switching up and switching down. However, as there is no explicit signal indicating when the server should switch up, our mechanism performs active experiments by probing the network to ensure that there is enough capacity for the next higher streaming rate. We call these experiments, *switch-experiments*. The *switch experiment* is triggered whenever the server does not experience a congestion event for an interval T_E^i referred to as the *Inter-Experiment timer*.

The server then switches to the next higher available streaming rate, such that $r(t_k) = \min_{\substack{j=1,\ldots,N \\ V_j > r(t_{k-1})}} \{V_j\}$ and each experiment lasts for a maximum duration of T_S. During the experiment the server continues to monitor the network and if no congestion is caused, due to the experiment, the server stays at the higher rate. However, if congestion is detected, as indicated in Step 3 of the combined switch down algorithm, the sender reverts to the lower rate. The sender also learns from failed experiments by exponentially backing off the T_E^i for this rate, before retrying the experiment. The exponential back-off is performed as $T_E^{i+1} = \min(\gamma T_E^{i+1}, T_E^{\max})$ where $T_E^{\max}$ is the

maximum *Inter-Experiment* timer, and γ is a back-off factor. We clamp the back-off at a maximum to guarantee the sender will periodically probe for spare bandwidth. The *Inter-Experiment* timer of the new stream is reset to initial value T_E^{init}, when the switch experiment to this stream succeeds. The switching experiment duration T_S starts with an initial value T_S^{init} and is updated using an exponential moving average of the time difference between starting a switching experiment to the failure detection timeIn order to summarize the description of our overall adaptation strategy, we represented it as a flow diagram as in Figure 4.

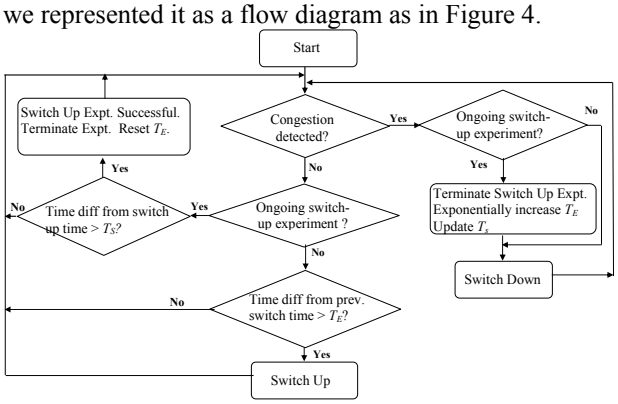

Figure 4. Flow diagram for adaptation

In the flow diagram we omit details for simplicity. For instance, we determine the rate to switch down to as in Step 3 of the combined switch down algorithm.

3.3. Determining the Adaptation Parameters

The performance of our adaptation strategies is controlled by a set of different parameters that include the sampling interval, the buffer drain time parameters α and β, the switch up times T_S^{init} and T_E^{init}, and the exponential back-off parameter γ. We should keep our sampling interval small so that we can effectively track the network bandwidth variations. However, a small sampling interval leads to larger overheads in system complexity and transmitted bandwidth (packets within a sampling interval can be aggregated and transmitted together to reduce network overheads.). In order to tradeoff these conflicting goals, we select the sampling interval based on the available network bandwidth; sample at small intervals when the network bandwidth is high (variations are likely to be more frequent) and sample at larger intervals when the network bandwidth is low (variations likely to be less frequent). We can do this by sampling every time we transmit a fixed number of bytes. Empirically, we have determined that sampling every time we transmit ~16000 bytes (since we transmit complete packets) provides a good tradeoff for the adaptation. The other adaptation parameters have been tuned empirically to provide a good visual quality, however we can derive analytical bounds on their values based on the statistical properties of the network and video bit-rates. This is a direction of future research.

4. RESULTS

In order to examine our proposed mechanism, we implemented the adaptation mechanism within the IBM's Adaptive Rich Media Streaming system [9]. Our testbed consists of a live source attached to an ARMS broadcaster that is connected to the video server via 100 Mbps Ethernet. The client connects to the server via a NIST Net box which is used to control the bandwidth between the server and the client. The testbed is shown in Figure 5.

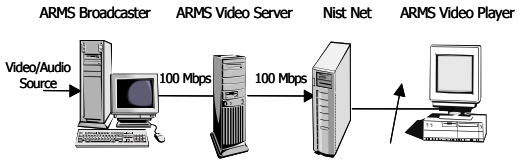

Figure 5. Performance study testbed

At the broadcaster, we use two independent encodings of 320×240 video, one at 512 Kbps and 15 frames per second (fps), and the other at 260 Kbps at 10 fps. From these we derive multiple sub-streams, by dropping frames, with bit-rates V_j (512, 417, 334, 255, 251, 213, 85 Kbps). The audio bit rate is fixed and equal to 32 Kbps. The adaptation mechanism parameters are: $N = 3$, $D = 3$ sec, $\gamma = 2$, $T_E^{init} = 10$ sec, $T_E^{\max} = 60$ sec, $T_S^{init} = 10$ sec. Additionally, we select the parameters $\alpha = 0.4$, and $\beta = 0.5$.

We generate random traces as sequences of independent and identically distributed (iid) random variables (each corresponding to a fixed bandwidth for a duration of 15 seconds) generated from a uniform distribution with values {200, 400, 600} corresponding to intermediate bit-rates in our available streams. We present sample results (~500 seconds) to highlight the performance of our adaptation and compare the instantaneous and combined decision strategies.

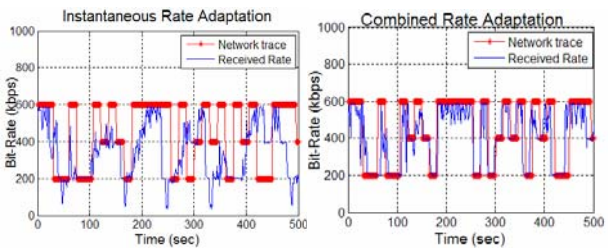

Figure 6. Adaptation performance for random trace

In the graphs we plot the network trace against the measured bandwidth at the client that uses a 2 second averaging moving window with overlaps of 1 second. As expected, the instantaneous decision leads to over-aggressive switching down, thereby not following the network trace accurately, unlike the combined decision strategy. However, as we have mentioned before, bandwidth fidelity is not the only performance metric we evaluate. We also measure the number of stream switches and the number of dropped packets (due to server buffer overflow). The combined adaptation outperforms the instantaneous

adaptation both in terms of the achieved bandwidth as well as in terms of a smaller number of stream switches. However, it has a larger number of lost packets. We quantify these results for a real network trace that was collected over 70 minutes with 50 active TCP connections and the bandwidth varied between 700 Kbps and 120 Kbps. At the broadcaster, we again create two independent encodings of 320×240 video, one at 512 Kbps and 15 fps, and the other at 260 Kbps at 10 fps. The multiple sub-streams with bit-rates V_j are shown in Table 1.

Table 1. Generated streams for real trace experiment

Encoding	Sub-Stream	GOP Structure	Bit-Rate (kbps)
1	1	I B B B P B B B P…	255
	2	I B X B P B X B P…	213
	3	I X B X P X B X P….	171
	4	I X X X P X X X P…	129
	5	I X X X X X X X P…	85
2	1	I B B P B B P…	512
	2	I B X P B X P…	417
	3	I X X P X X P…	334

The audio stream is again maintained at 32 Kbps. We show the adaptation performance over 3000 seconds in Figure 7.

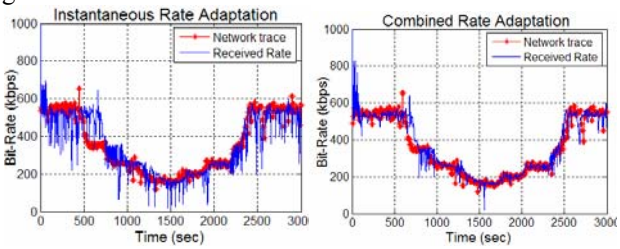

Figure 7. Adaptation performance for real trace

Clearly, the instantaneous decision strategy is overly conservative leading to many switches down and under-utilization of the available bandwidth. We now present the number of stream switches and the number of dropped packets in Table 2.

Table 2. Adaptation performance results for real trace

		Inst. Adapt.	Comb. Adapt.
Achieved Average Bit-Rate (kbps)		355	380
Number of Stream Switches		324	118
Data Loss	Number of Lost Packets (Audio and Video)	65 (0.14%)	459 (0.8%)
	Number of lost video frames	7	92

The combined strategy has better achieved bandwidth as well as fewer stream switches, but has more lost frames. However, a majority of frames lost are B frames, and hence do not contribute to any error propagation. The number of lost P frames (37) is smaller than the number of times that the instantaneous decision switches down to an all I channel, or worse to only audio, which means that the visual interruptions for the combined decision are fewer than for the instantaneous strategy. We also use subjective quality assessments to validate these observations. A direction of further research is combining these numbers into one visual quality metric.

5. CONTROLLING TCP BUFFERS

Buffers that the TCP implementation maintains lead to the adaptation mechanism getting delayed feedback from the network. The TCP sender uses a congestion window indicated by the variable *cwnd* to estimate the appropriate congestion window size and also a fixed size send buffer (*sendbuf*) to store application data before the data is transmitted. We propose to dynamically limit the TCP *sendbuf* size. This is possible on Linux and Windows using the setsockopt system call. We chose to set it to be 2**cwnd,* which ensures that the TCP has a window worth of unsent data to keep the self-clock of acknowledgments flowing. The proposed modification can be implemented at the application level by monitoring *cwnd* and adapting *sendbuf* size and ensuring that the sending buffer occupancy does not exceed 2**cwnd*. This enables us to adapt more accurately to network conditions. To examine the combined adaptation mechanism with the sender buffer scaling optimization, we implemented the mechanism in Opnet network simulation tool [14]. We chose to add this feature to the TCP implementation and determined that it does not affect the TCP throughput for other connections, as also shown by other authors [13]. We used a topology where the server streams to a client over a bottleneck link of 5Mbps bandwidth and 10ms round trip time, shared with n other TCP sources.

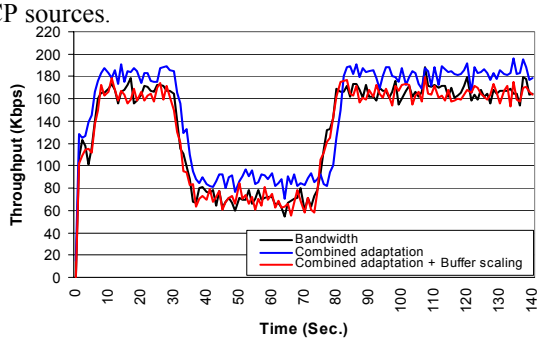

Figure 8. Stream-rate with adaptive TCP buffers.

We examined the adaptability of our proposed mechanism. To vary the bandwidth between server and client, we used different number of TCP connections (20-40) for simulation times varying between 0 and 70 minutes. The figure above shows the available TCP bandwidth and the video stream rate received at the client. We compared the stream rate achieved by the proposed adaptation mechanism with and without the TCP sender buffer scaling. The results show that although the adaptation mechanism without the buffer scaling is able to track the available bandwidth, using the buffer scaling will allow the application to track the available bandwidth more accurately, validating our hypothesis.

6. CONCLUSIONS

We develop server adaptation mechanisms, for live multimedia streaming over enterprise networks (using TCP

for transport). The server estimates information about the available network bandwidth by monitoring the application buffer and performs stream switching to meet bandwidth and delay constraints. We investigate instantaneous and look-ahead strategies for switching down the transmission rate, and switch up the rate in a controlled manner after observing periods of no congestion. We use a piecewise linear model for the network bandwidth to estimate the current and future server buffer drain delay, and derive the transmission rate to minimize client buffer starvation. We evaluate these algorithms over an enterprise video streaming system based on the IBM VideoCharger platform. We compare the performance of these algorithms in terms of their effect on the decoded video quality by measuring the average streaming bandwidth achieved by the algorithm, the number of stream switches, and the data loss (caused due to server buffer overflow). The strategy with combined look-ahead and instantaneous decisions can follow the network bandwidth accurately, while minimizing stream switches, and providing high visual quality.

Directions for future research include developing analytical methods to determine the parameters of our adaptation algorithms, including α, β, the switch up times T_S^{init} and T_E^{init}, and the exponential back-off parameter γ. We are also tuning the parameters of the adaptation to be able to receive information from TCP with less of a delay. Finally, we are investigating methods of integrating these different performance metrics into one visual quality metric that is consistent with subjective quality evaluations.

REFERENCES

[1] C. Krasic, K. Li, J. Wapole, "The case of streaming Multimedia with TCP", Proceedings of the 8th International Workshop on Interactive Distributed Multimedia Systems (IDMS), 2001.

[2] B. Wang, J. Kurose, P. Shenoy and D. Towsley, "Multimedia Streaming via TCP: An analytic performance study," Proceedings of ACM Multimedia, New York, October 2004.

[3] P. Hsiao, H. Kung, K. Tan, "Streaming Video over TCP Receiver-based Delay Control", Proceedings of ACM NOSSDAV, 2001.

[4] P. Cuetos, D. Saparilla, K. Ross, "Adaptive Streaming of Stored Video in a TCP –Friendly Context: Multiple Versions or Multiple Layers?", International Packet Video Workshop, Korea, April 2001.

[5] H. Kanakia, P. Mishra, A. Reibman, "An adaptive congestion control scheme for real-time packet video transport", SIGCOMM Symposium on Communications Architectures and Protocols, California, Sep. 1993.

[6] D. Saparilla and K. Ross, "Streaming Stored Continuous Media over Fair-Share Bandwidth", *NOSSDAV 2000*, Chapel Hill, 2000.

[7] M. Jain, C. Dovrolis, "End-to-End Available Bandwidth: Measurement, Methodology, Dynamics, and Relation with TCP Throughput", Proceedings of SIGCOMM, Pennsylvania, 2002.

[8] P. Mehra and A. Zakhor, "TCP-Based Video Streaming Using Receiver-Driven Bandwidth Sharing", Proceedings of the 13th International Packet Video Workshop, France, April 2003.

[9] E. Amir, S. McCanne, H. Zhang, "An application level video gateway", Proceedings of ACM Multimedia, San Francisco, Nov. 1995.

[10] C. Venkatramani, P. Westerink, O. Verscheure, P. Frossard, "Securing Media for Adaptive Streaming", Proceedings of ACM Multimedia, California, Nov. 2003.

[11] Nist Net Network Emulator, http://snad.ncsl.nist.gov/itg/nistnet/.

[12] H. Radha, M. van der Schaar, Y. Chen, "The MPEG-4 Fine-Grained Scalable video coding method for multimedia streaming over IP," *IEEE Transactions on Multimedia*, vol. 3, no. 1, pp. 53-68, March 2001.

[13] J. Semke, J. Mahdavi, M. Mathis, "Automatic TCP Buffer Tuning," Computer Communication Review, ACM SIGCOMM, volume 28, number 4, Oct. 1998.

[14] Opnet Network Simulator, www.opnet.com