Adaptive Media Playout for Low Delay Video Streaming over Error-Prone Channels *

Mark Kalman, Student Member IEEE, Eckehard Steinbach, Member IEEE, and Bernd Girod, Fellow IEEE

Information Systems Laboratory
Department of Electrical Engineering
Stanford University
{mkalman,bgirod}@stanford.edu
steinb@lkn.ei.tum.de

October 25, 2002

Abstract

When media is streamed over best-effort networks, media data is buffered at the client to protect against playout interruptions due to packet losses and random delays. While the likelihood of an interruption decreases as more data is buffered, the latency that is introduced increases. In this paper we show how Adaptive Media Playout (AMP), the variation of the playout speed of media frames depending on channel conditions, allows the client to buffer less data, thus introducing less delay, for a given buffer underflow probability. We proceed by defining models for the streaming media system and the random, lossy packet delivery channel. Our streaming system model buffers media at the client, and combats packet losses with deadline-constrained ARQ. For the channel, we define a two-state Markov model that features state-dependent packet loss probability. Using the models, we develop a Markov chain analysis to examine the trade-off between buffer underflow probability and latency for AMP-augmented video streaming. The results of the analysis, verified with simulation experiments, indicate that AMP can greatly improve the trade-off, allowing reduced latencies for a given buffer underflow probability.

*This work has been supported in part by a gift from Intel Inc., and, in part, by the Stanford Networking Research Center.

1 Introduction

This paper investigates a new method of reducing the latencies that are inherent in systems that stream packetized media over best-effort packet networks. These systems strive to allow the immediate viewing of media data as it is delivered from a remote server. In practice, however, the systems must buffer an amount of media at the client to prevent packet losses and delays from constantly interrupting the playout of the stream. While the likelihood of a playout interruption decreases as more data is buffered, the delays that buffering introduces increase. For this reason, system designers must trade the reliability of uninterrupted playout against delay when determining the amount of data to buffer. Designers of today's commercial media streaming products find, for example, that buffering delays ranging from 5 to 15 seconds typically strike a good balance between delay and playout reliability [1],[2]. In contrast, viewers accustomed to traditional broadcast television expect playout to be immediate and program changes to be instantaneous.

Adaptive Media Playout (AMP) allows the client to buffer less data and thus introduce less delay to achieve a given playout reliability. In this scheme, the client varies the rate at which it plays out audio and video according to the state of its playout buffer. Generally, when the buffer occupancy is below a desired level, the client plays media slowly to reduce its data consumption rate. In the case of live streaming, slowed playout will cause viewing latency to increase. In this case, faster-than-normal playout may be used during good channel periods to eliminate any excess latency accumulated with slowed playout. Faster-than-normal playout is unnecessary in the case of pre-stored programs, however. Pre-stored programs that are slowed during bad channel periods will simply last longer at the client. By manipulating playout speeds AMP can reduce initial buffering delays in the case of pre-stored streams, and reduce the viewing latency of live streams - all without sacrificing playout reliability.

To control the playout speed of media, the client scales the duration that each video frame is shown, and processes audio [3],[4] to scale it in time without affecting its pitch. We find that variations in the media playout rate are subjectively less irritating than playout interruptions and long delays. Informal tests have shown that playout speed variations of up to 25% are often un-noticeable, and depending on the content, rate variations up to 50% are sometimes acceptable. Also, note that playout speed modification has a precedent in traditional media broadcasting, albeit not for the purpose of source rate control: motion pictures shot at a frame rate of 24 fps are shown on European PAL/SECAM broadcast television at 25 fps. This is a constant speedup of 4.2% and it is done without audio time scale modification.

In this paper we characterize the delay versus buffer underflow probability performance of a streaming system under varying AMP policies and channel conditions. Towards this

end we define simplified models for the streaming system and the lossy packet delivery channel. We then present the results of Markov chain analysis and of simulation experiments that were performed using these models.

Retransmission of lost media packets is essential for a video streaming application over error-prone channels. Continuous video playout at the receiver can only be guaranteed if all packets are available due to the interdependency of successive video packets introduced by motion compensated prediction in modern video encoding schemes like MPEG-2, MPEG-4, or H.26L. Explicit modeling of a retransmission protocol in the context of a Markov chain based analysis is challenging. As pointed out, for instance, in a different context in [5], explicit handling of retransmissions requires an intractably large Markov chain state space. We circumvent this problem with the assumption that retransmission of lost packets during adverse channel conditions can be modeled as a reduction in throughput. In our experimental results we show that this assumption holds over a wide range of parameters by comparing our analytical results with simulations that do explicitly model retransmissions.

To our knowledge adaptive media playout for media streaming systems is relatively unexplored. The combination of adaptive speech playout and time-scale modification has recently been proposed for packet voice communication [6],[4],[7]. In comparison to [4] and [7] we consider adaptive playout of video as well as audio data and relax the stringent delay constraints imposed for real-time voice transmission so that multiple retransmissions of lost packets can be afforded. The work that is most similar in the literature is the video playout smoother proposed in [8]. The analysis developed in [8] simplifies a two-state source channel model by averaging its behavior over its stationary distribution, does not consider the state of the server in the case of live streams, and does not consider retranmsissions, or audio. In [9] Steinbach et al. present an analysis for Adaptive Media Playout for video streaming that uses a two-state, on-off, burst-error, Markov channel model. The analysis in [9] is restricted to live streaming with the aim of minimum mean delay. In comparison, the analysis in this paper allows a more general channel model with packet loss probabilities modulated by a two-state Markov model. In addition to live streaming, we analyze the streaming of stored programs with AMP as a means to reduce the initial pre-roll time.

This paper is organized as follows. In Section 2 we introduce Adaptive Media Playout (AMP) for low latency media streaming. In Section 3 we first specify our channel model and our streaming media system. We then develop a Markov chain analysis for AMP streaming systems. In Section 4 we report our experimental results.

2 Adaptive Media Playout

AMP is the client-controlled variation of the playout speed of media. It allows the client to adjust its data consumption rate without the immediate involvement of the server. An AMP-enabled client controls the playout rate of video by simply scaling the duration that video frames are shown [8]. For audio, signal processing techniques such as the enhanced WSOLA time-scale modification algorithm proposed in [4] allow low-latency, packet-by-packet changes in the playout speed while preserving the original pitch of the audio.

As observed in [10], AMP can be used outright as a form of source rate scalability. When a stored media program is encoded at a higher rate than a user's connection can support, for example, the client can slow its playout rate so that it only consumes data at a rate supportable by the channel. In this paper, however, we focus on the case that the mean of the channel's fluctuating transmission rate is just sufficient to stream a desired source, but buffering must absorb short-term dips in the channel capacity. In this case, video quality is strongly dependent on the buffer size and the corresponding delay.

There are two buffering delays that are noticeable to the user: *pre-roll delay* and *viewing latency*. Pre-roll delay is the time that it takes for the client buffer to fill to a desired level so that playout can begin after a user request. Viewing latency, noticeable in the case of live streams, is the time interval separating a live event and its viewing time at the client. To explain how AMP can be used to reduce these delays, we will distinguish among three separate modes of use, illustrated in Figs. 1, 2, and 3.

The first mode, AMP-Initial, is used to reduce pre-roll delay. In this mode, the client begins playing media before the buffer is filled to the usual target level, and allows the buffer to fill to the target level over time by initially playing the media slower than normal. The buffer fills over time since the data consumption rate during slowed playout is smaller than the arrival rate of the media packets, assuming that during normal playout the source rate and the channel goodput match the data consumption rate at the decoder. Once the target level is reached, the playout speed returns to normal. This technique allows fast switching between different programs or channels without sacrificing protection against adverse channel conditions, after the initial buffer is built up.

Fig. 1 illustrates AMP-Initial. The top plot in Fig. 1 shows the assumed source rate and channel goodput as a function of time. For illustration purposes, the source rate, the channel goodput and the consumption rate of the playout process at normal playout speed are 0.1 Mbit/s. The second plot in Fig. 1 shows the client buffer occupancy as a function of time for the case of non-adaptive playout. We assume a target buffer level of 1 Mbit, yielding a pre-roll time of 10 seconds in this example. The third plot illustrates the client buffer occupancy for the AMP-Initial scheme in which playout starts when the

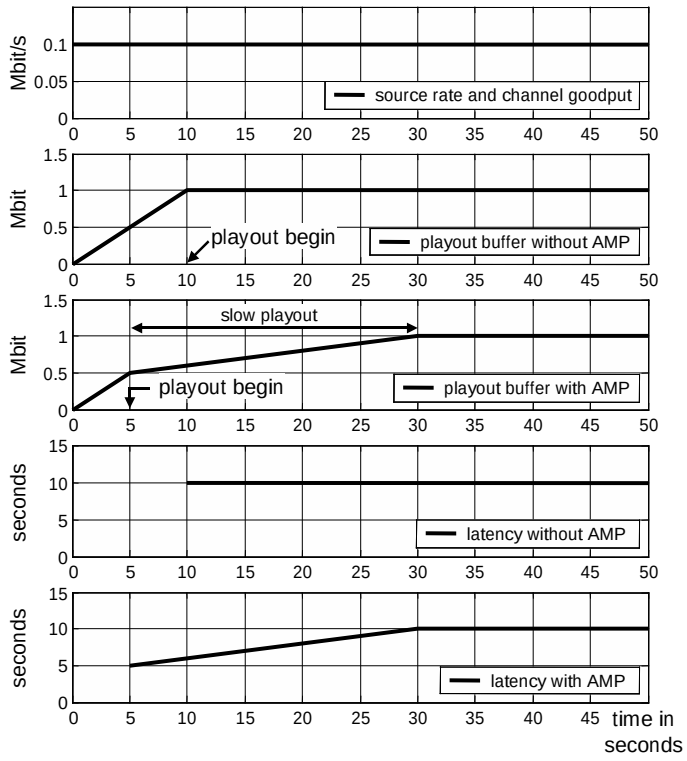

Figure 1: AMP-Initial: For low pre-roll delays, playout begins after only a small number of frames are buffered at the client. Slow playout allows the buffer occupancy to grow to a safer target level over time. In this example frame periods are stretched by 20% during slow playout periods.

buffer occupancy is only half the target level. In our example this occurs after 5 seconds. The client slows playout initially to allow the buffer occupancy to increase over time. After a total of 30 seconds the target buffer level is reached. The two lower plots in Fig. 1 show the viewing latency for the non-adaptive and the adaptive case. While the latency remains constant for the non-adaptive case, for the AMP scheme latency increases from 5 seconds initially to 10 seconds when the target buffer level is reached.

The second mode, AMP-Robust (see Fig. 2), increases the robustness of the playout process with respect to adverse channel conditions. In this mode the playout speed is simply reduced whenever the buffer occupancy falls below a threshold. Fig. 2 illustrates. The top plot in Fig. 2 shows the source rate and channel goodput as a function of time. As before, the source rate is a constant 0.1 Mbit/s. The channel goodput varies over time with a reduction to 0.05 Mbit/s at 15 seconds, an improvement to 0.133 Mbit/s at 25 seconds, and a complete channel outage at 40 seconds. The second plot shows the buffer occupancy as a function of time for non-adaptive playout. The target buffer level is again 1 Mbit which leads to playout start at 10 seconds. Playout is interrupted, however,

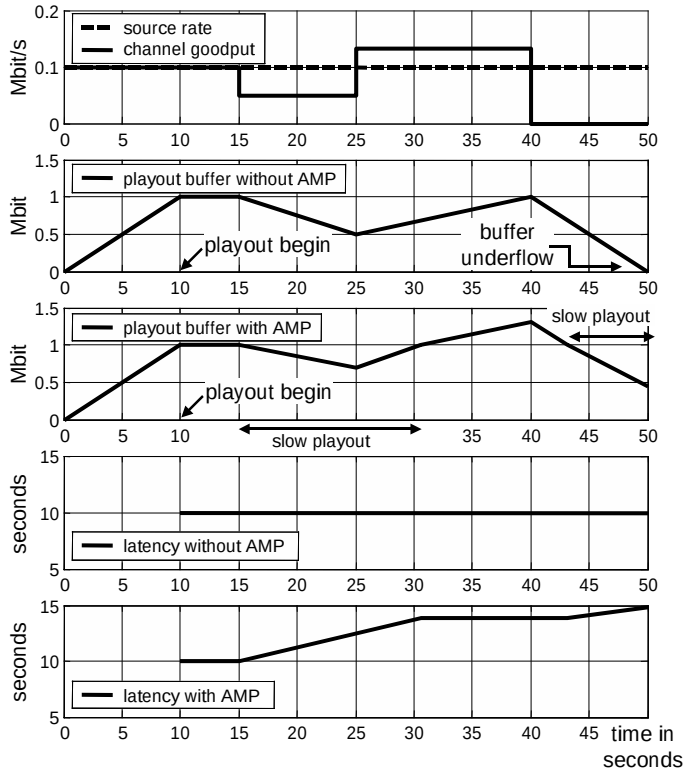

Figure 2: AMP-Robust: In this scheme, suitable for pre-stored programs where viewing latency is not important, slowed playout is used to keep the buffer occupancy at a desired level.

after 50 seconds, when reductions in the channel goodput lead to a buffer underflow. The third plot shows the buffer occupancy for the AMP scheme in which the client stretches frame periods by by 25% whenever the buffer occupancy falls below the target level. In this example, buffer underflow is averted with AMP. The lower two traces in Fig. 2 plot the viewing latency as a function of time. For non-adaptive playout, latency is constant. For the adaptive case, the latency increases whenever playout is slowed, which is fine for a pre-stored program. Note that playout starts at 10 seconds for both cases, however, AMP-Robust can be combined with AMP-Initial to also allow reduced pre-roll time.

The third mode, AMP-Live (see Fig. 3), is suitable for the streaming of live programs. In this mode, the client slows playout during bad channel periods but may also play media faster than normal during good channel periods to eliminate any viewing latency that has accumulated during periods of slowed playout. By playing the media faster and slower than normal, the mean viewing latency can be reduced for a given probability of buffer underflow. Fig. 3 illustrates. For AMP-Live, whenever the buffer occupancy falls below the target level, playout is slowed. When the occupancy is greater than the target level, media is played faster than normal to eliminate excess latency. In this example, during faster playout the client reduces frame periods by 25%, which corresponds to an increase

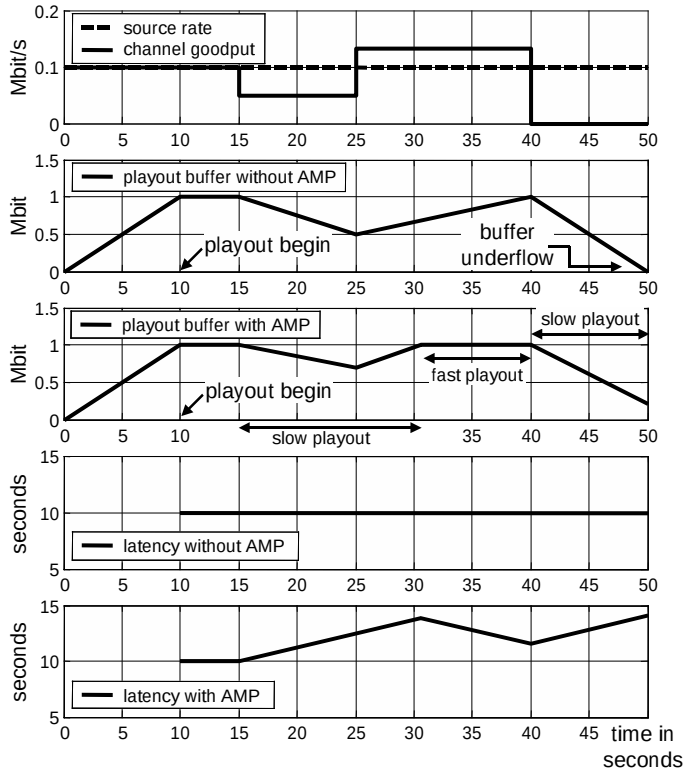

Figure 3: AMP-Live: For live streams, low viewing latency is desirable. In this scheme the client slows playout when poor channel conditions threaten to starve the client buffer. During good channel periods, however, faster-than-normal playout is used to eliminate latency accumulated with slowed playout.

in 33% increase in the data consumption rate. Therefore, the buffer remains at the target level in the third plot of Fig. 3 during fast playout. Latency, shown in the bottom plot of Fig. 3, decreases during faster-than-normal playout.

3 Analysis

In this section we develop techniques to analyze a streaming media system with respect to latency, pre-roll delay and probability of buffer underflow. We begin in Sections 3.1, and 3.2 by specifying the channel and the system models that we assume in our analysis. With these established, we then present our Markov queuing treatment in Section 3.4. Throughout our analysis, we will be making the simplifying approximation that frames of media are of uniform size, and media packets each contain one frame.

3.1 Channel Model

Our channel model features a fixed raw packet transmission rate and a two-state Markov process that modulates the packet error probability. Packet losses occur randomly with a loss probability that is dependent on whether the channel is in the good or the bad state. The model, a Gilbert-Elliott packet loss model[11] , allows us to investigate the effects of bursty packet losses on the buffering performance of a streaming system, but it does not simulate the variations in packet inter-arrival time known as delay jitter. With buffering delays on the order of seconds, and round-trip times on the order of hundreds of milliseconds, however, the effects of delay jitter, typically on the order of tens of milliseconds, are minor compared to those of packet loss.

Let the channel model be characterized by parameters

$$(R, t_{prop}, T_G, T_B, p_G, p_B) \tag{1}$$

where R is the raw transmission rate, the number of packetized, uniformly-sized frames of media that can be transmitted per normal frame period, t_F. For instance $R = 2.0$, means that two frames of media can enter the channel per normal frame period. T_G and T_B are the mean channel state durations, and p_G and p_B are the packet loss probabilities, for the good and bad states, respectively. The channel remains in the good and bad states for random holding times that are distributed exponentially with parameters $1/T_G$ and $1/T_B$, respectively. Finally, t_{prop} specifies the one-way propagation time from server to client.

While our simulation experiments implement the continuous-time channel model as described above, in our analysis we use a discrete-time version of this channel. In discrete time the exponentially distributed holding times are approximated with geometrically distributed times with the approximation growing exact as we increase the number of discrete slots per unit time [12]. Throughout, we assume that packet loss and channel state transition events are independent given the state of the channel. Again, we assume throughout that the frames of media are of uniform size (in bits), and that each frame represents t_F seconds of the program at normal playout rates.

3.2 Streaming Media System

As shown in Fig. 4, our system model consists of a source, a server, a channel, and a client. The source generates uniformly-sized frames and passes them to the server. If the source is a live program, throughout the streaming session it passes a new frame to the server every t_F seconds. If the source is a pre-stored program, all of the frames are transferred to the server at the start of the session.

When a frame is transferred to the server, it is given a sequence number, placed in a packet, and queued in the transmission queue (TX). A copy of the packet is also kept in the packet store in case it needs to be retransmitted later. When a retransmission request is received, the requested packet is placed in the retransmission queue (RTX) in sorted order according to sequence number. When packets are waiting, every t_F/R seconds one is transmitted over the channel with packets in the RTX queue given priority over those in the TX queue. Once in the channel, packets either arrive at the client after a fixed propagation delay, or are lost with probability p_s, $s \in \{G, B\}$, the channel-state dependent loss probability.

At the client, playout begins when the backlog in the playout queue reaches a specified threshold. After playout begins, the playout queue is serviced at a rate $\mu(n)$, which is constant during non-adaptive playout, and varies with n, the playout queue backlog, during adaptive playout. When a packet arrives at the client with a non-contiguous sequence number, the client assumes that the missing packets have been lost. The client places retransmission requests for the missing packets into the retransmission request queue (RTX Req.), in sorted order according to sequence number. When this queue is non-empty, a packet is transmitted through the channel, back to the server every t_F/R. Retransmission requests are never lost. The rationale is that since the requests are much smaller than frames of media, they can be adequately protected. After a fixed propagation delay, the retransmission requests arrive at the server where the appropriate packet is fetched from the packet store and placed in the retransmission queue.

3.3 Playout Control

In the diagram shown in Fig. 4, packets are removed from the playout queue at the playout rate $\mu(n)$. In the non-adaptive case this rate is constant and given by $1/t_F$. With AMP, the client varies the playout speed according to some policy to control $\mu(n)$. For example, a simple playout policy for AMP-Live might be:

$$\mu(n) = \begin{cases} \frac{1}{s \cdot t_F} & n < N_{adapt} \\ \frac{1}{t_F} & n = N_{adapt} \\ \frac{1}{f \cdot t_F} & n > N_{adapt} \end{cases} \tag{2}$$

where $s \geq 1$ represents a decrease in playout speed, $f \leq 1$ is an increase in playout speed, n is the number of contiguous frames in the playout queue, and N_{adapt} is a threshold value. When there are fewer than N_{adapt} frames in the playout queue, each frame plays for $s \cdot t_F$ seconds. When the number in the queue exceeds N_{adapt}, each frame plays for

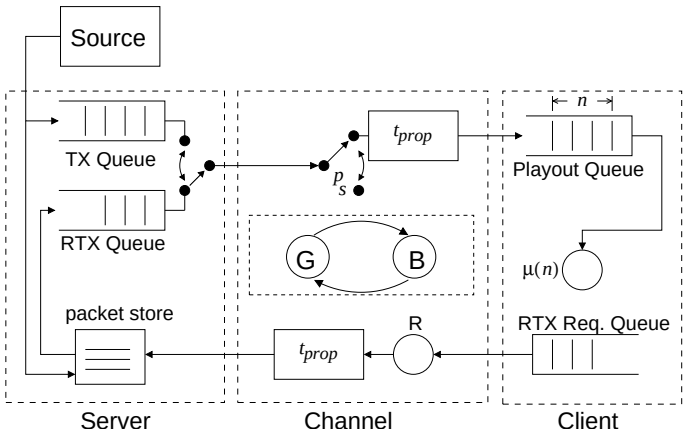

Figure 4: Our streaming video system model. The source generates frames and passes them to the server. The server queues packetized frames for transmission (TX) and frames which must be retransmitted (RTX). The server transmits a packet every t_F/R seconds with priority given to RTX. Packets are delayed and delivered or with probability p_s, $s \in \{Good, Bad\}$ they are lost. The client removes packets from the playout queue at a rate $\mu(n)$ which in the adaptive case is a function of the playout queue backlog. When packets arrive at the client with non-contiguous sequence numbers, a retransmission request is queued (RTX Req.). Retransmission requests are never lost in the channel.

$f \cdot t_F$ seconds. In general, we can define different functions $\mu(n)$ depending on our goals with respect to robustness, mean latency, and initial pre-roll delay (compare Section 2).

3.4 Markov Queuing Analysis

This section develops a Markov chain analysis of the system shown in Fig. 4. The goal is to enumerate the possible states of the system and then find probability distributions over these states. In our system, packets are lost randomly. Since each combination of packet losses places the system in a unique state, the number of states is exponential in the number of packets in the system. An explicit analysis of our system would be intractable.

For packet loss rates of $\leq$ 20%, typical of the Internet [13], however, after a few retransmission attempts the probability that a packet is received is nearly 1. Because the playout buffer in our system will allow a few seconds for retransmission attempts, and round trip times will be a few hundred milliseconds, each packet will have many retransmission opportunities. In our analysis we can therefore use the approximation that our system is a packet erasure channel with unlimited time for retransmission attempts. We can thus model packet loss as a reduction in throughput [14],[15]. Fig. 5 shows the simplified system that we assume in our analysis. Now, at discrete transmission

opportunities t_F/R seconds apart, a packet crosses the channel with probability $1 - p_s$, $s \in \{G, B\}$, or remains in the TX queue with probability p_s. The state dependent mean throughput thus becomes $R \cdot (1 - p_s)$.

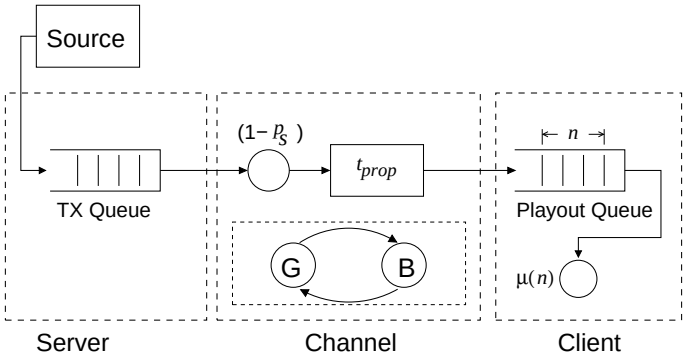

Figure 5: The simplified streaming video system model. To allow a tractable Markov chain analysis, we translate the packet loss rate into a reduction in transmission rate.

3.4.1 Analysis for Live Streams (AMP-Live)

For live streaming, we would like to find the probability of client buffer underflow as a function of mean latency. We proceed by first finding the probability distribution over the possible states of the system at the end of the pre-roll buffering period. With this distribution in hand, we can then find the probability distribution over the state space of the system during playout. From this we find the probability of underflow and the mean latency.

We can express the state of the system during the pre-roll period with the vector

$$S^{PR} = (m, n, c, k, \phi). \tag{3}$$

In (3), the component $m \in \{0, \ldots, M\}$ specifies the TX queue backlog, the component $n \in \{0, \ldots, N_{start}\}$ specifies the playout queue backlog, and $c \in \{G, B\}$ specifies the state of the channel. The client begins playout when N_{start} frames are in the playout buffer. M is the maximum size of the TX queue. We analyze the system in discrete time with K time slots per frame period. The state component $k \in \{0, \ldots, K-1\}$ encodes the system's slot position within the frame period. This component is necessary so that by observing the state of the system, one can know whether it is time for the source to deliver a frame to the server. Similarly, the state component ϕ is necessary because in our model packets are transmitted to client at t_F/R second intervals. With ϕ, by observing the state of the system, one can know if the current time slot corresponds to a transmission time.

We set R, the packet transmission rate, so that t_F/R, the packet transmission interval, is an integer number of time slots. For instance, if $K = 4$, we may set $R = 4/3$ so that transmissions occur every 3 time slots while frame deliveries from the source occur every 4 time slots. If the first transmission occurs at $k = 0$, subsequent transmission opportunities will occur every three slots, at $k = 3$, $k = 2$, $k = 1$, and finally again at $k = 0$, whereupon the cycle repeats. The state component $\phi \in \{0, \ldots, \Phi - 1\}$ specifies the system's position in this cycle relative to k, its position in the source delivery cycle. In our example, at any given time slot, the system can be in one of 12 possible positions with respect to the two cycles, 12 being the least common multiple of 3 and 4. In this case we need $\Phi = 3$ so that there are 12 (k,ϕ) pairs, enough to uniquely specify the 12 phase positions.

Because a (k, ϕ) pair uniquely specifies the position of the system in the packet transmission cycle, we can define a function $I_\phi(k, \phi)$ that indicates if a (k, ϕ) pair corresponds to a packet transmission time.

$$I_\phi(k, \phi) = \begin{cases} 1 & \text{if a packet is transmitted} \\ 0 & \text{otherwise} \end{cases} \tag{4}$$

We can also define a function $g_\phi(k, \phi)$ to determine the value that ϕ will take on in the next time step, given the values of k and ϕ in the current state.

Let

$$\pi^{PR}(t) = [\pi_0^{PR}(t), \pi_1^{PR}(t), \ldots, \pi^{PR}_{(M_{max}+1)(N_{start}+1)(2)(K)(\Phi)-1}] \tag{5}$$

be the probability distribution over the state space of the system at discrete time t. Let the function $u_{PR}(S^{PR}(t))$ map each possible vector (m, n, c, k, ϕ) to a unique index $i \in \{0, \ldots, (M_{max}+1)(N_{start}+1)(2)(K)(\Phi) - 1\}$. The elements of the vector in (5) are given by $\pi_i^{PR}(t) = Pr\{u_{PR}(S^{PR}(t)) = i\}$. We assume that at the beginning of the session, the channel state is distributed according to the stationary distribution of the two-state channel model, the first packet has arrived at the server ($m = 1$), and there are no packets at the client ($n = 0$). Thus, the initial probability distribution over the state space is

$$\pi_i^{PR}(0) = \begin{cases} \frac{T_G}{T_G+T_B} & \text{if } i = u_{PR}(1,\ 0,\ \text{Good},\ 0\ ,0) \\ \\ \frac{T_B}{T_G+T_B} & \text{if } i = u_{PR}(1,\ 0,\ \text{Bad},\ 0,\ 0) \\ \\ 0 & \text{otherwise} \end{cases} \tag{6}$$

Once streaming begins, the source delivers a packet to the server in states for which $k = 0$, incrementing m. From one time slot to the next, the k component transitions deterministically to $(k + 1)$ mod K and the ϕ component transitions deterministically to $g_\phi(k, \phi)$. For states indicated by $I_\phi(k, \phi)$ and for which the TX queue is not empty

$(m > 0)$, a packet crosses the channel with probability $1 - p_c$, incrementing n and and decrementing m. States for which the channel is good may transition to a state for which the channel is bad with probability $1/(K \cdot T_G)$ and vice versa.

Let the transition matrix, $P^{PR} \in \Re^{(M_{max}+1)(N_{start}+1)(2)(K)(\Phi)\times(M_{max}+1)(N_{start}+1)(2)(K)(\Phi)}$,

$$[P^{PR}]_{i,j} = Pr\{u_{PR}(S^{PR}(t+1)) = j \mid u_{PR}(S^{PR}(t)) = i\}, \tag{7}$$

encapsulate these transition probabilities. With the transition matrix, we can find the probability distribution over the state space at a given time:

$$\pi^{PR}(t+1) = \pi^{PR}(t)P^{PR}. \tag{8}$$

Because playout starts when N_{start} packets have been stored in the client buffer, and our goal is to find the distribution over the states of the system when playout starts, we specify that our transition matrix transfers any probability that flows into a state for which $n = N_{start}$ back into that state with probability one. Thus, if we start with the initial distribution $\pi^{PR}(0)$ given in (6) and propagate the distribution forward in time, whenever probability mass moves into a state for which $n = N_{start}$, it stays there, adding to the probability accumulated for that state's particular values of m, c, k, and ϕ. In time all the probability mass moves to states for which $n = N_{start}$ and the distribution ceases to change over time. At that point we have found the distribution over m, c, k, and ϕ when playout starts.

We find the distribution over m, c, k, and ϕ at playout start as

$$\pi^{PR}_{start} = \lim_{t\to\infty} \pi^{PR}(t) \tag{9}$$

which in practice we find by propagating the state forward according to (8) until

$$\sum_{i\in\mathcal{N}} \pi^{PR}_i(t_{final}) > 1 - \epsilon \tag{10}$$

where $\mathcal{N} = \{u_{PR}(m, n, c, k, \phi) \ \forall \ (m, n, c, k, \phi) : n = N_{start}\}$ is the set of states for which the client buffer backlog has reached $n = N_{start}$, and pre-roll has completed. The starting distribution is the distribution over $n = N_{start}$, m, c, k, and ϕ in $\pi^{PR}(t_{final})$ normalized by $\sum_{i\in\mathcal{N}} \pi^{PR}_i(t_{final})$.

Once we have the probability distribution over the states of the system when playout starts, π^{PR}_{start}, we can continue to find the distribution over the states of the system during playout (PL) and from these we can find the quantities we seek: mean latency and underflow probability. We can specify the state of the system during playout with the vector

$$S^{PL}(t) = (m, n, c, k, \phi, \psi) \tag{11}$$

where c and k and ϕ are the same as in the pre-roll case, $m \in \{0, 1, 2, \ldots, M\}$ is the server buffer backlog, and $n \in \{0, 1, 2, \ldots, N\}$ is the client buffer backlog. $\psi \in \{0 \ldots \Psi - 1\}$ encapsulates information regarding the current playout rate and the phase of packet departures from the client buffer relative to the frame period.

To understand the state component, ψ, consider the non-adaptive playout case. During non-adaptive playout, a frame is removed from the playout queue every frame period. If playout begins at $k = 0$, then every K time slots when $k = 0$, a frame is taken from the playout queue for playout. Now suppose that for states in which n is below a threshold N_{adapt}, the adaptation scheme plays each frame for $K + 1$ slots. In this case a frame will leave the playout queue when $k = 0$, $k = 1$, $k = 2$, and so on. Then, if n should regain the threshold value, frames will once again depart every K slots. Instead of frame departures occurring whenever $k = 0$, departures will now occur at whatever offset has accumulated during slowed playout.

In the state vector (m, n, c, k, ϕ, ψ), ψ allows us to keep track of these offsets as well as the current playout speed. We need ψ to determine if a frame departs the playout queue during a given time slot. Fig. 6 illustrates with an example. In the figure, $K = 4$, $s = 1.25$ and $f = .75$ (f and s were introduced in Sec. 3.3). Thus slow frames play for 5 slots, normal ones for 4 slots and fast ones for 3. If, for example, playout starts at $k = 0$ and $\psi = 0$ and the playout speed remains normal, ψ will remain at zero and every 4 slots a frame will arrive at the server and a frame will depart the client. Similarly, if playout starts with $\psi = 4$ and playout speed remains slow, a packet departs the client every 5 slots and ψ cycles between 4 and 8. In general, however, each time a packet is removed from the client for playout (the $n - -$'s in the figure), the playout rate is determined according whether n is above or below N_{adapt}. ψ changes accordingly. The large arrow in the figure illustrates how ψ may change during a frame departure at the client if the playout speed changes. In this case playout changes from normal to slow because we are supposing that $n < N_{adapt}$.

Given k and ψ, we can determine, therefore, whether a frame departs the client buffer in the current state, and given ψ, k, and n, we can determine the next value of ψ. Let the the function $I_\psi(k, \psi)$ indicate packet departure times,

$$I_\psi(k, \psi) = \begin{cases} 1 & \text{if a packet departs} \\ 0 & \text{otherwise} \end{cases} \tag{12}$$

Also, let $g_\psi(k, \psi, n)$ specify the deterministic transition that the ψ component of the state will take, given the values of ψ, n and k.

Let the probability distribution over the playout state space at time t be given by:

$$\pi^{PL}(t) = [\pi_0^{PL}(t), \pi_1^{PL}(t), \ldots, \pi^{PL}_{(M+1)(N+1)(2)(K)(\Phi)(\Psi)-1}] \tag{13}$$

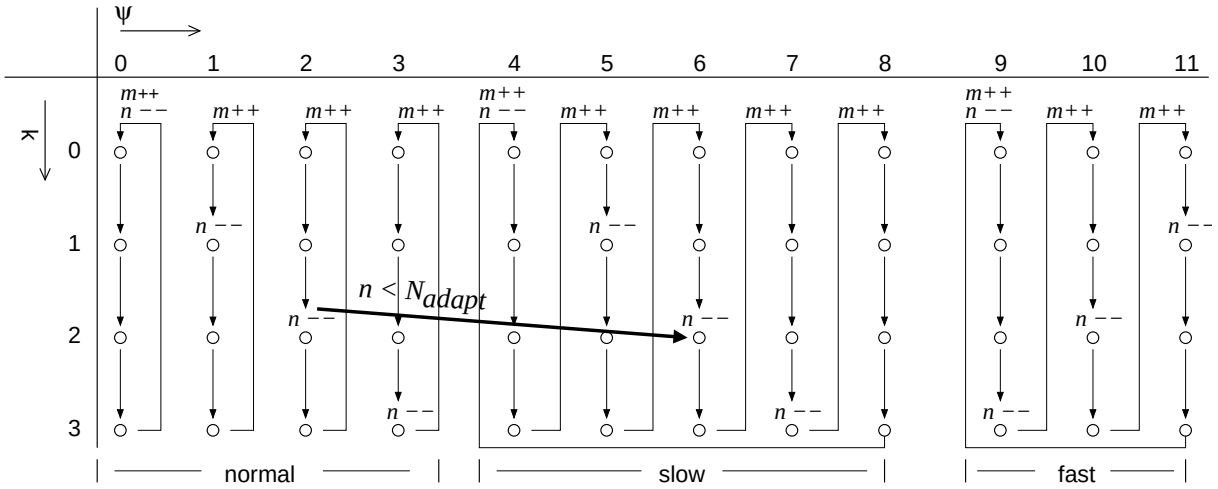

Figure 6: The small arrows indicate the cyclical succession of ψ values over time if playout remains either normal, slow, or fast. In this example $K = 4$ slots/frame, $s = 1.25$ (5 slots per frame departure during slow playout), and $f = 0.75$ (3 slots per frame departure during fast playout). $m++$ and $n--$ indicate frame arrivals at the server and frame departures at the client, respectively. The large arrow is an example of how ψ may change after a frame departure at the client ($n--$) if the playout speed changes. In this case playout changes from normal to slow because we are supposing that $n < N_{adapt}$.

where $\pi_i^{PL}(t) = Pr\{u_{PL}(S^{PL}(t)) = i\}$. $u_{PL}(S^{PL})$ maps state vectors to unique indices i. When playout begins, the distribution over the state space is as given in (9):

$$[\pi^{PL}(0)]_{u^{PL}(m,n,c,k,\phi,\psi)} = \begin{cases} [\pi^{PR}_{start}]_{u^{PR}(m,N_{start},c,k,\phi)} & \text{if } m \in \{0,1,\ldots,M_{max}\}, \\ & n = N_{start}, \psi = k \\ 0 & \text{otherwise} \end{cases} \tag{14}$$

Let the state transition matrix during playout be

$$[P^{PL}]_{i,j} = Pr\{u_{PL}(S^{PL}(t+1)) = j \mid u_{PL}(S^{PL}(t)) = i\}. \tag{15}$$

Some of the components of the state transition deterministically: k transitions to $(k+1)$ mod K, ϕ transitions according to $g_\phi(k,\phi)$, and ψ transitions according to $g_\psi(k,\psi,n)$. The transitions of the m and n components are not generally deterministic. While the influence of frame deliveries at the server when $k = 0$ and playout queue departures when $I_\psi(k,\psi) = 1$ is deterministic, m and n are affected non-deterministically for states, indicated by I_ϕ, in which a packet crosses the channel with probability $1 - p_c$. Channel state transitions are also non-deterministic.

Underflows at the client occur when $n = 0$, no packet arrives during the slot, and $I_\psi(k,\psi) = 1$, indicating that a packet is needed for playout. When an underflow occurs,

playout freezes and the session must begin again with the pre-roll process. Therefore, any mass that corresponds to an underflow is redistributed according to the starting distribution π^{PR}_{start} to simulate re-buffering and a resumption of playout.

Note that the random process $\{S^{PL}(t), S^{PL}(t+1), S^{PL}(t+2), \dots\}$ is cyclostationary with period K, because of the periodic, deterministic packet arrivals from the source. Using any of the K cyclostationary distributions, we can find the probability of underflow during a time slot as the probability that the state of the system has $n = 0$, no packet arrives from the server, and a packet is due to be removed from the client. Let any of the K cyclostationary distributions be π^{PL}. Then

$$\begin{aligned} Pr\{underflow\} &= \sum_{m\neq 0,n=0,c,k,\phi,\psi} I_\psi(k,\psi)(1 - I_\phi(k,\phi)(1-p_c))\pi^{PL}_{u_{PL}(m,n,c,k,\phi,\psi)} + \\ &\quad \sum_{m=0,n=0,c,\phi,\psi,k} I_\psi(k,\psi)\pi^{PL}_{u_{PL}(m,n,c,k,\phi,\psi)} \end{aligned} \tag{16}$$

Because packets enter the system deterministically at frame-period intervals and packets leave the system when they are played out, each packet in the system constitutes one frame period of latency. We can thus find the mean latency, L, in units of frame periods by finding the mean number of packets in the system during playout.

$$L = t_{prop} + \sum_{m,n,c,k,\phi,\psi} (m + n + k/K)\pi^{PL}_{u_{PL}(m,n,c,k,\phi,\psi)} \tag{17}$$

3.5 Analysis for Stored Streams (AMP-Initial, and AMP-Robust)

Our analysis proceeds differently when the source of the media stream is not live but has been previously stored. First, since the entire program is available from the start, there is no reason to keep track of the state of the server buffer. A packet is always available. Similarly, viewing latency is not a meaningful quantity. The quantities of interest here are pre-roll delay and the underflow probability during playout.

We can find the probability distribution over the states of the system when playout begins in same manner as in the live program case. In the process we can also find the mean pre-roll time. We begin by defining our state space. Because we do not need to track the backlog of the server's TX queue, the state of the system during pre-roll can now be specified with a vector $S^{PR} = (n, c, k, \phi)$, where $n \in \{n : \ 0 \leq n \leq N_{start}\}$ is the backlog of client's playout queue, $c \in$ {Good, Bad} is the state of the channel, $k \in \{0,$ $\dots,$ $K-1\}$ is the time slot position of the state within a frame period, and ϕ specifies the offset of packet transmission opportunities relative to frame periods as described earlier. Let the probability distribution over the state space of the system at time slot t be

$$\pi^{PR}(t) = [\pi^{PR}_0(t), \pi^{PR}_1(t), \dots, \pi^{PR}_{2(N_{start}+1)-1}] \tag{18}$$

where $\pi_i^{PR}(t) = Pr\{u_{PR}(S^{PR}(t)) = i\}$. u_{PR} maps state vectors to unique indices.

Just as in the live program case, we assume that at the beginning of a session the channel state is distributed according to the stationary distribution of the two-state channel model and that there are no packets queued at the client ($n = 0$). Thus the starting probability distribution over the state space is

$$\pi_i^{PR}(0) = \begin{cases} \frac{T_G}{T_G+T_B} & \text{if } i = u_{PR}(0, \text{Good}, 0, 0) \\ \frac{T_B}{T_G+T_B} & \text{if } i = u_{PR}(0, \text{Bad}, 0, 0) \\ 0 & \text{otherwise} \end{cases} \tag{19}$$

Once streaming begins, the state of the system can transition in the same way as in the pre-roll case for live streams, except that in the stored program case, packet transmission opportunities are no longer contingent upon whether a packet is waiting in the TX queue. When $I_\phi(k, \phi) = 1$, a packet crosses the channel with probability $1 - p_c$, incrementing n. Again, we build a transition matrix P^{PR} which specifies the transition probability between each pair of states, so that $\pi^{PR}(t+1) = P^{PR}\pi^{PR}(t)$.

As in the live streaming pre-roll case, we build the transition matrix so that any mass in states for which $n = N_{start}$ transitions back to that state with probability 1. By propagating the state forward in time, we can find the distribution over c, k, and ϕ when playout starts (see (9), (10)). By also observing how much mass flows into the $n = N_{start}$ states at each time t, we can find the mean pre-roll time,

$$T_{PR} = \sum_{t=1}^{\infty} \left(\sum_{i \in \mathcal{N}} \pi_i^{PR}(t) - \sum_{i \in \mathcal{N}} \pi_i^{PR}(t-1) \right) \cdot t, \tag{20}$$

where $\mathcal{N} = \{u_{PR}(n, c, k, \phi) \ \forall \ (n, c, k, \phi) : \ n = N_{start}\}$ is the set of states for which $n = N_{start}$ packets have arrived at the client and playout can begin.

Once we have the distribution over the states of the system when playout starts, we can proceed to find the probability that an underflow will occur by a given time during the playout of a stored program. Let the state of the system during playout be given by $S^{PL} = (n, c, k, \phi, \psi)$ where $n \in \{0, \ldots, N\}$, $c \in \{\text{Good, Bad}\}$, $k \in \{0, \ldots, K-1\}$, $\phi \in \{0, \ldots, \Phi - 1\}$ and $\psi \in \{0, \ldots, \Psi - 1\}$. The state components are the same as in the playout of a live stream, but the component m is eliminated because it is no longer necessary to track the backlog of the server queue. A packet is always waiting.

As in the live playout case, let π^{PL} be the distribution over the states of the system during playout. Let $u_{PL}(S^{PL})$ map state vectors to unique indices, i. The probability

distribution over the states of the system at playout start is given by

$$[\pi^{PL}(0)]_{u^{PL}(n,c,k,\phi,\psi)} = \begin{cases} [\pi^{PR}_{start}]_{u^{PR}(N_{start},c,k,\phi)} & \text{if } n = N_{start}, \psi = k \\ \\ 0 & \text{otherwise} \end{cases} \tag{21}$$

We again define a transition matrix, P^{PL} to propagate the distribution forward in time. The transition matrix is similar to that of the live playout case, except now packet transmission opportunities are no longer contingent on whether a packet is available in the TX queue. Another difference concerns the handling of underflows. Because the goal is to determine the probability that a buffer underflow will occur by a given time during playout, an extra state is added to the state space for underflows. In our transition matrix, any mass that corresponds to an underflow, which occurs in states for which $n = 0$, $I_\psi(k, \psi, n) = 1$, and no packet crosses the channel, transitions to this underflow state. We specify that the underflow state transitions to itself with probability 1. Thus, any mass that enters the underflow state remains there. As the transition matrix propagates the distribution forward in time, the mass that accumulates in the underflow state corresponds to the probability that an underflow has occurred by a given time.

4 Experimental Results

In this section we present the results of Markov chain analysis and simulation experiments for the three AMP schemes that we have considered: AMP-Live, AMP-Robust and AMP-Initial. The results show that our AMP schemes do in fact enhance the buffering performance of streaming media systems. They also validate our Markov chain analysis by showing close agreement between analytical and experimental outcomes.

As described in Section 3.4, in our analysis we use a simplified treatment of packet loss and ARQ (see Fig. 5) that is based on the assumption that the round-trip time is short enough to allow several retransmission attempts before packets are due for playout. In our simulation experiments, however, we explicitly model packet losses and retransmissions (see Fig. 4). The model includes a simple ARQ protocol in which the client requests retransmissions when packets arrive out of order as described in Section 3.2. Note that in this model, round-trip times are bounded below. For example, suppose a packet is lost. The client learns of the loss, at the earliest, t_F/R seconds later when the next packet arrives. If the client immediately queues a retransmission request, the request will arrive at the server no earlier than $2 \cdot (t_F/R) + t_{prop}$ seconds after the original packet loss. When the server retransmits the lost packet, the retransmission may arrive at the client $3 \cdot (t_F/R)$ $+ 2 \cdot t_{prop}$ seconds after the packet loss, at the earliest. For $t_F = 100$ ms, $R = 4/3$, and

t_{prop} = 10 ms, a lower bound on the round-trip time is 245 ms. If multiple packets are lost, queuing delays will increase round trip times.

4.1 Results for AMP-Live

AMP-Live is designed to allow reduced viewing latencies for a given probability of buffer underflow during the streaming of a live program. In Section 3.4.1 we presented a Markov chain model to analyze this scheme. Fig. 7 plots results for a channel characterized by $t_{prop} = 10$ ms, $R = 4/3$, $T_G = 28.5$ s, $T_B = 1.5$ s, $p_G = .01$, and $p_B = 1$, and for a playout speed control policy as given in (2) with frame period scaling factors s and f as indicated for each trace. The figure plots the Mean Time Between Buffer Underflows (MTBBU) in minutes, as a function of mean latency in seconds.

In our Markov analysis the mean viewing latency is given by (17). MTBBU can be found as a function of the per-time-slot underflow probability given in (16):

$$\text{MTBBU} = \left(\frac{1}{Pr\{\text{underflow}\}}\right)\left(\frac{t_F}{K \cdot 60}\right) \tag{22}$$

The expression follows if each discrete time slot is seen as an independent Bernoulli trial with the outcome being either underflow, or no underflow. The waiting time until an underflow occurs is therefore distributed geometrically over the succession of time slots. The expression in (22) is the mean of this geometric random variable expressed in minutes. In simulation experiments we find the mean latency by averaging how long it takes for frames to be played out after they are delivered by the source. Likewise, we find MTBBU by simply averaging playout durations between buffer underflows. In Fig. 7, there are three traces. The first, with $s = f = 1$, is the non-adaptive case where the viewing latency is a fixed value given by the pre-roll delay. In the second trace with $s = 1.25$ and $f = .75$, frame periods can be stretched or compressed by 25%. In the third trace frame periods can be stretched or compressed by 50 %. We see that for the given channel and adaptation rates, AMP-Live can reduce the mean latency by 25 to 35 percent compared to the non-adaptive case, for a fixed MTBBU. Conversely, for a given latency, the MTBBU can be improved by a factor of 2 or more, with only 25 % playout adaptation. We see good agreement between simulation and analytical results.

4.2 Results for AMP-Robust

For stored streams, the performance-defining quantities are pre-roll delay and underflow probability. AMP-Robust is designed to reduce the buffer underflow probability given an initial buffer size and associated pre-roll delay. In Section 3.5 we presented a Markov chain model to analyze this scheme with respect to these quantities.

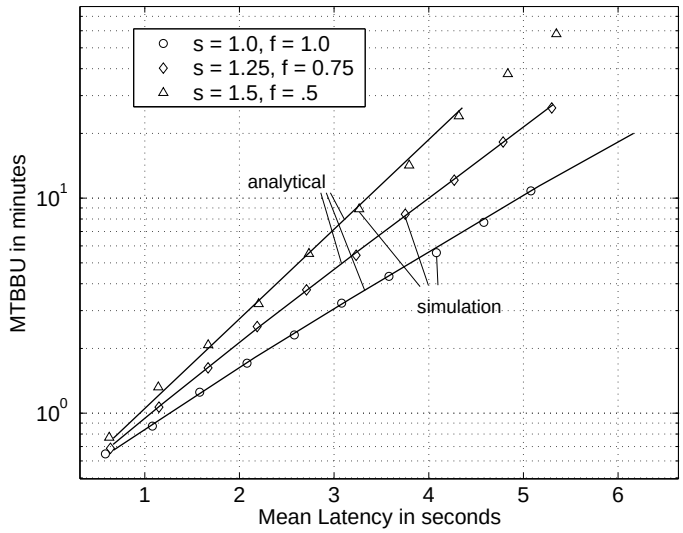

Figure 7: Mean Time Between Buffer Underflows (MTBBU) vs. Mean Latency for a channel C characterized by $t_{prop} = 20$ ms, $R = 4/3$, $T_G = 28.5$ s, $T_B = 1.5$ s, $p_G = .01$, and $p_B = 1$.

In Fig. 8 we plot the pre-roll delay that is necessary such that a program will play for a given duration with underflow probability less than 1%. Playout duration is plotted along the independent axis, and the required pre-roll time for 99% reliability is plotted on the dependent axis. The results are for a channel characterized by $t_{prop} = 20$ ms, $R = 4/3$, $T_G = 18$ s, $T_B = 2$ s, $p_G = .01$, and $p_B = 1$, and playout speed control policy as given in (2), with frame period scaling factors $f = 1$, and s as indicated for each trace. Playout begins when the client buffer backlog reaches N_{start} packets, and playout is slowed when the backlog falls below $N_{adapt} = N_{start}$. In this plot, we constrain the maximum number of packets that can be stored in the playout queue to $N = 200$, or 20 seconds of media data.

In this plot we see that AMP-Robust allows playout durations that are orders magnitude longer for a given pre-roll time and for $Pr\{\text{underflow}\} \leq 0.01$ than the playout durations allowed by the non-adaptive case. Conversely, for a given reliable playout duration, AMP-Robust can allow pre-roll times to be reduced by a factor of two or more. Because the maximum number of packets in the client buffer is constrained to $N = 200$, and because the probability of a long bad channel period increases the longer that playout continues, the required pre-roll time turns sharply upward for playout durations beyond several hundred seconds. In Fig. 9 of Sec. 4.3 we will see that using a larger buffer can eliminate the upward turns.

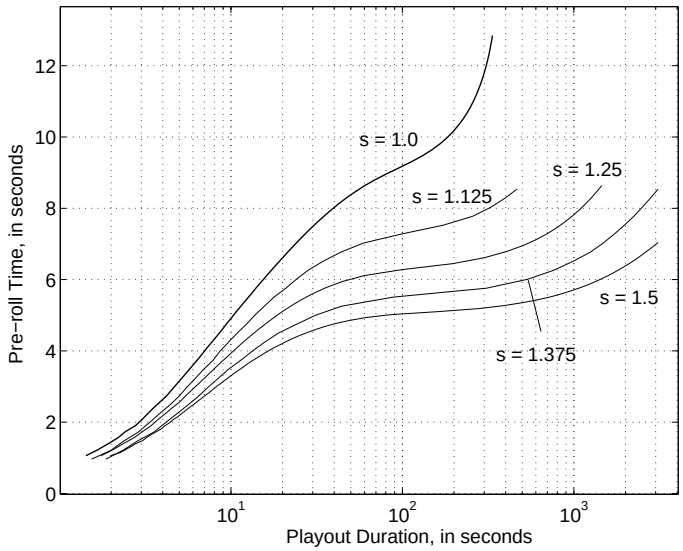

Figure 8: A plot of the required pre-roll time as a function of program length so that $Pr\{\text{underflow}\} \leq 0.01$ for AMP-Robust. In this plot $T_G = 18$ s, $T_B = 2$ s We see that AMP-Robust allows shorter pre-roll times for a given playout rate than the non-adaptive ($s = 1.0$) case.

4.3 Results for AMP-Initial

In this section we see that AMP-Initial, by allowing N_{start} to be less than N_{adapt}, offers shorter pre-roll times for a given underflow probability and program length compared to AMP-Robust. The Markov chain analysis is the same as in the previous section, but now $N_{start} < N_{adapt}$.

Compare Fig. 9 to Fig. 8. The channel parameters remain the same: $t_{prop} = 20$ ms, $R = 4/3$, $T_G = 18$ s, $T_B = 2$ s, $p_G = .01$, and $p_B = 1$. To achieve the further reductions in pre-roll time for a given playout duration as shown in Fig. 10, however, we allow playout to start when the threshold N_{start} is achieved, but fix N_{adapt}, the threshold for playout rate adaptation, at 100 frames. In other words, for these traces playout is slowed whenever fewer than 10 seconds of media remain in the buffer.

When the maximum size of the client buffer is allowed to increase to $N = 400$, or 40 seconds of media, the results are as shown in the Fig. 10. In this overprovisioned scenario, where the mean channel capacity is higher than the rate of the source, the larger buffer can provide the desired reliability, regardless of program playing time, provided enough data is buffered before playout begins. In this scenario, as in the $N = 200$ case, AMP again allows pre-roll times to be reduced by factors of 2 or more. Unlike the the $N = 200$ case, however, the required pre-roll time for a reliable playout duration does not suddenly increase as programs grow longer. Compare Figs. 10 and 9.

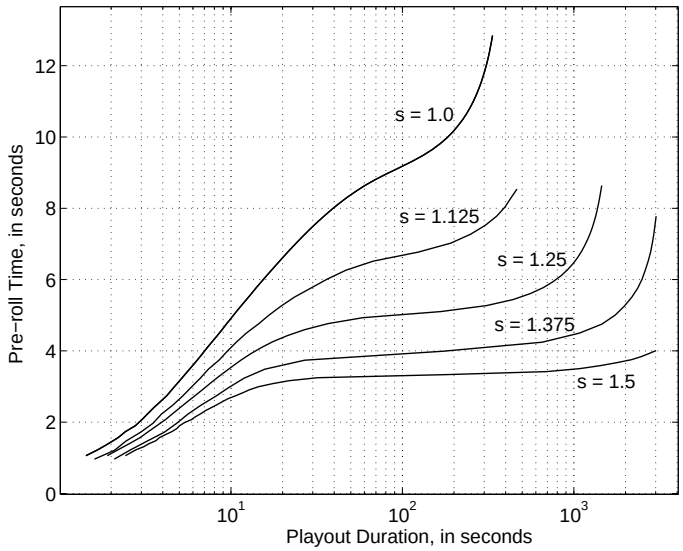

Figure 9: As in Fig. 10, a plot of the mean program length so that $Pr\{\text{underflow}\} \leq 0.01$ for AMP-Initial. Because $N_{adapt} = 100$ for all N_{start}, we see shorter pre-roll times for a given program length as compared to AMP-Robust in Fig. 8.

5 Conclusions

In this paper we propose and analyze adaptive media playout for low latency video streaming over lossy packet networks. We present a Markov chain analysis of AMP video streaming and study the delay-underflow trade-off for three cases. In the first case we consider AMP for low initial delay which enables fast switching between programs. The second case considers adaptive media playout for interactive and time critical streaming sessions. In the third case we show how AMP can be used to significantly decrease the buffer underflow probability for streaming of stored programs. For all three AMP schemes, we show that significant performance improvements can be obtained in comparison to non-adaptive media playout.

Acknowledgments

The authors would like to thank Dr. Niko Färber and Prof. Nicholas Bambos for helpful discussions.

References

[1] Microsoft, Windows Media Player (http://www.windowsmedia.com)

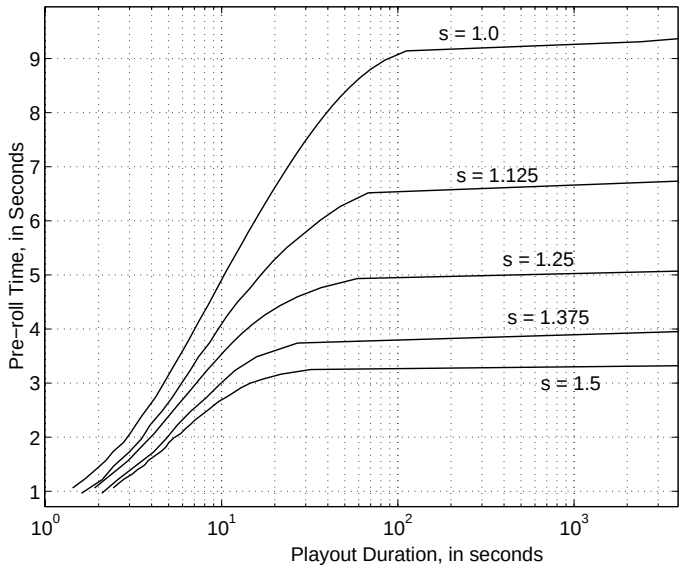

Figure 10: A plot of the mean required pre-roll time as a function of program length so that $Pr\{\text{underflow}\} \leq 0.01$ for AMP-Initial. Here, the maximum size of the playout buffer is $N = 400$ as compared to $N = 200$ in Fig. 10.

[2] Real Networks, RealPlayer (http://www.realnetworks.com)

[3] W. Verhelst and M. Roelands, "An overlap-add technique based on waveform similarity (WSOLA) for high quality time-scale modification of speech," *Proc. ICASSP '93*, pp. 554-557, April 1993.

[4] Y.J. Liang, N. Färber, and B. Girod, "Adaptive Playout Scheduling Using Time-scale Modification in Packet Voice Communication," *Proc. ICASSP '01*, Salt Lake City, May 2001.

[5] M. Podolsky, S. McCanne, and M. Vetterli, "Soft ARQ for layered streaming media," *Tech. Rep. UCB/CSD-98-1024, University of California, Computer Science Division, Berkeley, CA,* Nov. 1998.

[6] A. Stenger, K. Ben Younes, R. Reng, and B. Girod, "A new error concealment technique for audio transmission with packet loss," *Proc. EUSIPCO '96*, Trieste, Italy, Sept. 1996.

[7] Y.J. Liang, E.G. Steinbach, and B. Girod, "Real-time Voice Communication over the Internet Using Packet Path Diversity," *Proc. ACM Multimedia 2001*, Ottawa, Canada, Sept./Oct. 2001.

[8] M.C. Yuang, S.T. Liang, and Y.G. Chen, "Dynamic Video Playout Smoothing Method for Multimedia Applications," *Multimedia Tools and Applications*, vol. 6, pp. 47-59, 1998.

[9] E.G. Steinbach, N. Färber, and B. Girod, "Adaptive Playout for Low Latency Video Streaming," *Proc. International Conference on Image Processing (ICIP-01)*, Thessaloniki, Greece, Oct. 2001.

[10] M. Kalman, E. Steinbach, and B. Girod, "Adaptive playout for adaptive media streaming," *Proc. IEEE International Symposium on Circuits and Systems, (ISCAS-2002)*, Scottsdale, AZ, May 2002.

[11] L.N. Kanal and A.R.K. Sastry, "Models for channels with memory and their applications to error control," *Proc. of the IEEE*, vol. 66, no. 7, pp. 724-744, July 1978.

[12] A. Leon Garcia, *Probility and Random Processes for Electrical Engineering*, Addison Wesley, 1994

[13] G.J. Conklin, G.S. Greenbaum, K.O. Lillevold, A.F. Lippman, and Y.A. Reznik, "Video coding for streaming media delivery on the Internet," *IEEE Transactions on Circuits and Systems for Video Technology*, pp. 269 - 281,vol. 11, no. 3, March 2001.

[14] S. Lin, D.J. Costello Jr., "Automatic repeat request error control schemes", *IEEE Commun. Mag.*, vol. 22, no. 12, Dec. 1984.

[15] S.B. Wicker, "Error Control Systems for Digital Communication and Storage," *Englewood Cliffs: Prentice Hall*, 1995.