Adaptive Rate Control for Streaming Stored Fine-Grained Scalable Video

Philippe de Cuetos
Institut EURECOM
2229, route des Crêtes
06904 Sophia Antipolis, France
philippe.de-cuetos@eurecom.fr

Keith W. Ross
Institut EURECOM
2229, route des Crêtes
06904 Sophia Antipolis, France
keith.ross@eurecom.fr

ABSTRACT

In this paper we investigate adaptive streaming of stored fine-grained scalable video over a TCP-friendly connection. The goal is to develop low-complexity yet high-performing schemes that adequately adapt to the short- and long-term variations in available bandwidth. We first present a novel framework for low-complexity streaming of fine-grained scalable video over a TCP-friendly connection. In the context of this scheme, and under the assumption of complete knowledge of bandwidth evolution, we derive an optimal policy for a criterion that involves both image quality and quality variability during playback. Based on this ideal optimal policy, we develop a real-time heuristic to stream fine-grained scalable video over the Internet, and we study its performance using real Internet traces. We find that our heuristic policy performs almost as well as the ideal optimal policy for a wide-range of bandwidth scenarios and when run over ordinary TCP the policy is essentially as good as when running the policy over popular TCP-friendly algorithms.

Categories and Subject Descriptors

H.4.m [**Information Systems**]: Miscellaneous

General Terms

Algorithms, Theory, Experimentation

Keywords

Internet video streaming, Fine-Grained Scalability, network-adaptive applications

1. INTRODUCTION

It is now commonly accepted that the transmission of multimedia streams over the Internet should be made fair with TCP traffic. From [5], a flow is said *TCP-friendly* if its arrival rate does not exceed the arrival rate of a conformant TCP connection in the same network circumstances. A number of TCP-friendly congestion control algorithms have been proposed recently, such as RAP [12], TFRC [6], SQRT [3]. As does TCP, TCP-friendly algorithms react to indications of network congestion by reducing the transmission rate. The transmission rate of TCP-friendly algorithms is typically smoother than that of TCP [17]. Nevertheless, because network congestion occurs at multiple time scales [10], the bandwidth available to TCP-friendly streams typically fluctuates over several time scales [17, 6].

In this paper we investigate adaptive streaming of stored video over a TCP-friendly "connection". The goal is to develop low-complexity yet high-performing schemes that adequately adapt to the short- and long-term variations in available bandwidth.

Small time-scale bandwidth fluctuations (on the order of a few RTTs) can be accommodated by maintaining a small play-back delay of a few seconds from the moment when the video frame is sent from the server to the time the frame is decoded at the client. Buffers at the client store the video frames before they are decoded. Maintaining a small playback delay also provides the server the opportunity to retransmit lost video packets before their decoding deadline [8, 13].

Longer time-scale bandwidth fluctuations, on the order of a few seconds to tens of seconds, can be addressed by using multiple versions of the same video [4] or a layered-encoded video [14]. When using a layered-encoded video, the application needs adaptive control policies to decide which layers should be streamed at all times [11, 9, 15] in order to maximize the overall quality of the video rendered to the user.

In this paper, we study the streaming of 2-layer Fine-Grained Scalable (FGS) videos. Like the other types of video scalability (SNR, spatial or temporal [2]), FGS encodes the video into a base and one or several enhancement layers. However, with FGS, any number of bits of the enhancement layer can be suppressed at the server before transmission, and the decoder can use all of the truncated bitstream to increase video quality at the client [7]. Recently, bit-plane based FGS-coding has been added to the MPEG-4 standard [1]. However our framework is intended to be valid for any 2-layer FGS-encoded video, and not only MPEG-4 FGS videos.

The first contribution of this paper is a new framework for streaming stored FGS video over TCP-friendly connec-

NOSSDAV'02, May 12-14, 2002, Miami, Florida, USA.

tion. This new framework calls for client buffers, prefetching throughout playback, and synchronous transmission across base and enhancement layers. Policies that operate within this framework require minimum real-time processing, and are thus suitable for servers that stream a large number of simultaneous unicast streams.

Our second contribution is to formulate and solve an optimal streaming problem. Our optimization criterion is based on simplistic, but easily tractable metrics, which account for the total video display quality as well as variability in display quality. We develop a theory for determining an optimal streaming policy under ideal knowledge of the evolution of the future bandwidth. The optimal ideal policy provides bounds on the performance of real-time policies and also suggests a real-time heuristic policy. Simulations from real Internet traces show that the heuristic performs almost as well as the ideal optimal policy for a wide-range of scenarios.

We also compare streaming stored-FGS video over an ordinary TCP connection to streaming over a TCP-friendly connection. The performance of current TCP-friendly congestion control algorithms is usually assessed in terms of their fairness with TCP, responsiveness to changes in network congestion and smoothness of throughput [17]. Because popular TCP-friendly algorithms have an available bandwidth that is typically smoother than TCP available bandwidth, one expects TCP-friendly algorithms to perform better, particularly for reducing quality fluctuations. However, our experiments show that video quality fluctuations are in the same range for both TCP and TCP-friendly algorithms.

This paper is organized as follows. In Section 2 we present our framework for streaming FGS-encoded video over a single TCP-friendly connection. Then, we formulate the optimization problem and define the performance metrics considered in this paper. In Section 4 we develop a theory for optimal streaming under ideal knowledge of future bandwidth evolution. In the next section, we present our real-time rate adaptation heuristic, which is inspired from the optimization theory in Section 4. We use simulations with Internet traces and simulations with an end-to-end MPEG-4 FGS streaming platform to study the performance of the heuristic. Finally, in Section 6 we compare the performance of our heuristic when run on top of TCP to when run on top of popular TCP-friendly algorithms.

2. FRAMEWORK

Let $X(t)$ denote the available bandwidth at time t. By permitting prefetching into client buffers, our server streams the video at the maximum rate $X(t)$ at each instant t. We suppose that the connection is made reliable, e.g., by using retransmissions [8], so that losses may only occur due to missed deadlines. Selective retransmissions are possible because of client buffering. Buffering will also allow us to neglect in our analysis the transmission delay between the server and the client.

The stored video is encoded into two layers, the Base Layer (BL) and the fine-grained Enhancement Layer (EL). For simplicity, we assume that the video is CBR-encoded. We denote the encoding rates of the base and enhancement layers by r_b and r_e, respectively. The length of the video is denoted by T sec.

Figure 1 shows the architecture of the server. The server

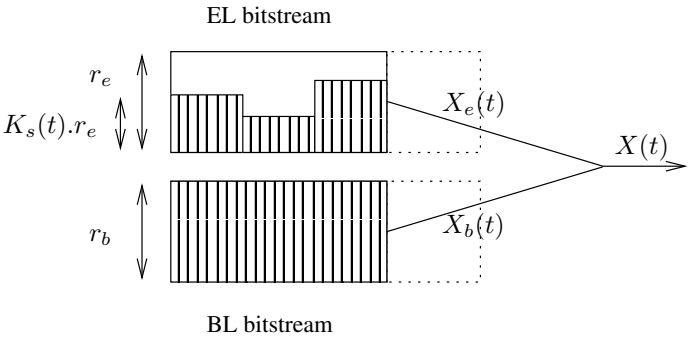

Figure 1: Server

stores the video as two separate files: one file contains the bitstream pertaining to the base layer and the other file contains the bitstream pertaining to the enhancement layer. Because the server transmits at maximum rate $X(t)$, at any given instant of time the server may be sending prefetched frames to the client that are minutes into the future. To reduce the server complexity, we require that the server always sends the base and enhancement instances of the same frame together, thus acting synchronously. This implies that at the client the number of prefetched BL frames is always equal to the number of prefetched EL frames.

For each transmitted frame, the server sends the entire BL of the frame and a portion of the EL of the frame. Because of the fine-grained property of the enhancement layer, the server can truncate the EL portion of the frame at any level. Thus, at each instant, the server must decide how much enhancement layer data to send. Specifically, in our design, time is broken up in slots $[t_k, t_{k+1})$, where $t_0 = 0$ and $t_n = T$. At the beginning of a slot, the server determines the enhancement layer level, denoted by $K_s(k) \cdot r_e$, that it streams for the duration of the slot. Thus, as shown in Figure 1, all frames sent during slot $[t_k, t_{k+1})$ will include the entire base layer and a same fraction of the enhancement layer, $K_s(k) \in [0, 1]$. The frames sent during the slot may be prefetched frames for display seconds or even minutes into the future. The length of a slot can be chosen so that the slot is composed of one or more complete video scenes. This would keep constant the fraction of enhancement layer that is transmitted for each video scene, avoiding changes in perceptual image quality within the same video scene. The length of a time slot should be on the order of seconds, which will also help to maintain low server complexity. For simplicity, we will assume that the slot length is equal to a constant C, so that we can write $t_k = k \cdot C$, for $k = 0, \ldots, n$.

The rate at which the server transmits frames into the network depends on the available bandwidth during the slot, i.e $X(t)$ for $t_k \leq t < t_{k+1}$. Because we are requiring that the base and enhancement layer components of a frame be sent at the same time, the available bandwidth dedicated to the base layer and to the enhancement layer at time $t \in [t_k, t_{k+1})$ is respectively:

$$X_b(t) = \frac{r_b}{r_s(k)} X(t) \quad \text{and} \quad X_e(t) = \frac{K_s(k) \cdot r_e}{r_s(k)} X(t) \qquad (1)$$

where $r_s(k) = r_b + K_s(k) \cdot r_e$ is the total coding rate of the video being streamed between times t_k and t_{k+1}. By

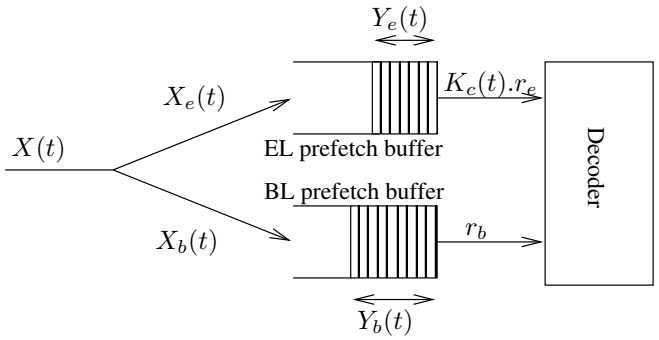

Figure 2: Client

$X(t)$	Available bandwidth at time t
r_b	Coding rate of the base layer
r_e	Coding rate of the fine-grained enhancement layer
T	Length of the video in sec.
$K_s(k)$	Proportion of the EL sent by the server at time t_k
$K_c(t)$	Proportion of the EL decoded by the client at time t
$r_s(k)$	Total coding rate of the video sent between t_k and t_{k+1}
$r_c(t)$	Total coding rate of the video decoded at time t
C	Length of a server slot in sec.
n	Number of time slots
t_{end}	Ending time of the streaming
$Y_b(t)$	Content of the client BL prefetch buffer at time t
$Y_e(t)$	Content of the client EL prefetch buffer at time t
Δ_0	Initial build up delay
$\Delta(t)$	Playback delay at time t
Δ_k	Playback delay at time t_k

Table 1: Notations

extension, the total coding rate of the video being streamed at time t will be denoted by $r_s(t) = r_b + K_s(t) \cdot r_e$.

As shown in Figure 2, the client stores temporarily the data coming from the network in base and enhancement layer prefetch buffers. Let $Y_b(t)$ and $Y_e(t)$ denote the amount of data stored in the prefetch buffers at time t. At time t, the decoder drains both buffers at rates r_b and $K_c(t) \cdot r_e$, where $K_c(t) \in [0, 1]$ is the fraction of enhancement layer available at the client for the frame which is scheduled to be decoded at time t. The encoding rate of the video being displayed at the client at time t can be expressed as $r_c(t) = r_b + K_c(t) \cdot r_e$.

The server initially sends Δ_0 seconds time-worth of full quality video to the client. This phase is called the initial build-up and Δ_0 is the initial playback delay between the server and the client. At time $t = 0$, the client starts to decode the data from its prefetching buffers and to render the video, while the server streams the rest of the video frames. To simplify our analysis, we suppose that, once the playback starts at the client, the user is going to watch the video until the last frame without performing any VCR command. Let t_{end} denote the time when the server stops sending video frames. We will have $t_{end} < T$ if the server has sent the last video frame before video is fully rendered; otherwise, we will have $t_{end} = T$.

We denote $\Delta(t)$ for the number of seconds of prefetched video contained in the prefetch buffers at time t. Since the server acts synchronously and each layer is streamed in proportion to its encoding rate, the base and enhancement prefetch buffers will always have the same value of $\Delta(t)$ for each time t. Because the server changes the encoding rate of the enhancement layer at each time t_k, the enhancement layer prefetch buffer contains parts of the video encoded at different bit-rates. However, because the base layer is never truncated, we can write at each time t, $\Delta(t) = \frac{Y_b(t)}{r_b}$. Since, at time t, the server streams a video frame which is going to be decoded by the client $\Delta(t)$ seconds into the future, the total coding rate of the video which is decoded at time t can be expressed as:

$$r_c(t + \Delta(t)) = r_s(t), \text{ or } r_c(t) = r_s(t - \Delta(t)). \quad (2)$$

To simplify notation, we denote $\Delta_k = \Delta(t_k)$. We assume that, at each time t_k, the server knows the value of Δ_k (e.g. through periodical receiver reports).

Because we suppose that the connection is made reliable, losses at the client only occur when data arrives at the client after their decoding deadline. Such data will not be decoded. Assuming a playback delay of $\Delta_k > 0$ at time t_k, losses may only start to happen at time $t_k < t < t_{k+1}$ when $\Delta(t) = 0$ and when $X(t) < r_s(t)$, i.e., when the client buffers are empty and the available bandwidth is not high enough to feed the decoder at the current video encoding rate. Since the server acts synchronously for both layers and each layer is sent in proportion to its coding-rate, losses can only happen for both layers at the same frame time. When there is loss at time $t \in (t_k, t_{k+1})$, we suppose that the server is not able to react: it will keep streaming the current part of the video, even if it will not meet its decoding deadline. Meanwhile, the client keeps incrementing its decoding timestamp and waits for the part of the video that has the new current decoding deadline. A negative $\Delta(t)$ indicates that the part of video arriving from the server has an earlier decoding deadline than the part of the video that the client is waiting for. Therefore, there will be loss of data whenever $\Delta(t) < 0$. Table 1 summarizes our notations.

3. PROBLEM FORMULATION

A *transmission policy*, denoted by $\mathbf{r}_s = (r_s(0), \ldots, r_s(n-1))$, is a set of successive video encoding rates, each of which is chosen by the server at the beginning of a time slot $[t_k, t_{k+1})$, for video sequence k. In order to provide the user with the best perceived video quality, the transmission policy should minimize a measure of total distortion [18], such as the commonly employed MSE (Mean Squared Error), and also minimize variations of distortion between successive images. Each video sequence may have different rate-distortion characteristics, and for a given video sequence k, the distortion typically does not vary linearly with the sequence chosen coding rate $r_s(k)$.

In this study, we restrict to transmission policies that (i) ensure a minimum of quality, by ensuring the decoding of the BL data without loss, (ii) maximize the bandwidth efficiency, i.e. the total number of bits decoded given the available bandwidth, which gives a good indication of the total video quality, and (iii) minimize the variations of the rendered coding rate between successive video sequences, which gives an indication of the variations in distortion.

Although these metrics are simple and independent of the rate-distortion characteristics of a particular video, our analysis remains useful for deriving real-time heuristics, and for comparing different transport protocols.

3.1 Bandwidth Efficiency

We define the bandwidth efficiency E as the ratio between the average number of bits decoded at the client by seconds over the total rate of the video:

$$E = \frac{\bar{r_c}}{r_b + r_e} \text{ where } \bar{r_c} = \frac{\text{total number of bits decoded}}{T} \tag{3}$$

Observing that the video data at the receiver may come either from the initial build up or from the streaming, we can write :

$$E = \frac{\Delta_0(r_b + r_e) + \int_0^{t_{end}} X(t)dt - (\text{nb of bits lost})}{T \cdot (r_b + r_e)} \tag{4}$$

Recall that the "number of bits lost" is the number of bits sent by the server that do not make their deadline at the client.

3.2 Coding Rate Variability

Previous studies in [9] have designed various measures to account for the rate variability in the case of layered-video with a small number of layers. Here, we propose a measure for the case of FGS encoding, for which the displayed video encoding-rate $r_c(t)$ can take continuous values between r_b and $r_b + r_e$. Since from Equation 2, $r_c(t) = r_s(t - \Delta(t))$, and since $r_s(t) = r_s(k)$ over all intervals $[t_k, t_{k+1})$, differences in consecutive values for $r_c(t)$ at the client are accounted by differences in consecutive values for $r_s(k)$ at the server. Therefore the following measure accounts for differences in consecutive values for the encoding rate of the video being displayed to the user:

$$V = \frac{1}{\bar{r_s}} \sqrt{\frac{1}{m} \sum_{k=0}^{m-1} [r_s(k) - r_s(k+1)]^2} \tag{5}$$

where $m < n$ is the index of the last time slot during which the server has data left to stream, i.e. the streaming ends at time $t_{end} \in [t_m, t_{m+1})$. $\bar{r_s}$ denotes the mean value of the time series $\{r_s(k)\}$, for $k = 0, \ldots, m$.

Because the human eye is more likely to perceive a high variation in quality than a small one, this measure penalizes high differences in consecutive values for $r_s(k)$. Moreover, in FGS coding, the less important bit-planes correspond to higher values of $r_s(k)$. Therefore, for the same video with both layers encoded at the same r_b and r_e, the higher the mean value of the transmitted coding rates $r_s(k)$, the less visible the differences in consecutive values for $r_s(k)$. This is why our measure of rate variability is normalized by $\bar{r_s}$.

4. OPTIMAL TRANSMISSION POLICY

In this section we assume that the available bandwidth for the connection, $X(t)$, from beginning to end of transmission, is known a priori. This will allow us to formulate and solve an optimal stochastic control problem. The analysis and solution serves two purposes. First, it provides a useful bound on the achievable performance when bandwidth evolution is not known a priori. Secondly, the theory helps us design an adaptation heuristic for the realistic case when the bandwidth is not known. The optimization criteria studied in this paper prioritize three metrics: base-layer loss; bandwidth efficiency; and coding rate fluctuations.

4.1 Condition for No Losses

Losses of base layer data at the decoder may degrade considerably the perceived video quality. Depending on the level of error resilience used by the coding system, losses may cause freezing of the image for some time. Thus, base layer losses can be more disturbing for the overall quality than the number of bits used to code the video (represented by $\int_0^{t_{end}} X(t)dt$ in (4)). In this subsection we determine a necessary and sufficient condition for the transmission policy $\mathbf{r}_s = (r_s(0), \ldots, r_s(n-1))$, to have no base-layer loss. Recall that in our synchronous model no base layer loss implies no enhancement layer loss. To this end, denote:

$$\beta_k(\Delta) = \min_{t \in [t_k + \Delta, t_{k+1}]} \frac{\int_{t_k}^{t} X(u)\ du}{t - (\Delta + t_k)} \tag{6}$$

THEOREM 1. *The transmission policy* $(r_s(0), \ldots, r_s(n-1))$ *yields no loss of data over the whole decoding duration if and only if, for all* $k = 0, \ldots, n-1$, $r_s(k) \leq \beta_k(\Delta_k)$ *whenever* $\Delta_k < C$.

This theorem provides, for each slot, an upper-bound on the video coding rate that yields no loss. This bound depends on the contents of the prefetch buffers at the beginning of the slot, and on the available bandwidth during the duration of the slot.

PROOF. Having no loss of data over $[0, T]$ is equivalent to having no loss of data over each interval $[t_k, t_{k+1})$. Fix a $k \in \{0, \ldots, n-1\}$. If $\Delta_k \geq C$, there is enough data in the prefetching buffers at time t_k to insure the decoding without loss during the current slot $[t_k, t_{k+1})$ of length C seconds.

Now suppose that $\Delta_k < C$. Clearly there is no loss in the interval $t \in [t_k, t_k + \Delta_k]$, as the data in the prefetch buffers at time t_k is sufficient to feed the decoder up through time $t_k + \Delta_k$. At time $t_k + \Delta_k$ all of the data that was in the prefetch buffer at time t_k is consumed. Subsequently, the client starts to consume data that was sent after time t_k, which has been encoded at rate $r_s(k)$. Thus, after time $t_k + \Delta_k$ and at least up to time t_{k+1}, the client attempts to consume data at rate $r_s(k)$.

It follows that there is no loss if and only if for every time $t \in [t_k + \Delta_k, t_{k+1}]$ the total amount of data that the client attempts to consume in $[t_k + \Delta_k, t]$ is less than the amount of data that was transmitted in the interval $[t_k, t]$, that is, if and only if for all $t \in [t_k + \Delta_k, t_{k+1}]$

$$r_s(k) \cdot [t - (\Delta_k + t_k)] \leq \int_{t_k}^{t} X(u)\ du \tag{7}$$

Rearranging terms in the above equation gives the condition in the Theorem. □

Definition 1. Let $\mathcal{L}(\Delta_0)$ be the set of all possible transmission policies $\mathbf{r}_s$ that satisfy Theorem 1:

$$\mathcal{L}(\Delta_0) := \{\mathbf{r}_s \in [r_b, r_b + r_e]^n : \text{ no loss in } [0, T]\} \tag{8}$$

4.2 Maximizing Bandwidth Efficiency

In this subsection we consider the problem of maximizing bandwidth efficiency over all policies that give no loss. When the no loss condition in Theorem 1 holds, then the overall efficiency E given in (4) is maximized if and only if

$\int_0^{t_{end}} X(t)\ dt$ is maximized, which is equivalent to maximizing t_{end}.

Let $t_{end}(\mathbf{r}_s) \leq T$ be the time at which the server finishes streaming under transmission policy $\mathbf{r}_s \in \mathcal{L}(\Delta_0)$. For a fixed value of Δ_0, we can define the maximum ending time of streaming under all transmission policies in $\mathcal{L}(\Delta_0)$, as:

$$t_{end}^{max} = \max_{\mathbf{r}_s \in \mathcal{L}(\Delta_0)} t_{end}(\mathbf{r}_s) \tag{9}$$

We can observe that t_{end}^{max} only depends on Δ_0 and $X(t)$ for $t \in [0, T]$.

Now, let E^* be the maximum value of E that can be attained by a policy $\mathbf{r}_s \in \mathcal{L}(\Delta_0)$, and let $\mathcal{E}(\Delta_0)$ be the set of policies $\mathbf{r}_s \in \mathcal{L}(\Delta_0)$ that attain E^*. Since maximizing E is equivalent as maximizing t_{end}, we have:

THEOREM 2. *The set of transmission policies that maximizes E satisfies:*

$$\mathcal{E}(\Delta_0) = \{\mathbf{r}_s \in \mathcal{L}(\Delta_0) :\ t_{end}(\mathbf{r}_s) = t_{end}^{max}\} \tag{10}$$

In particular, the maximum value of E is given by:

$$E^* = \frac{\Delta_0}{T} + \frac{\int_0^{t_{end}^{max}} X(t)\ dt}{T \cdot (r_b + r_e)} \tag{11}$$

This simply states that, in order to maximize bandwidth efficiency, the streaming application should try to exploit the transmission channel as long as possible. A transmission policy which is not sufficiently aggressive in its choice of $r_s(k)$ may have streamed all the video frames long before the end of rendering at the client ($t_{end}(\mathbf{r}_s) < t_{end}^{max} \leq T$), and thereby not use the additional bandwidth that is available until the end of rendering.

Let $\bar{X} = \frac{1}{T}\int_0^T X(t)\ dt$ be the average available bandwidth during the playback interval $[0, T]$. A special case to consider is when $t_{end}^{max} = T$, i.e. when there exists transmission policies in $\mathcal{L}(\Delta_0)$ that can maintain the streaming until the very end of the rendering at the client. In this case, we have: $E^* = \frac{\Delta_0}{T} + \frac{\bar{X}}{r_b + r_e}$. Otherwise, when $t_{end}^{max} < T$, we will have: $E^* < \frac{\Delta_0}{T} + \frac{\bar{X}}{r_b + r_e}$.

4.3 Minimizing rate variability

From all the set of transmission policies $\mathbf{r}_s$ which yield no loss of data and maximize bandwidth efficiency E, i.e. $\mathbf{r}_s \in \mathcal{E}(\Delta_0)$, we will now look for those which minimize the rate variability V, given the available bandwidth $X(t)$. We will show that this problem can be solved by finding the shortest path in a graph. Let V^* be the minimum value of V that can be attained by a policy $\mathbf{r}_s \in \mathcal{E}(\Delta_0)$.

Definition 2. We define the optimal state graph $\mathcal{G}$ of our system as the graph represented in Figure 3, whose nodes represent, at each time t_k, all the possible sampled values for the buffering delay at time t_k, i.e. Δ_k. The arcs represent the evolution of the buffering delay after the streaming of the video from time t_k to time t_{k+1}, such that there is no loss of video data over the entire duration of the streaming. The initial state of the graph represents the initial build up of $\Delta(0) = \Delta_0$ seconds of video data, present in the client prefetch buffers at time $t = 0$. The final state is reached at the time when the server shall finish streaming its video data in order to maximize the bandwidth efficiency E, i.e. at time $t = t_{end}^{max} \in [t_m, t_{m+1})$.

THEOREM 3. *The problem of finding an optimal transmission policy $\mathbf{r}_s^* \in \mathcal{E}(\Delta_0)$ which minimizes the variability V can be solved by finding the shortest path in the system state graph $\mathcal{G}$.*

PROOF. From the definition of the optimal state graph $\mathcal{G}$, the no loss condition expressed in Theorem 1 is satisfied. Considering the streaming of the video by the server between times t_k and t_{k+1}, we thus can easily show that:

$$\forall k \in \{0, \ldots, m-1\}\ , \Delta_{k+1} = \Delta_k + \frac{\int_{t_k}^{t_{k+1}} X(t)dt}{r_s(k)} - C \tag{12}$$

This means that the transition from one state at time t_k to the next possible state at time t_{k+1} is completely determined by the choice of $r_s(k)$. Then, all the possible paths between the initial state to the final state will give all the possible transmission policies $\mathbf{r}_s \in \mathcal{L}(\Delta_0)$.

The final state of the graph insures that all these transmission policies will satisfy $t_{end}(\mathbf{r}_s) = t_{end}^{max}$, thus insuring by Theorem 2 that the maximum bandwidth efficiency is reached.

The cost of an arc from state $\Delta_k = \Delta$ to $\Delta_{k+1} = \Delta'$ will be denoted $c_k(\Delta, \Delta')$, as shown on Figure 3. This cost will be obtained recursively during the computation of the shortest path from the initial state to the final state, obtained by dynamic programming. It will be defined as $(r - r')^2$, where r is the unique value of $r_s(k)$ that makes the system transition from state $\Delta_k = \Delta$ to $\Delta_{k+1} = \Delta'$, and r' the value of $r_s(k+1)$ that makes the system transition from state $\Delta_{k+1} = \Delta'$ to the next state in the shortest path from $\Delta_{k+1} = \Delta'$ to the final state. This way, the shortest path from the initial state to the final state yields $\mathbf{r}_s^*$ which minimizes the measure of variability V, as given in Section 2. □

Given Δ_0 and $X(t)$, this theorem assumes the knowledge of the value of t_{end}^{max}. Let's assume that $\mathcal{E}(\Delta_0) \neq \emptyset$ (it also means that $\mathcal{L}(\Delta_0) \neq \emptyset$, i.e. we can at least stream all the base layer without loss). In this case, t_{end}^{max} is defined in $(0, T]$. We can first set its value to T and decrease it recursively until we find a shortest path in the optimal state graph $\mathcal{G}$. Indeed, if for a given possible value of t_{end}^{max} there exists no possible path in $\mathcal{G}$ from the initial state to the final state, it will mean that $\mathcal{E}(\Delta_0) = \emptyset$, which contradicts our hypothesis.

Given Δ_0 and $X(t)$, we have implemented the algorithm for finding the shortest path in graph $\mathcal{G}$ as well as t_{end}^{max}, yielding the minimum of variability V^*. The actual number of nodes in the graph is difficult to assess. It depends on the size of $\mathcal{L}(\Delta_0)$, the value of t_{end}^{max} and the sampling precision of the buffering delays Δ_k.

5. REAL-TIME RATE ADAPTATION ALGORITHM

Henceforth, we no longer assume that the available bandwidth $X(t)$ is known a priori for the whole duration of the streaming. Motivated by the theory of the previous section, we will provide a heuristic real-time policy that will adapt on-the-fly to the variations of $X(t)$. The theory of the previous section will also provide a useful bound to which we will be able to compare the performance of our heuristic.

for $k = 0$ to $n-1$:
- Retrieve the value of Δ_k from the client
- Compute $X_{avg}(k-1)$
- Compute the value of $r_s(k)$:

If $\Delta_k \leq C$
$r_s(k) = r_b$
Else if $C < \Delta_k \leq 2C$
$r_s(k) = \alpha \cdot X_{avg}(k-1) + (1-\alpha) \cdot r_s(k-1)$
Else if $\Delta_k \geq 2C$
$r_s(k) = \alpha \cdot X_{avg}(k-1) \cdot \frac{\Delta_k}{2C} + (1-\alpha) \cdot r_s(k-1)$

Figure 4: Real-time Algorithm

Figure 4 presents our real-time algorithm. At the beginning of each time slot of length C seconds, i.e., at each time t_k, the server fixes the encoding rate for the slot, i.e., it fixes $r_s(k)$. Recall that the server knows the number of seconds of video data contained in the client prefetch buffers, Δ_k, at each time t_k. The server can compute the average goodput as seen in the previous slot, between time t_{k-1} and t_k, denoted by $X_{avg}(k-1)$. It can be shown that if there is no loss of data in the previous time slot, then $X_{avg}(k-1)$ can be expressed as: $X_{avg}(k-1) = r_s(k-1) \cdot \frac{\Delta_{k-1} - \Delta_k + C}{C}$.

As shown in Figure 4, at the beginning of each slot the algorithm operates according to whether $\Delta_k \leq C$, $C < \Delta_k \leq 2C$, or $\Delta_k \geq 2C$. When $\Delta_k < C$, there is potential loss in the upcoming slot; because minimizing base-layer loss is our most important objective, we set $r_s(k) = r_b$, that is, we send only the base layer during the slot. The no loss condition as expressed in Theorem 1, i.e., $r_s(k) \leq \beta_k(\Delta_k)$, would give a less conservative choice for $r_s(k)$, but $\beta_k(\Delta_k)$ strongly depends on the variations of the available bandwidth in the next C seconds, which is very difficult to predict. Additionally, this choice attempts to maintain a minimum of C seconds of data in the client buffer, which will be sufficient to mitigate jitter and allow for retransmission of lost packets.

When $C < \Delta_k \leq 2C$, the server can start increasing the value of $r_s(k)$. In order to maintain a high bandwidth efficiency E, we know from Theorem 2 that we must make the streaming last as long as possible, i.e., we need to maximize t_{end}. Therefore, we use a video encoding rate that tracks the average available bandwidth of the connection. But the values of the averages $X_{avg}(k)$ may have large fluctuations, as discussed in Section 1. So, we include a smoothing factor $\alpha \in (0,1)$, which aims to smooth the variations of $X_{avg}(k)$. By choosing $r_s(k) = \alpha \cdot X_{avg}(k-1) + (1-\alpha) \cdot r_s(k-1)$, we try to minimize the differences in consecutive values of $r_s(k)$, while getting close to a smoothed average of the available bandwidth. The value of α can be chosen to trade off small quality variability (small α) with better overall bandwidth utilization (high α). When $\Delta_k > 2C$, our heuristic is more aggressive with respect to the available bandwidth (by a factor of $\frac{\Delta_k}{2C}$). The heuristic further increases the value of $r_s(k)$, resulting in less video data in the client prefetch buffers, and preventing the streaming from ending too early.

5.1 Simulations from Internet traces

We first made simulations from real Internet TCP traces. We used `snoop` on Solaris to collect goodput from 5mn-long TCP connections at different times of the day between University of Pennsylvania, Philadelphia and Institut Eurecom, France. We performed all of the simulations with a time slot length of $C = 5$s, and a video length of $T = 300$s. The client prefetch buffers collect $\Delta_0 = 6$s of video data before the client starts decoding and rendering. We used four traces, whose 1 second averages are depicted in Figure 5.

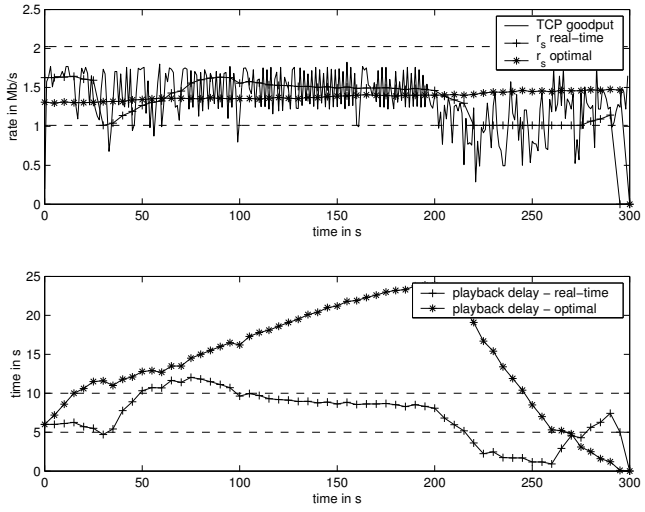

Figure 6: Rate adaptation - trace 1, $r_b = r_e = 1$ Mbps, $\bar{X}_1 = 1.35$ Mbps, $\alpha = 0.2$

Figure 6 shows the results of a simulation, for which we used TCP trace 1 with average available bandwidth $\bar{X}_1 = 1.35$ Mbps. The coding rates of both layers are set to $r_b = r_e = 0.75 \cdot \bar{X}_1 = 1$ Mbps, so that the total coding rate of the video is strictly superior to the average available bandwidth. The smoothing factor is set to $\alpha = 0.2$. The top plot shows the transmitted video encoding rate of our real-time heuristic, the encoding rate of the optimal transmission policy (given by Theorem 3) and the connection goodput. The plot below shows, at each time t_k, the amount of data in seconds in the client prefetch buffers, Δ_k, for both the real-time and the optimal transmission policies. We observe that the optimal policy prefetches up to 25 seconds of video data during playback to smooth bandwidth variations and achieve the maximum bandwidth efficiency. The real-time transmission policy prefetches up to 12 seconds during playback to smooth bandwidth fluctuations, although it doesn't smooth as well as the optimal transmission policy. The top plot shows that the real-time algorithm adapts well to the varying available bandwidth. In particular, after time $t = 200$ms the video coding-rate drops to its minimum value, r_b, because the playback delay drops below $C = 5$ seconds, as shown on the second plot. The real-time transmission policy provides a bandwidth efficiency E that is close to the maximum ($E = .63$ compared to $E^* = .68$). Indeed, the real-time policy nearly satisfies Theorem 2 since it doesn't stop transmitting video frames until just one time slot before the last one ($r_s(295) = 0$ on the graph).

In order to study the performance of our real-time algorithm in various bandwidth situations with respect to the video coding rate, we define the normalized base layer coding rate $r_n = \frac{r_b}{\bar{X}}$. For simplicity, we assume $r_b = r_e$ throughout the rest of the paper. Figures 7 and 8 show the evolution of the measures E and V as a function of r_n for the same trace

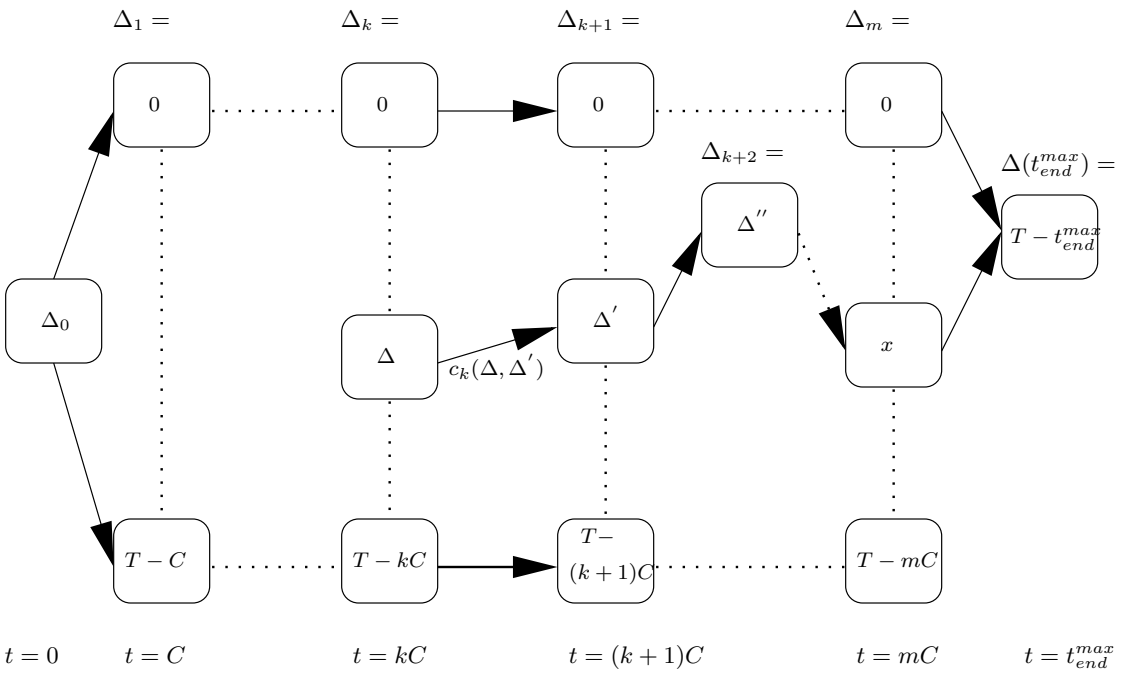

Figure 3: Optimal state graph $\mathcal{G}$

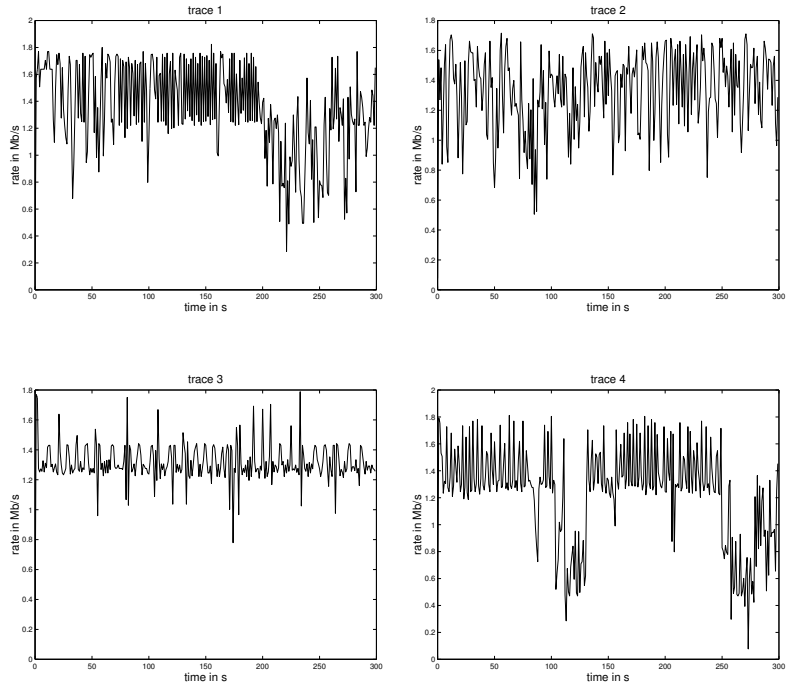

Figure 5: 1 second average goodput of the collected TCP traces

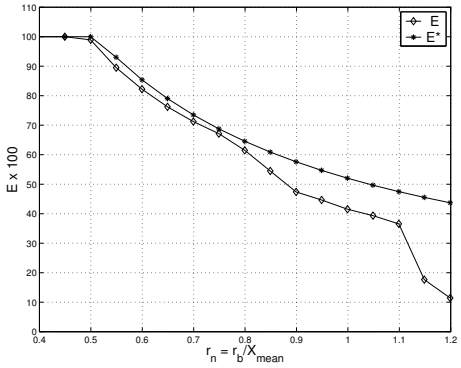

Figure 7: Bandwidth Efficiency, $\alpha = 0.2$

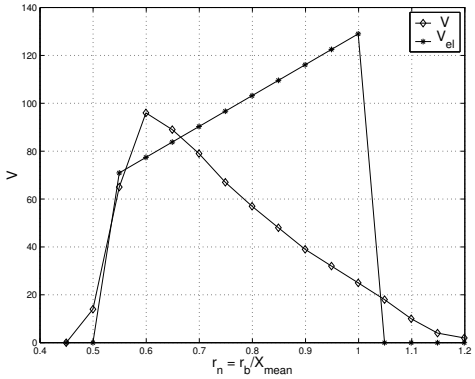

Figure 8: Rate Variability, $\alpha = 0.2$

as in Figure 6 and the same constants. Figure 7 depicts the variations of E compared to the maximum achievable bandwidth efficiency as given in Theorem 2. We see that the real-time heuristic's overall bandwidth efficiency is close to the maximum for r_n between 0.4 and 0.8, corresponding to the favorable situation when the average available bandwidth is high enough to sustain the streaming of the base layer and a part of the enhancement layer without loss. When $r_n > 0.8$, the average available bandwidth is closer to the base layer coding rate, resulting in some loss of base layer data, which increases the difference between the achieved bandwidth efficiency and the maximum.

Figure 8 shows the evolution of variability (as defined in Section 3), given by our adaptation heuristic. It also shows the value of V, denoted by V_{el}, obtained with the streaming policy which just adds or removes the entire EL once during the whole streaming duration. We did not plot the minimum variability obtained by the optimal allocation since it is always very close to zero, and thus insignificant. When we consider the variability of our real-time algorithm for FGS video, we notice from Figure 8 that V reaches a maximum at about $r_n = 0.6$, then decreases as r_n increases. Indeed, when r_n becomes large, the average available bandwidth becomes lower than $r_b + r_e$, resulting in fewer opportunities to stream a high bit-rate video, i.e., to choose high values for $r_s(k)$. We also see that, in most bandwidth conditions, the total variability obtained by our heuristic is lower or roughly equal to V_{el}, which gives an indication of the low level of rate variations achieved by our heuristic.

In Table 2, we give the results in terms of E, V and losses of video data for the three other traces in Figure 5, and for different values of r_n. The results also confirm the good performance of our real-time algorithm in terms of high bandwidth efficiency and low variability for a variety of bandwidth scenarios.

5.2 Simulations from an end-to-end MPEG-4 FGS streaming platform

We also did simulations with a local testbed consisting of a streaming server and a client with an MPEG-4 FGS-decoder (provided by Thomson Multimedia through the French national project VISI). The video was a 25 fps 4-minute video clip depicting the city of Brest, France and featuring both high and low motion sequences. The BL and EL coding rates were respectively $r_b = 256$ Kbps and $r_e = 768$ Kbps. The link between the server and the client was set to $X_{max} = 1024$ Kbps $= r_b + r_e$, and variable cross traffic with mean $\bar{X} = 512$ Kbps was generated from the server to the client, in order to yield a variable available bandwidth for the streaming application. Table 3 shows the results obtained by our adaptive algorithm when $\Delta_0 = C = 5$ sec and $\alpha = 0.2$, compared with the cases of BL only, both layers without loss, and the case when there is no rate-adaptation with a constant encoding rate $r_s(k) = \bar{X} = 512$ Kbps. Results are given in terms of mean PSNR and sum of absolute differences in PSNR for successive images. As we can see, the adaptation algorithm gives better overall image quality compared with the static case, at the price of a small increase in total quality variability between successive images (still less than the variability induced when streaming only the BL).

6. STREAMING OVER TCP-FRIENDLY ALGORITHMS

In order to reduce image quality variation, a number of studies advocate the use of smooth-rate TCP-friendly algorithms for streaming [16, 11, 6]. In this section we show that, for stored FGS-encoded video, streaming over ordinary TCP can achieve essentially the same low level of quality variability that can be achieved with a smooth-rate TCP-friendly algorithm. We achieve this low-level of variability by combining prefetching and coarse-grained rate adaptation, as provided by our real-time heuristic of the previous section.

Our real-time adaptation algorithm can be run on top of TCP or on top of a TCP-friendly algorithm such as TFRC (TCP-Friendly Rate Control), defined in [6]. We used `ns` to collect TCP and TFRC traces under the same network conditions. The network topology is the commonly employed "single bottleneck," for which congestion only occurs in the link between two routers (we used access links with delay 10ms and a 15Mbits bottleneck link with delay 50ms). If the playback delay between the server and the client, $\Delta(t)$, is maintained at an order of a few seconds, the server has the ability to retransmit lost packets. Therefore we assume that our TFRC connection can be made fully reliable. We use a RED bottleneck queue to avoid global TCP synchronization. From these collected traces, we run the real-time adaptation algorithm for different values of r_n and α, and compare the results in terms of measures E and V. We de-

	Trace 2 $\bar{X} = 1.34$ Mbps			Trace 3 $\bar{X} = 1.31$ Mbps			Trace 4 $\bar{X} = 1.40$ Mbps		
	E/E^*	V/V_{el}	losses	E/E^*	V/V_{el}	losses	E/E^*	V/V_{el}	losses
$\frac{r_b}{\bar{X}} = 0.6$	.83/.85	52/116	0	.84/.85	68/116	0	.83/.85	106/116	0
$\frac{r_b}{\bar{X}} = 0.75$	.66/.69	33/97	0	.67/.69	46/97	0	.63/.69	86/97	0.7s
$\frac{r_b}{\bar{X}} = 0.9$	.56/.58	19/77	0	.56/.58	25/77	0	.52/.58	50/77	1.1s

Table 2: Simulations from Internet traces

	mean PSNR	diff PSNR
BL without loss	29.46	1944
EL with $r_s(k) = cst = 512$ Kbps	29.77	1524
EL with $\Delta_0 = C = 5$ s, $\alpha = 0.2$	29.94	1762
EL without loss	31.83	1468

Table 3: Simulation with a MPEG-4 FGS decoder

load	V^*_{tfrc}	V^*_{tcp}	V_{tfrc}	V_{tcp}	V_{el}	α_{tfrc}	α_{tcp}
5	6	6	42	74	97	0.15	0.01
10	6	10	61	55	97	0.01	0.01
15	5	8	42	43	97	0.07	0.04
20	7	12	8	85	97	0.02	0.02
25	6	8	49	93	97	0.15	0.01
30	6	9	50	23	97	0.15	0.01
35	6	6	49	82	97	0.05	0.2
40	6	11	83	116	97	0.01	0.04

Table 4: Performance as a function of network load

fine the network load as the number of simultaneous (TCP and TFRC) connections inside the bottleneck. We vary the load between 5 and 40.

The first part of Table 4 shows, as a function of the network load for a base layer normalized coding rate of $r_n = 0.75$, the minimum variability achieved by the optimal transmission policy given in Theorem 3. As we can see, the minimum variability when using TCP is only slightly higher than the minimum variability that can be achieved when using TFRC, even though the TFRC long-term throughput is considerably smoother than TCP, especially at low loss rates [6, 17]. We also observe that for both cases V^* remains low for all network loads, which indicates that our application-layer smoothing approach has the potential to work well in a wide range of network conditions.

We then applied our real-time algorithm to both the TCP and TFRC traces. For a given network load, we varied the smoothing parameter α between 0.01 and 0.95, which gives different values for the couple (E, V). Then, among the choices of α that bring E to within 1% of the maximum, we keep the α that minimizes the variability V. The second part of Table 4 gives the results for V_{tfrc} and V_{tcp}, along with the variability obtained with a transmission policy that would add the full EL just once, denoted by V_{el}. The last two columns show the corresponding value of α. We first observe that V_{tfrc} is, for most network loads, less than V_{tcp}. However, both values remain quite low (usually less than V_{el}), and the difference may probably not be noticed by the user. We also observe that for a network load with less than 30 competing TCP and TFRC connections, the smoothing parameters that minimizes V while insuring a high E satisfy $\alpha_{tfrc} \geq \alpha_{tcp}$. Indeed, because TFRC has smoother rate variations than TCP in a low loss environment, the application will need to smooth bandwidth variations of TCP more than TFRC.

Finally, Figure 9 shows the rate adaptation provided by the optimal policy and the real-time algorithm for TCP and TFRC connections, respectively. The bottleneck link is shared by 25 long-lived TCP and 25 long-lived TFRC connections, and the base layer normalized encoding rate is set to $r_n = 0.75$. In both cases, we use the optimal value of the smoothing parameter α in our real-time algorithm (as given in Table 4) for a network load of 25, i.e., we use $\alpha_{tfrc} = 0.15$ and $\alpha_{tcp} = 0.01$. As in Figure 6, the top plot in both figures shows the coding rate of our real-time heuristic, the coding rate of the optimal transmission policy, and the connection goodput. The bottom plot shows, at each time t_k, the amount of data in seconds in the client prefetch buffers, Δ_k, for both the real-time and the optimal transmission policies.

We first compare the real-time transmission policies in both figures. The top plots show that the real-time rate adaptation algorithm yields very smooth variations in the transmitted video coding rate for both TCP and TFRC, which is consistent with the low values obtained for V_{tfrc} and V_{tcp} in Table 4 when $load = 25$. Furthermore, the real-time algorithm sustains the duration of the streaming almost until the end of the rendering in both cases, ensuring a high overall bandwidth efficiency. When comparing the bottom plots, we see however that the real-time algorithm needs to prefetch up to 30 seconds of video to smooth variations of TCP, while it needs to prefetch only 13 seconds of video in the case of TFRC. When comparing the optimal transmission policies in both figures, we see that the variability attained is negligible in both cases, and the optimal transmission policies require up to 10 seconds of video data to be prefetched in both cases.

7. CONCLUSION

We presented a new framework for streaming stored FGS encoded videos. We derived analytical results and a method to find an optimal transmission policy, which maximizes a measure of bandwidth efficiency, while minimizing a measure of coding rate variability. We then presented a real-time algorithm for adaptive streaming of FGS video. Our simulations showed that our heuristic yields near-optimal performance in a wide range of bandwidth scenarios. Finally, in the context of streaming stored FGS video, using client buffering, we have argued that streaming over TCP gives video quality results that are comparable with streaming over smoother TCP-friendly connections. In future work we intend to improve the performance of our heuristic by tak-

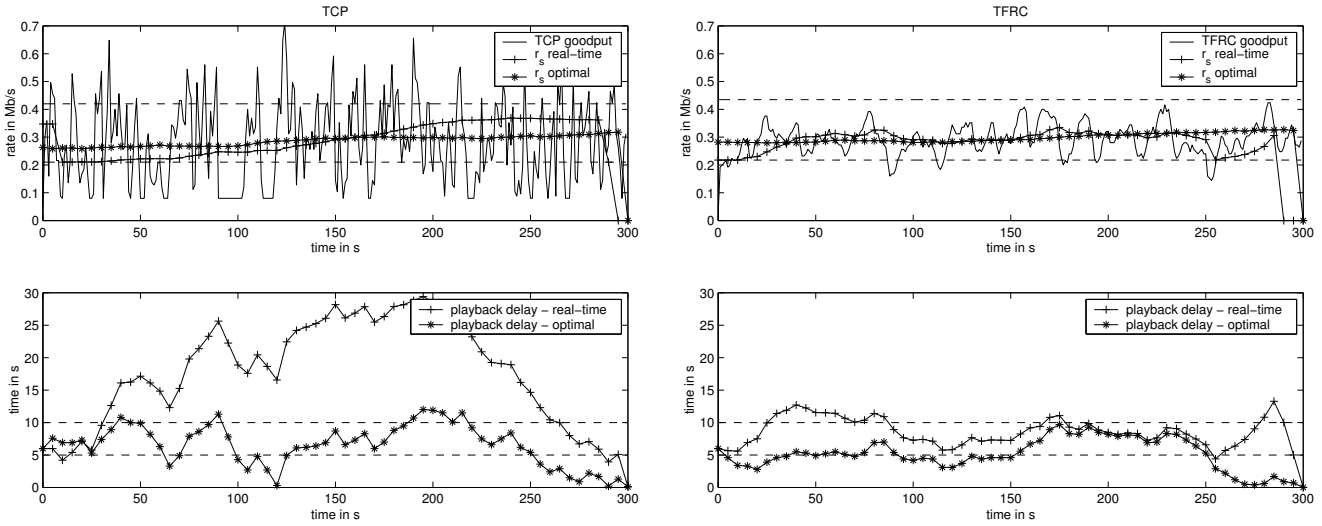

Figure 9: Rate adaptation for TCP and TFRC - $load = 25$**,** $\alpha_{tcp} = 0.01$**,** $\alpha_{tfrc} = 0.15$

ing into account the different rate-distortion characteristics of successive video scenes.

8. REFERENCES

[1] *ISO/IEC JTC1/SC29/WG11 Information Technology - Generic Coding of Audio-Visual Objects : Visual ISO/IEC 14496-2 / Amd X*, December 1999.

[2] R. Aravind, M. R. Civanlar, and A. R. Reibman. Packet Loss Resilience of MPEG-2 Scalable Video Coding Algorithms. *IEEE Trans. Circuits and Systems for Video Technology*, 6:426–435, October 1996.

[3] D. Bansal and H. Balakrishnan. Binomial Congestion Control Algorithms. In *Proc. of IEEE Infocom*, pages 631–640, Anchorage, AL, May 2001.

[4] P. de Cuetos, D. Saparilla, and K. W. Ross. Adaptive Streaming of Stored Video in a TCP-Friendly Context : Multiple Versions or Multiple Layers ? In *Proc. of the International Packet Video Workshop*, Kyongju, Korea, May 2001.

[5] S. Floyd and K. Fall. Promoting the Use of End-to-End Congestion Control in the Internet. *IEEE/ACM Trans. on Networking*, 7(4):458–472, August 1999.

[6] S. Floyd, M. Handley, J. Padhye, and J. Widmer. Equation-based Congestion Control for Unicast Applications. In *Proc. of ACM SIGCOMM*, Stockholm, Sweden, August 2000.

[7] W. Li. Overview of Fine Granularity Scalability in MPEG-4 Video Standard. *IEEE Trans. on Circuits and Systems for Video Technology*, 11(3):301–317, March 2001.

[8] D. Loguinov and H. Radha. On Retransmissions Schemes for Real-time Streaming in the Internet. In *Proc. of Infocom*, pages 1310–1319, Anchorage, AL, May 2001.

[9] S. Nelakuditi, R. Harinath, E. Kusmierek, and Zhang Z. Providing Smoother Quality Layered Video Stream. In *Proc. of NOSSDAV '00*, Chapel Hill, North Carolina, June 2000.

[10] V. Paxson. End-to-End Internet Packet Dynamics. *IEEE/ACM Trans. on Networking*, 7(3):277–292, June 1999.

[11] R. Rejaie, D. Estrin, and M. Handley. Quality Adaptation for Congestion Controlled Video Playback over the Internet. In *Proc. of ACM SIGCOMM*, pages 189–200, Cambridge, September 1999.

[12] R. Rejaie, M. Handley, and D. Estrin. RAP: An End-to-End Rate-Based Congestion Control Mechanism for Realtime Streams in the Internet. In *Proc. of IEEE Infocom*, pages 1337–1345, New York, March 1999.

[13] I. Rhee. Error Control Techniques for Interactive Low-bit Rate Video Transmission over the Internet. In *Proc. of ACM SIGCOMM*, pages 290–301, September 1998.

[14] D. Saparilla and K. W. Ross. Optimal Streaming of Layered Video. In *Proc. of IEEE Infocom*, pages 737–746, Tel Aviv, Israel, March 2000.

[15] D. Saparilla and K. W. Ross. Streaming Stored Continuous Media over Fair-Share Bandwidth. In *Proc. of NOSSDAV '00*, Chapel Hill, North Carolina, June 2000.

[16] W. Tan and Zakhor. A. Real-Time Internet Video Using Error Resilient Scalable Compression and TCP-Friendly Transport Protocol. *IEEE Trans. on Multimedia*, 1(2):172–186, June 1999.

[17] Y. Yang, M. Kim, and S. Lam. Transient Behaviors of TCP-fiendly Congestion Control Protocols. In *Proc. of IEEE Infocom*, pages 1716–1725, Anchorage, AL, May 2001.

[18] Q. Zhang, W. Zhu, and Y-Q. Zhang. Resource Allocation for Multimedia Streaming over the Internet. *IEEE Transactions on Multimedia*, 3(3):339–355, September 2001.