Adaptive Streaming of Stored Video in a TCP-Friendly Context: Multiple Versions or Multiple Layers?

Philippe de Cuetos, Despina Saparilla, Keith W. Ross
Institut EURECOM
2229, route des Crêtes
06904 Sophia Antipolis, France
{decuetos, saparill, ross}@eurecom.fr

Abstract

Video transmission over the current best-effort Internet should be made fair to competing TCP traffic. Because the available bandwidth to a TCP-friendly stream changes significantly over medium and long time scales, it is desirable to adapt the streaming rate of the video to the current bandwidth conditions. We consider two adaptive streaming schemes for stored video. The first scheme switches among multiple encoded versions, with each version encoded at a different rate. The second scheme adds and drops encoding layers. To compare the two schemes, we develop streaming control policies for each scheme and evaluate their performance using trace-driven simulation. Our results show that when analogous streaming policies are used, switching versions outperforms adding/dropping layers because of the overhead associated with layering. However, the enhanced flexibility of layering can compensate the performance degradation due to the layering overhead.

1 Introduction

As multimedia applications become widespread on the Internet, the problem of streaming stored multimedia over the best-effort Internet while remaining fair to competing TCP traffic has become increasingly important. The need to be fair to competing TCP traffic has lead to the notion of TCP-friendly streams [4, 2, 6, 8]. The available bandwidth to a TCP-friendly scheme varies significantly over both medium and long time scales [10]. For streaming stored video, it is therefore useful to consider schemes that adapt video quality as a function of the available bandwidth. At the same time, the scheme should not lead to frequent changes in perceived video quality, in order to maximize overall viewer satisfaction [7].

There are three popular techniques for adapting stored video to time-varying available bandwidth: on-the-fly encoding, adding/dropping layers, and switching among multiple encoded versions. On-the-fly encoding is CPU intensive and thus generally regarded as unsuitable for streaming stored video. In this paper we focus on the remaining two techniques, i.e., "adding/dropping layers" and "switching versions".

Adding/dropping layers has been proposed by many researchers as an effective scheme that provides both rate adaptation and error resilience in video communications [5, 3, 1, 9]. In the adding/dropping layers scheme, the video stream is partitioned into several layers. It is composed of a base layer, which contains the most essential information for the reconstruction of the video, and results in low but generally acceptable quality, and one or several enhancement layers that contain additional information and deliver a higher quality video. The video

streaming application can add or drop the higher quality layers to adjust the transmission rate to the available bandwidth. A key characteristic of layered encoding is that in order to decode the higher quality layers, all lower quality layers should also be available at the decoder.

Although adding/dropping layers is an effective video transmission technique, it has certain drawbacks. Layered encoding increases the complexity of the video coding significantly and results in inferior coding efficiency than non-layered compression. Layering introduces a coding overhead at the source coder and the transport layer, which depends on several factors, including the particular layering structure employed, the bit rate of the video stream, and the spatial and temporal resolution of the source [1]. Nevertheless, when the base layer is delivered with higher reliability [14] (by combining layered encoding with transport prioritization mechanisms, such as FEC), adding/dropping layers offers a higher resilience to transmission errors relative to a non-layered scheme.

Switching versions, is widely used in the industry. For switching versions, the video is compressed at different rates (and therefore different quality levels), and each version is stored and available at the server. The server detects changes in available bandwidth and switches among the versions to adapt the transmission rate (and hence quality level) to the available bandwidth. Although non-layered compression into multiple versions increases the storage requirements at the server, it is simpler and results in higher video quality when compared to layered encoding, under a constant-bit-rate constraint [1].

In this paper we compare adding/dropping layers with switching versions for streaming stored video while being TCP-friendly. We first present our model and the performance metrics we use to compare the two schemes. We then design streaming control policies for adding/dropping layers, and for switching versions. The heuristic streaming policies developed for each scheme are analogous, in order to permit a fair comparison of the two schemes. We compare the adaptive streaming schemes in simulation experiments in which we vary critical conditions, such as (i) the average available bandwidth, and (ii) the percent bit-rate of overhead that is associated with layered video. Our simulations experiments use TCP throughput traces collected from Internet experiments. In the following section we briefly describe the collected throughput traces.

2 TCP-Friendly Internet Bandwidth

To compare the two streaming schemes, adding/dropping layers and switching versions, we need to obtain TCP-friendly traces. Several TCP-friendly rate adjustment protocols have recently been developed by researchers [6, 12, 13, 8, 11]. The rate adjustment protocols reported in the literature have different limitations, depending on the TCP characterization on which they are based, and achieve TCP friendliness only under specific types of scenarios. For example, some of the proposed protocols are not TCP-friendly at loss rates higher than 5% [8, 11]. Some of the protocols are specific to multicast applications and rely on data layering [12, 13]. We do not use any of the TCP-friendly rate adjustment schemes in the literature to generate TCP-friendly traces. Instead, it is natural to suppose that the perceived rate fluctuations of a TCP-friendly scheme exhibit similar behavior as the fluctuations of TCP throughput over medium (seconds) and long (minutes) time scales. To obtain TCP-friendly bandwidth conditions we collect throughput traces from TCP connections on the Internet.

We traced a number of unidirectional TCP bulk data transfers between three pairs of hosts, located in Finland, France and the United States. From the experiments, we collected several

Trace	Src-Dest.	date	Throughput (Mbps)		
			Peak	Mean	σ
A1	US-FR	29-06-99 15:00	2.41	0.70	0.43
A2	US-FR	29-06-99 16:00	3.89	1.10	0.82

Table 1: Summary of 1-hour long traces.

1-hour long instantaneous throughput traces for TCP flows during four consecutive days at different times of the day. In this work we present results obtained using only two of the collected traces due to space limitations. Table 1 summarizes statistics for the two traces, and Figure 1 shows the average throughput of these traces over time scales of 10 and 100 seconds. As shown in the figure, both traces exhibit a high degree of variability and burstiness over both time scales. A more detailed description of the throughput measurements and collected traces can be found in [10].

3 Video Streaming Model and Assumptions

We develop a fluid transmission model to analyze the video streaming problem. We denote by $X(t)$ the TCP-friendly bandwidth that is available to the streaming application at time t. For $X(t)$ we use Internet traces, with each trace being averaged over 1-second intervals. Our model allows for an initial playback delay, denoted by Δ, which is set to four seconds in all numerical work. We assume that the delay between the server and the client is zero; this assumption is reasonable, given that round trip times are relatively small, and often an order of magnitude smaller than the initial playback delay. We also suppose that the source host always sends data at rate $X(t)$, and all data sent into the network will eventually be consumed at the receiver (i.e., the server only transmits data that will meet their deadline for consumption). Thus, packet loss from the video stream occurs only due to buffer starvation. Finally, we suppose that the prefetch buffer at the client is not restricted in size (this assumption is motivated by disk sizes in modern PCs).

For simplicity, we suppose that the video is CBR-encoded. In the multiple versions scheme, we consider two non-layered versions each encoded at a different rate and each stored at the server. We denote the encoded rate of the low-bit rate version by r_1 bits per second, and the encoded rate of the high bit-rate version by r_2 bits per second. We will refer to the low bit-rate version as version v_1 and to the high bit-rate version as version v_2. In the layered video scheme, we suppose that the base layer can be decoded independently to generate a comparable video quality to the the low bit-rate version v_1. We also suppose that there is also a single enhancement layer, which when decoded with the base layer, delivers a quality comparable to version v_2. We let r_b and r_e denote the rates of the base and enhancement layers, respectively.

3.1 Comparison of Rates

To fairly compare the two streaming approaches, we need to define the relationship between rates of the layers and those of the two versions. Layered encoding results in a lower compression gain than non-layered coding. For example, a study of MPEG-2 scalability structures [1] has shown that in all cases, layered coding has a coding penalty, which reduces the overall viewing quality when compared to non-layered coding, under a constant-bit-rate constraint. The

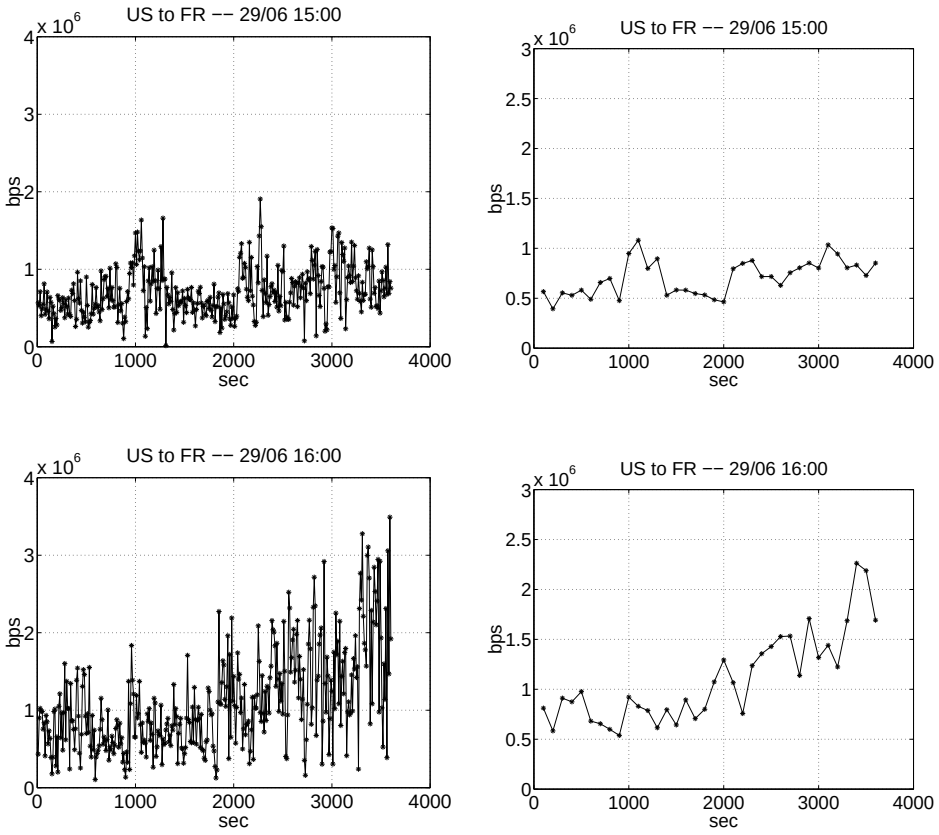

Figure 1: Average throughput over time scales of 10 and 100 seconds for traces A1 and A2.

coding penalty of layering depends on several factors, including the particular scalability technique used. The study of MPEG-2 scalability structures in [1] showed that data partitioning has the smallest coding overhead and spatial scalability the largest.

In this study we adapt a simple approach, which nevertheless allows for a realistic comparison of the two streaming schemes. We denote the overall percent coding overhead of layering by H. As explained in [14], the overhead introduced by layering can be due to source coding or transport. In data partitioning, for instance, the coding overhead is due to additional headers in the enhancement layer stream needed for synchronization with the base layer stream. In SNR scalability, the enhancement layer is a re-quantization of the base layer at a finer resolution, so that both layers include some information about all DCT coefficients. We assume that the bit-rate coding overhead is associated with the enhancement layer. We identify the following relationships for rates r_b, r_e, and r_1, r_2 :

$$r_b + r_e = (1 + H) \cdot r_2 \quad \text{and} \quad r_b = r_1. \tag{1}$$

In our numerical work H is a parameter that we vary between 1 and 10%. We believe that this upper limit on H is conservatively chosen. (Some justification for the selection of the above bounds for H can be found in [1]).

3.2 Performance metrics

We compare the performance of the two adaptive streaming schemes based on three metrics:

- *Fraction of high-quality viewing time.* We denote by t_h the fraction of time that the best quality of the encoded video can be viewed at the client. In the case of layered video, t_h is the fraction of time both layers are delivered and decoded together. In the case of multiple versions, t_h is the fraction of time that version v_2 is rendered to the client.

- *Fraction of time the decoder can not display the video.* We denote by t_d the fraction of time during which the decoder can not render a part of the video of either quality to the receiver. In the case of layered video, t_d is the fraction of time that the base-layer prefetch buffer is starved. In the case of switching versions, t_d is the fraction of time that neither v_1 nor v_2 data is available for consumption.

- *Quality fluctuations.* We denote by S the total number of times that the video quality changes at the decoder. In the case of layered video, there are quality fluctuations when the enhancement layer is dropped or added, or when the enhancement layer prefetch buffer at the client is starved. In the case of multiple versions, S is the total number of switches among the two versions during the viewing duration of the video, that is switching from version v_1 to v_2, and vice versa.

4 Streaming Control Policies

In this section we develop control policies for adding/dropping layers and for switching among versions. A control policy determines when to add and drop the enhancement layer, as well as how to allocate bandwidth among the layers (when both layers are being streamed). Results in [9] have shown that control policies based on the content of the prefetch buffers at the client attain good performance in a TCP-friendly context. Some simple threshold control policies were introduced in [10] for layered encoded videos. In this paper we design equivalent control policies for adding/dropping layers and for switching versions, so that we can fairly compare the two adaptive streaming schemes.

4.1 Adding/dropping layers

We begin by describing the control policy for adding/dropping layers. At any time instant, the server must determine how to allocate bandwidth among the layers. We first define some useful notation for describing the bandwidth allocation problem. We denote $\pi_b(t)$ and $\pi_e(t)$ the fraction of $X(t)$ that is allocated to the base layer and the enhancement layer at time t, respectively. We denote by $Y_b(t)$ and $Y_e(t)$ the contents of the base and enhancement layer prefetch buffers at the client at time t, respectively. As shown in Figure 2, at time t the base-layer prefetch buffer is fed at rate $\pi_b(t)X(t)$ and, when nonempty is drained at rate r_b. An analogous statement is true for the enhancement layer. We suppose that the server can estimate the amount of data in the client prefetch buffers using regular receiver reports (e.g., RTP/RTCP reports).

Based on the prefetch buffer contents at the client and on available bandwidth conditions, the server decides at each time instant whether to add/drop the enhancement layer. The goal of our control policy is to maintain continuous viewing of the base layer (i.e. avoid buffer starvation) while at the same time maximize the overall playback quality by rendering

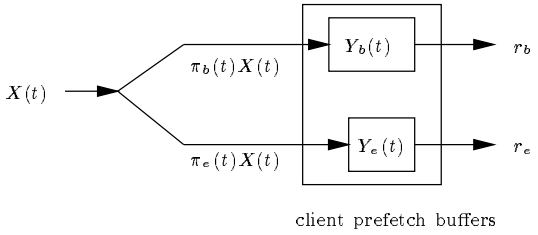

Figure 2: System of client prefetch buffers in layered streaming model.

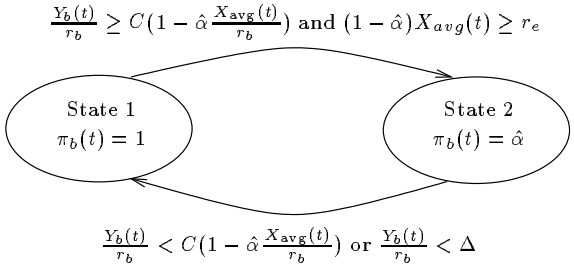

Figure 3: State transition diagram of a streaming control policy for adding/dropping layers.

the enhancement layer for long periods. Figure 3 shows a state transition diagram for the adding/dropping layers policy. The server begins by streaming only the base layer; in state 1, all available bandwidth is allocated to the base layer, i.e., $\pi_b(t) = 1$. When the enhancement layer is added, the server begins sending data from both layers and the process transitions to state 2. In state 2, available bandwidth is allocated among the layers in proportion to each layer's consumption rate, i.e., $\pi_b(t) = \hat{\alpha}$ where $\hat{\alpha} = \frac{r_b}{r_b+r_e}$.

Two conditions control the transition from state 1 to state 2. The first condition requires that when the server adds the enhancement layer, the amount of buffered base layer data at the client is enough to avoid future buffer starvation. In particular the server will add the enhancement layer at time s if the following holds:

$$\frac{Y_b(s)}{r_b} \geq C(1-\hat{\alpha}\frac{X_{\text{avg}}(s)}{r_b}) \tag{2}$$

In the above expression, $X_{\text{avg}}(s)$ is the most recent estimate of average available bandwidth. This estimate is obtained using a Weighted Exponential Moving Average (WEMA) of all previous bandwidth observations at the server. C is a constant denoting the prediction interval over which the estimate of average bandwidth is considered useful. Condition (2) requires that the amount of buffered base layer data at time s is enough to avoid starvation, given that average available bandwidth during the next C seconds equals the most recent bandwidth estimate $X_{\text{avg}}(s)$, and that a fraction $\hat{\alpha}$ of the available bandwidth will henceforth be allocated to the base layer. The second condition for adding the enhancement layer at time s requires that the transmission rate of enhancement layer data at time s (and during the prediction interval) exceeds the consumption rate, i.e., $(1-\hat{\alpha}) \cdot X_{\text{avg}}(s) \geq r_e$. This condition aims at avoiding rapid quality fluctuations caused by frequently starving the enhancement layer buffer. The server will add the enhancement layer at time s only if both conditions hold.

The server drops the enhancement layer when the likelihood of base layer buffer starvation becomes high. To avoid starvation of the base layer buffer, the server drops the enhancement

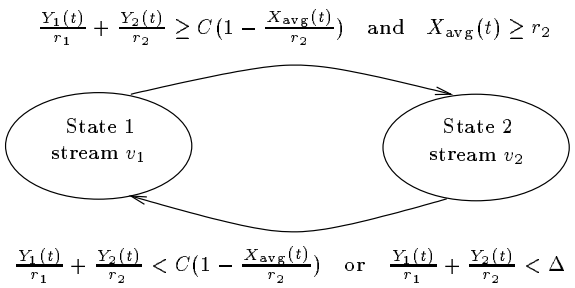

Figure 4: State transition diagram of a streaming control policy for switching versions

layer at time s if (2) does not hold. The server will also drop the enhancement layer, regardless of the estimated bandwidth conditions, if the amount of buffered data drops below the amount needed to mitigate jitter and short time scale bandwidth variations, i.e. if $Y_b(t) < r_b \cdot \Delta$, where Δ is the initial playback delay in seconds.

A key issue in the design of the adaptation mechanism for adding/dropping layers is determining which portion of the enhancement layer to transmit once the layer has been added. In our first implementation of adding/dropping layers, the server transmits the enhancement layer data with the same playback deadline as the base-layer currently being transmitted. In other words, the base and enhanced portions of each future frame are transmitted together. As a result, already prefetched base-layer data are consumed without enhancement.

4.2 Switching versions

We next develop a streaming control policy for switching versions. We denote the prefetch buffer contents in version v_1 at time t by $Y_1(t)$. Similarly, we denote the prefetch buffer contents in version v_2 at time t by $Y_2(t)$. As we shall see, to minimize the risk of buffer starvation, the server should not stream highest v_2, unless a reserve of future data exists in either of the two versions. We again suppose that the amount of buffered data at the client can be estimated at the server using regular receiver reports.

Figure 4 shows the state transition diagram for the switching versions policy. The server begins by streaming version v_1 at the available bandwidth rate. After the initial playback delay, the client begins to consume buffered data from version v_1. Whenever $X(t)$ exceeds r_1, future portions of version v_1 are prefetched into client storage. The amount of data buffered at the client in version v_1 indicates whether available bandwidth conditions allow streaming a higher-quality version. In a similar way as in the heuristic for adding/dropping layers, the server switches to version v_2 if two conditions hold. The first condition requires that the total amount of buffered data at the client is enough to avoid buffer starvation during the next C seconds. In particular, the server switches to v_2 at time s if:

$$\frac{Y_1(s)}{r_1} + \frac{Y_2(s)}{r_2} \geq C(1 - \frac{X_{\text{avg}}(s)}{r_2}). \tag{3}$$

The second condition for switching to v_2 at time s requires that $X_{\text{avg}}(s) \geq r_2$. Similar to the adding/dropping layers heuristic, this condition minimizes the quality fluctuations caused by frequently switching among the versions. The server continues streaming v_2 until, the likelihood of starvation becomes high, or until the amount of data buffered at the client falls below the required initial build-up. The server switches back to v_1 if (3) does not hold, or

if $\frac{Y_1(s)}{r_1} + \frac{Y_2(s)}{r_2} < \Delta$. To permit the fair comparison of adding/dropping layers and switching versions, $X_{avg}(s)$ is computed using the same estimation procedure (i.e. WEMA) as in the case of layers.

Similar to our implementation of adding/dropping layers, the server transmits data from v_2 beginning with the first video frame that has not yet been buffered in v_1. Already prefetched data from v_1 are decoded and displayed in their entirety before the consumption of version v_2 data begins.

4.3 Enhancing adding/dropping layers

In our proposed mechanism for adding/dropping layers in Section 4.1, the server streams both layers synchronously, i.e., streams the base and enhanced streams pertaining to the same portion of the video at the same moment. This implementation does not fully take advantage of the flexibility offered by layering. Indeed, when conditions are favorable enough to stream the enhancement layer, the server can enhance the base layer stream already prefetched in the client buffer, instead of simply enhancing the part of the base layer currently being sent. By transmitting the enhancement-layer data with the earliest playback deadline, the playback quality at the client can be enhanced immediately. We refer to this variation of the adding/dropping layers control policy as the *immediate enhancement* mechanism.

Immediate enhancement of the playback quality in the switching versions scheme can also be implemented. The server can switch to version v_2 by transmitting the v_2 data with the earliest playback deadline. We note, however, that in the case of versions, the immediate enhancement mechanism results in a waste of bandwidth; a portion of the prefetched v_1 data remains unused.

5 Numerical Results

We compare adding/dropping layers and switching versions in simulation experiments. The simulations implement the streaming control policies described in the previous section, and use the TCP traces described in Section 2 to simulate available bandwidth.

Table 2 summarizes results obtained with trace A1. The two schemes are compared based on the three performance metrics discussed in Section 3.2 when the percent bit-rate overhead associated with layering, H, varies between 1 and 10%. Metrics t_h, t_d and S are studied for different video consumption rates. Although, the video consumption rate is approximately constant for a certain quality video, we choose to vary the consumption rate in order to study the behavior of our heuristic policies under several bandwidth conditions. Parameter r_n is the ratio of the highest quality video consumption rate (i.e. $r_b + r_e$ for the case of layers, and r_2 for the case of versions) over the average trace bandwidth (i.e. 0.7 Mbps for trace A1). In Table 2, *Layers-imm* and *Versions-imm* represent the immediate enhancement mechanisms, for adding/dropping layers and switching versions respectively.

The results shown in the first two rows of Table 2 indicate that the performance of adding/dropping layers and switching versions are identical when $H = 0\%$. This can confirm that the adaptation mechanisms used in the two schemes are equivalent. When $H = 1\%$, the performance of adding/dropping layers deteriorates as bandwidth conditions become less favorable (i.e. for $r_n = 1.0$ and $r_n = 1.3$). In general, higher coding overhead results in further performance degradation of the adding/dropping layers scheme. The fraction of high quality viewing time t_h is lower in the case of layers than in the case of versions, when $H = 10\%$ under all r_n values. The fluctuations in quality (S) generally increase with H, although S remains

Scheme	$r_n = 0.7$			$r_n = 1.0$			$r_n = 1.3$		
	t_h	t_d	S	t_h	t_d	S	t_h	t_d	S
Versions	**98.42**%	0%	1	**85.12**%	0%	1	**52.68**%	0%	7
Layers $H = 0\%$	98.42%	0%	1	85.12%	0%	1	52.68%	0%	7
Layers $H = 1\%$	98.42%	0%	1	81.92%	0%	3	48.1%	0%	7
Layers $H = 5\%$	98.22%	0%	1	77.23%	0%	3	49.15%	0%	9
Layers $H = 10\%$	95.92%	0%	3	72.82%	0%	1	43.23%	0%	10
Layers-imm $H = 0\%$	99.03%	0%	3	**85.45**%	0%	19	**53.61**%	0%	37
Layers-imm $H = 1\%$	98.97%	0%	3	83.68%	0%	21	51.01%	0%	41
Layers-imm $H = 5\%$	**98.72**%	0%	3	79.93%	0%	21	49.15%	0%	49
Layers-imm $H = 10\%$	97.25%	0%	5	75.99%	0%	23	44.38%	0%	55
Versions-imm	97.64%	0%	3	76.42%	0%	13	34.41%	2.2%	26

Table 2: Results for trace A1

Scheme	$r_n = 0.7$			$r_n = 1.0$			$r_n = 1.3$		
	t_h	t_d	S	t_h	t_d	S	t_h	t_d	S
Versions	**94.31**%	0%	5	**57.66**%	0%	5	**43.78**%	0.4%	5
Layers-imm $H = 0\%$	95.56%	0%	9	62.36%	0%	21	44.41%	0.4%	19
Layers-imm $H = 1\%$	**95.06**%	0%	11	62.07%	0%	21	**43.94**%	0.4%	21
Layers-imm $H = 5\%$	91.71%	0%	17	**60.34**%	0%	25	41.27%	0.4%	25
Layers-imm $H = 10\%$	88.22%	0%	27	57.23%	0%	25	38.67%	0.4%	31

Table 3: Results for trace A2

reasonably low in all cases. Finally, we note that t_d is in all cases equal to 0, indicating that the video is delivered to the client without interruption.

We next consider the performance of adding/dropping layers and switching versions when the immediate enhancement mechanism in employed. Results in Table 2 indicate that with no overhead ($H = 0\%$), adding/dropping layers performs slightly better than switching versions. When $r_n = 0.7$, adding/dropping layers with immediate enhancement attains higher t_h than switching versions (regardless of whether immediate enhancement is employed in the case of versions), for coding overhead values as high as 5%. Under more adverse bandwidth conditions (i.e. $r_n = 1.0$ and $r_n = 1.3$), the immediate enhancement mechanism in the case of layers does not offer sufficient improvement in performance in the presence of coding overhead. When bandwidth conditions are scarce, the amount of data buffered at the client at any time is small. As a result the immediate enhancement scheme is overly aggressive, resulting in high S and no significant improvement in t_h.

Our results also demonstrate the inefficiency of employing the immediate enhancement mechanism in the case of versions. Employing the mechanism actually results in both lower t_h and higher S. As discussed earlier in Section 4.3, immediate improvement in playback quality results in a waste of bandwidth in the case of versions. The utilization of available resources is less efficient resulting in decreased overall performance.

Table 3 shows results obtained with trace A2. We observe that the adding/dropping layers scheme employing the immediate enhancement mechanism can result in higher t_h than switch-

ing versions, even in the presence of coding overhead. This is true for coding overheads up to 1% when $r_n = 0.7$ or $r_n = 1.3$, and up to 5% when $r_n = 1.0$. Again, for all values of r_n, increased performance in terms of t_h is attained at the expense of quality fluctuations.

6 Conclusion

In this paper we have designed equivalent streaming policies for adding/dropping layers and for switching among versions, in order to compare the two adaptation schemes under different critical conditions. Our results showed that our heuristics for both schemes successfully adapt the streaming to exploit the available TCP-friendly bandwidth. But, in the simplest implementation, the overhead introduced by layering makes switching versions always perform better than adding/dropping layers. However, when using the immediate enhancement implementation for adding/dropping layers, and depending on the particular bandwidth behavior, neither scheme seems to dominate : the enhanced flexibility provided by layering can compensate the loss in high quality viewing time that comes from the layering overhead, but at the expense of more fluctuations in quality. Since adding/dropping layers seems more proper to caching than switching versions, we advocate to use adding/dropping layers rather than switching versions. In future research, we intend to compare precisely the performance of both methods in the context of caching.

References

[1] 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.

[2] 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, Aug. 1999.

[3] X. et al. Li. Layered Video Multicast with Retransmission (LVMR): Evaluation of Error Recovery Schemes. In G. Parulkar, editor, *NOSSDAV '97*. Springer, Berlin, May 1997.

[4] J. Mahdavi and S. Floyd. TCP-Friendly Unicast Rate-Based Flow Control. Technical report, Jan. 1997.

[5] S. McCanne, V. Jacobson, and M. Vetterli. Receiver-driven Layered Multicast. In *Proc. ACM SIGCOMM*, Stanford, CA, Aug. 1996.

[6] J. Padhye, J. Kurose, D. Towsley, and R. Koodli. A model based tcp-friendly rate control protocol. In *IEEE NOSSDAV'99*, Basking Ridge, NJ, June 1999.

[7] R. Rejaie, D. Estrin, and M. Handley. Quality Adaptation for Congestion Controlled Video Playback over the Internet. In *Proc. of ACM SIGCOMM '99*, Cambridge, Sept. 1999.

[8] 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*, New York, Mar. 1999.

[9] D. Saparilla and K. W. Ross. Optimal Streaming of Layered Video. In *Proceedings of IEEE Infocom*, Tel Aviv, Israel, March 2000.

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

[11] D. Sisalem and H. Schulzrinne. The Loss-Delay Based Adjustment Alogrithm: A TCP-Friendly Adaptation Scheme. In *NOSSDAV*, 1998.

[12] T. Turletti, S. Parisis, and J. Bolot. Experiments with a layered transmission scheme over the Internet. Technical Report RR-3296, INRIA, France, 1997.

[13] L. Vicisano, L. Rizzo, and Crowcroft J. TCP-like congestion control for layered multicast data transfer. In *INFOCOM*, 1998.

[14] Y. Wang and Q. Zhu. Error Control and Concealment for Video Communications: A Review. In *Proc. of the IEEE*, volume 86, pages 974 – 997. May 1998.