# Providing Smoother Quality Layered Video Stream

*Srihari Nelakuditi Raja R Harinath Ewa Kusmierek Zhi-Li Zhang*
University of Minnesota, Minneapolis
{srihari,harinath,kusmiere,zhzhang}@cs.umn.edu

## I. Introduction

In recent years, one of the most popular Internet applications is web-based audio and video playback, where stored video is streamed from the server to a client on-demand. Rigid playback deadlines coupled with constraints on resources such as network bandwidth and client buffer make video delivery a challenging task [2]. These resources could be limited in such a way that it may not be possible to deliver full-quality video. In such a situation, it is desirable to minimize the degradation in the video quality while operating within the resource constraints [9]. Layered encoding is proposed to provide finer control on video quality: the video signal is split into layers and a subset of these layers is chosen such that the resource constraints are met [5]. However it is not a trivial task to select layers such that better but consistent quality playback is ensured when the network conditions are constantly varying.

In our work, we address this layer selection problem in layered video delivery and show how smoother[1] quality video playback can be provided by utilizing the client buffer for prefetching. We first define smoothness criteria, design metrics for measuring it, and then develop off-line algorithms to maximize smoothness for the case where the network bandwidth is varying but known *a priori*. We then describe an adaptive algorithm for providing smoothed layered video delivery that doesn't assume any knowledge about future bandwidth availability. The results of our experiments for measuring and comparing the performance these schemes are then presented. We conclude the paper with a brief discussion on our future work.

## II. Smoothness Criteria and Quality Metrics

One of the problems in assessing the performance of a video delivery scheme is the lack of a good metric that captures the user's perception of video quality. In general, the higher the amount of detail in the played video, the better is its quality. However, it is generally agreed that it is visually more pleasing to watch a video with consistent, albeit lower, quality than one with highly varying quality. Thus, a good metric should capture both the amount of detail per frame as well as its uniformity across frames.

Consider the example sequences of layers in a video stream shown in Figure 1. The top two streams consume the same amount of network resources, as the bottom two streams do. However, the sequences on the right are "smoother" than the ones on the left. In the first case, the degradation in quality is more gradual in the "smoother" sequence. In the second case the "smoother" sequence has fewer changes in quality. These smoothness criteria can be captured in a metric by giving higher weight to lower layers and to longer runs[2] of continuous frames in a layer.

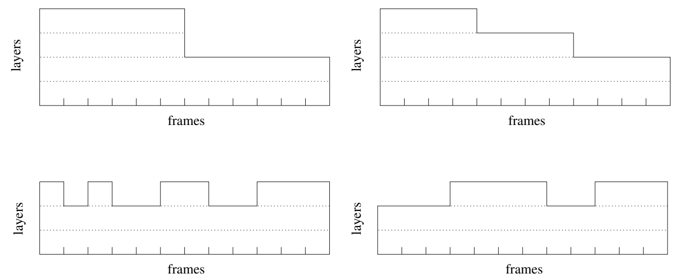



Fig. 1. Illustration of smoothness criteria

We propose a family of metrics that attempt to measure the smoothness in the perceived quality of a layered video. They represent the smoothness, $M$, of a video stream by a vector of the form $M = (m_1, m_2, \ldots, m_L)$, where $L$ is the total number of layers and $m_i$ is the measure associated with layer $i$. Using these metrics, two video streams can be compared by using lexicographic ordering of the corresponding vectors. A stream with smoothness measure $M^1$ is considered better than a stream with $M^2$ if $\exists i, m^1_j = m^2_j, j < i$ and $m^1_i > m^2_i$. In other words, the stream with a higher measure at a lower layer is considered smoother.

Each metric in the family computes its $m_i$ differently. We consider three such metrics in our work: *avgrun*, *minrun*, *exprun*. The *avgrun* metric measures the mean length of a run in layer $i$ and *minrun* its minimum run. The *exprun* metric measures the expected run length in layer $i$. In other words, the *exprun* metric gives the run length in layer $i$ that can be expected to be seen around an arbitrary frame in the layer.

These metrics can be computed as follows:

$$\begin{aligned} avgrun &= \tfrac{n_1+n_2+\ldots+n_k}{k} \\ minrun &= \min\{n_1, n_2, \ldots, n_k\} \\ exprun &= \tfrac{n_1^2+n_2^2+\ldots+n_k^2}{N} \end{aligned}$$

where $k$ is the total number of runs in a layer and $n_1, n_2, \ldots, n_k$ are the lengths of these runs. These values are then normalized by the length of the video sequence and presented as a value between 0 and 1.

The following table lists the smoothness of each sequence in Figure 1, as measured by each of the above metrics. It is assumed that the length of the video is 12 frames.

[1] Here smoothing refers to the perceived video quality, not bandwidth.

[2] Hereafter *run* refers to a sequence of consecutive frames shown in a layer.

| | top left | top right |
|---|---|---|
| *avgrun* | $(1.00, 1.00, 0.50, 0.50)$ | $(1.00, 1.00, 0.67, 0.33)$ |
| *minrun* | $(1.00, 1.00, 0.50, 0.50)$ | $(1.00, 1.00, 0.67, 0.33)$ |
| *exprun* | $(1.00, 1.00, 0.25, 0.25)$ | $(1.00, 1.00, 0.44, 0.11)$ |
| | bottom left | bottom right |
| *avgrun* | $(1.00, 1.00, 0.15)$ | $(1.00, 1.00, 0.30)$ |
| *minrun* | $(1.00, 1.00, 0.08)$ | $(1.00, 1.00, 0.25)$ |
| *exprun* | $(1.00, 1.00, 0.10)$ | $(1.00, 1.00, 0.17)$ |

Consider for example the bottom left sequence in Figure 1. The total number of runs is $1, 1, 4$ respectively for the layers $1, 2, 3$. For layers 1 and 2 the average run length and minimum run length is same as the length of the video. Hence their normalized measures are each 1.0. The layer 3 has 4 runs of length $1, 1, 2, 3$ respectively. Thus the average run length is 1.75 and the corresponding *avgrun* measure is $\frac{1.75}{12} = 0.145$. Similarly the minimum run length is 1 and the *minrun* measure is $\frac{1}{12} = 0.083$. The expected run length of layer 3 is $\frac{1^2+1^2+2^2+3^2}{12} = 1.25$ and hence the corresponding *exprun* measure is $\frac{1.25}{12} = 0.104$. As can be seen, the relative smoothness of the sequences in Figure1 is reflected by each of the metrics in the above table.

The metrics *avgrun* and *minrun* are easy to understand and each measure a different statistic on the runs present. But they fail to take into account the absence of runs. For example, given a sequence of runs of a layer, by dropping all the runs except the longest run we can generate a new sequence with larger *avgrun* and *minrun* values. Such a new sequence may not necessarily be perceptually better than the original sequence. In order to address this we also need to consider the absence of a frames in a run. The *exprun* metric captures this notion by taking both the sum of all runs and the length of each individual run into account.

We now proceed to formulate the layer selection problem in video delivery and develop algorithms that choose layers such that the smoothness of the resulting sequence is maximized as measured by these metrics.

## III. Problem Formulation

The objective of a layer selection scheme is to optimize the perceived video quality, as measured by metrics described earlier, given the resource constraints. In formulating this problem, we consider a discrete-time model at the frame level. Each time slot represents the unit of time for playing back a video frame. For simplicity of exposition, we assume startup delay of one slot, i.e., the server starts video transmission one time slot ahead of the time the client starts playback. In other words, server starts transmission at time 0 and the client starts displaying the frame 1 at time 1. We also ignore the network delay. Table I summarizes the notation we introduce in this section.

TABLE I
NOTATION

| | | |
|---|---|---|
| $N$ | : | length of video in frames |
| $L$ | : | number of layers of video |
| $f_i^j$ | : | size of $j^{th}$ layer of $i^{th}$ frame |
| $B_i^j$ | : | buffer occupancy by layer $j$ at slot $i$ |
| $\hat{B}_i^j$ | : | buffer constraint during slot $i$ layer $j$ |
| $\hat{C}_i^j$ | : | bandwidth constraint during slot $i$ layer $j$ |
| $S$ | : | a layer sequence |
| $\hat{S}$ | : | an optimal feasible layer sequence |
| $A(S)$ | : | a transmission schedule w.r.t. sequence S |
| $A_i(S)$ | : | cumulative data sent by server over $[1, i]$ |
| $a_i(S)$ | : | amount of data sent by server in slot $i$ |
| $D(S)$ | : | underflow curve w.r.t sequence S |
| $D_i(S)$ | : | cumulative data consumed by the client over $[1, i]$ |
| $U(S)$ | : | overflow curve w.r.t sequence S |
| $U_i(S)$ | : | maximum cumulative data that can be received by the client over $[1, i]$ |

Consider a video stream with $N$ frames and $L$ layers. The size of $j^{th}$ layer of $i^{th}$ frame is denoted by $f_i^j$. Let $\hat{C}^j = (\hat{C}_0^j, \hat{C}_1^j, \ldots, \hat{C}_N^j)$ and $\hat{B}^j = (\hat{B}_0^j, \hat{B}_1^j, \ldots, \hat{B}_N^j)$, where $\hat{C}_i^j$ denote network bandwidth and $\hat{B}_i^j$ client buffer capacity during time slot $i$ at a layer $j$. Let $S = (S_1, S_2, \ldots, S_N), 0 \leq S_i \leq L$ be a layer sequence.

Let $D(S) = (D_0(S), D_1(S), \ldots, D_N(S))$ where $D_i(S) = \sum_{k=0}^{i} \sum_{j=0}^{S_i} f_k^j$, and let $U(S) = (U_0(S), U_1(S), \ldots, U_N(S))$ where $U_i(S) = D_i(S) + \hat{B}_i^1$. We refer to $D(S)$ as the *(client) buffer underflow curve with respect to* $S$, and $U(S)$ as the *(client) buffer overflow curve with respect to* $S$. A server transmission schedule $A(S)$ associated with $S$ is a schedule which only transmits layers of frames included in $S$, namely, layer $j$ of frame $i$ is transmitted under $A(S)$ if and only if $j \leq S_i$. Let $a_i(S)$ be the amount of video data transmitted during time slot $i$, $i = 1, \ldots, N$[3]. The schedule $A(S)$ is denoted by $A(S) = (A_0(S), A_1(S), \ldots, A_N(S))$ where $A_0(S) = 0$ and $A_i(S) = \sum_{k=0}^{i} a_k(S)$.

A server transmission schedule $A(S)$ is said to be *feasible* with respect to $S$ if and only if for $i = 0, 1, \ldots, N$, 1) *rate constraint is not violated*, i.e., $a_i(S) \leq C_i^1$; 2) *buffer constraint is not violated*, i.e., $A_i(S) \leq U_i(S)$; and 3) *playback constraints are not violated*, i.e, $D_i(S) \leq A_i(S)$. For a given rate and buffer constraints $(\hat{C}, \hat{B})$, we denote the collection of all feasible layer sequences by $\mathcal{FLS}(\hat{C}, \hat{B})$.

Now the optimal layer selection problem can be stated as follows: find a feasible sequence $\hat{S}$ that maximizes the associated metric *quality*($\hat{S}$), formally

*Find a sequence* $\hat{S}$ *such that* $\hat{S} \in \mathcal{FLS}(\hat{C}, \hat{B})$ *and* $quality(\hat{S}) = \max\{quality(S) : S \in \mathcal{FLS}(\hat{C}, \hat{B})\}$.

## IV. Optimal Layer Selection

In this section, we discuss the potential approaches to designing optimal layer selection algorithms for maximizing each of the metrics defined earlier. We first make some simplifying assumptions about the problem setting. We then describe a generic layer selection procedure which uses a metric-specific procedure for selecting frames within a layer. The frame selection procedures for these metrics namely, MAXAVGRUN, MAXMINRUN, and MAXEXPRUN are then presented.

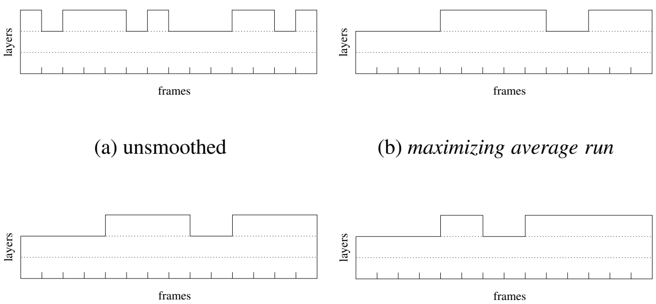
We assume that each layer in the video is of CBR, i.e., all frames in a layer are of the same size. This enables us to maximize the given metric for each layer in isolation. For simplicity of exposition, we further assume that all layers of equal bit rate[4]. Now, without loss of generality, all the units can be scaled such

[3] When optimizing *avgrun* and *minrun* metrics, to account for the absence of runs, we need to add an additional constraint that a transmission schedule has to be work-conserving, i.e., $a_i(S) = min(\hat{B}_i^1 - \sum_{j=1}^{L} B_i^j, \hat{C}_i^1)$

[4] Note that the algorithms described here do not hinge on this assumption

PROCEDURE OPTLAYERS($\hat{C}$, $\hat{B}$)<br>
  Initialization : $\hat{C}^1 = \hat{C}, \hat{B}^1 = \hat{B}$<br>
  For j = 1 to L<br>
    $(\hat{C}^{j+1}, \hat{B}^{j+1}, \hat{S}^j) =$ *maxanyrun*$(\hat{C}^j, \hat{B}^j)$<br>
END PROCEDURE

Fig. 2. The generic optimal layer selection procedure




(a) unsmoothed (b) *maximizing average run*

(c) *maximizing minimum run* (d) *maximizing expected run*

Fig. 3. Optimal layer sequences: (a) buffer=0; (b)(c)(d) buffer=3

1. PROCEDURE MAXAVGRUN($\hat{C}^j$, $\hat{B}^j$)<br>
2. Initialization : $B_0^j = 0, \Theta = 0$<br>
3. For i = 1 to N<br>
4. Update buffer: $B_i^j = B_{i-1}^j + \hat{C}_i^j - \Theta$<br>
5. If $B_i^j > \hat{B}_i^j$, i.e., buffer overflow?<br>
6. Select phase: $\Theta = 1; B_i^j = \hat{B}_i^j$<br>
7. Else If $B_i^j < 0$, i.e., buffer underflow?<br>
8. Discard phase: $\Theta = 0$; $B_i^j = 0$<br>
9. Else<br>
10. Continue same phase<br>
11. Note frame status: $\hat{S}_i^j = \Theta$<br>
12.<br>
13. Initialization : $\check{B}_N^j = 0$<br>
14. For i = N to 1<br>
15. If $B_i^j > \check{B}_i^j$, i.e., buffered enough?<br>
16. Select frame: $\hat{S}_i^j = 1$<br>
17. Residual buffer limit: $\hat{B}_i^{j+1} = \hat{B}_i^j - \check{B}_i^j$<br>
18. If $\hat{C}_i^j > \check{B}_i^j + \hat{S}_i^j$, i.e., bandwidth enough?<br>
19. Residual bandwidth: $\hat{C}_i^{j+1} = \hat{C}_i^j - \check{B}_i^j - \hat{S}_i^j$<br>
20. Buffer not needed for future: $\check{B}_{i-1}^j = 0$<br>
21. Else<br>
22. Adjust future buffer: $\check{B}_{i-1}^j = \check{B}_i^j - \hat{C}_i^j + \hat{S}_i^j$<br>
23. No residual bandwidth: $\hat{C}_i^{j+1} = 0$<br>
24. END PROCEDURE

Fig. 4. The algorithm for maximizing average run length

that size of each frame in a layer is 1, i.e., $f_i^j = 1$ and buffer and bandwidth values are all integers.

The generic optimal layer selection procedure shown in Figure 2 can be described as follows. Consider each layer $j$ from lower to higher starting with layer 1. Select an optimal subset, $\hat{S}^j$, as measured by the metric, of all frames at that layer $j$ treating it as the only layer of the video. A metric-specific procedure is used in place of *maxanyrun* to perform the selection of frames in a layer. This optimal selection has to satisfy the resource constraints given by the residual bandwidth, $\hat{C}^j$ and residual buffer, $\hat{B}^j$ that is available after considering all the layers up to $j-1$. Since it is always preferable, according to our metrics, to select lower layer frames and also because all frames are of equal size, with optimal selection of frames at each layer, the resulting layer sequence would also be optimal.

The Figure 3 shows an example unsmoothed sequence and the corresponding smoothed sequences that would result after applying the MAXAVGRUN, MAXMINRUN, and MAXEXPRUN algorithms. The maximum available buffer is assumed to be 3. While the unsmoothed sequence has 5 runs, all the smoothed sequences have only 2 runs and thus longer runs. However, the run lengths are different in each case so as to optimize the respective metrics. In the following subsections we explain these algorithms in detail.

### A. *Maximizing the average run length*

The average run length in a sequence can be maximized by minimizing the number of runs while keeping the sum of all the runs as high as possible. We propose MAXAVGRUN algorithm that achieves this by delaying the start of a run as late as possible and stretching its end as far as possible. Intuitively, a new run is not initiated unless the buffer is accumulated adequately and it is not terminated until the buffer is drained completely.

The Figure 4 shows the details of this algorithm operating at a layer $j$. It consists of two scans across the length of the video: one in forward direction (lines 2-11) and one in backward direction (lines 13-23). The forward scan can be viewed as a step that identifies the end of each run and the minimal number of runs. The backward scan essentially extends each run towards the front of it while at the same time maximizing the residual buffer made available to higher layers.

The algorithm during a forward scan switches between *select phase* in which frames are selected (line 6) and *discard phase* in which frames are discarded (line 8). It starts with empty buffer and in discard phase (line 2). In each slot, the buffer occupancy, $B_i^j$ is updated (line 4) such that bandwidth constraint is not violated. It enters select phase if the buffer is full even after consuming a frame (line 5) and switches to discard phase if the buffer is empty even before consuming a frame (line 7). If the buffer occupancy stays within the bounds, the same phase is continued (lines 9-10). The current frame is either selected or discarded based on the current phase (line 11).

At the end of a forward scan it is possible that accumulated buffer is not completely drained. Furthermore, before each run the buffer may be filled up too early and hence rendered unusable by higher layers. The backward scan addresses these concerns by accumulating the buffer as late as possible and only when needed. It keeps track of the buffer requirement at a frame $i$, $\check{B}_i^j$, for the selected frames beyond $i$ in future. Since no buffer is required beyond frame $N$, it is initialized to 0 (line 13). The current frame is selected whenever the current buffer occupancy is more than the future buffer requirement (lines 15-16). The residual buffer available for layer $j+1$ is set by subtracting the buffer required for layer $j$ from the the total amount available to it (line 17). If enough bandwidth is available at slot $i$ to satisfy the future buffer requirements of layer $j$, then some residual bandwidth is made available to layer $j+1$ and the buffer requirement beyond slot $i-1$ is also adjusted accordingly (lines 18-23). Thus by filling the buffer closer to where it is consumed, larger

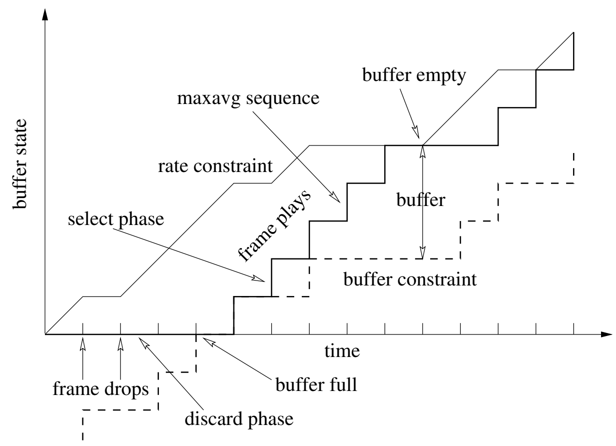



Fig. 5. Illustration of MAXAVGRUN sequence

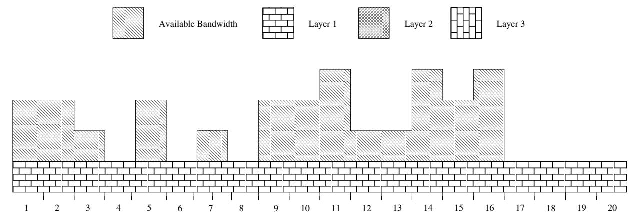
residual buffer is made available to higher layers and longer runs are made possible at each layer.

Figure 5 shows the MAXAVGRUN algorithm in operation at layer 3 on the example unsmoothed sequence in Figure 3(a) while generating the smoothed sequence in Figure 3(b). The top curve represents the rate constraint corresponding to the residual bandwidth available at layer 3 as given by unsmoothed sequence. The bottom curve gives the buffer constraint which is essentially the rate constraint shifted down by the buffer size which is 3 in this example. It is assumed that data transmitted in a slot is added to the buffer only at the end of the slot. The middle curve is the consumption curve corresponding to the frames selected by the MAXAVGRUN procedure. The crossover of consumption curve and the buffer constraint curve implies a buffer overflow while that of consumption and rate constraint curves means a buffer underflow.

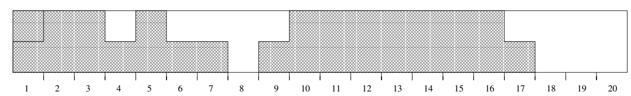
The MAXAVGRUN algorithm, as described earlier, starts in discard phase and continues dropping frames till the $5^{th}$ frame where upon it enters select phase. Otherwise, i.e, if we had dropped frame 5 also, it would have caused a buffer overflow. Once in select phase it continues selecting frames up to 9 at which point it is forced to switch to discard phase lest buffer would underflow. It then selects another run of frames 12-14, even before the overflow point, during the backward scan.

We now illustrate the operation of the MAXAVGRUN algorithm using an example. Consider a layered video clip of length 20 frames with 3 layers. Let us assume that client can buffer up to 2 units of the video. Suppose that the curve given in Figure 6(a) represents the available bandwidth in each frame slot. The bandwidth used to transmit layer 1 and the residual bandwidth after processing layer 1 are marked differently in the figure. Since the available bandwidth in each slot is greater than 1, the layer 1 can be delivered completely without using any buffer for prefetching.

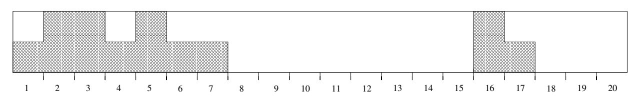
Figure 6(b) shows the buffer occupancy in each frame slot as computed during the forward scan at layer 2. In slot 1, it uses the 2 units of bandwidth available for building the buffer for layer 2. In slot 2, since the buffer is full and bandwidth is available, it initiates a run and starts selecting frames. The run is terminated at slot 18 beyond which it can not be continued due to lack of bandwidth and buffer size limitation. In this scan, the buffer is



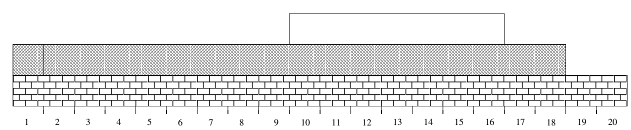

(a) available bandwidth for layer 2



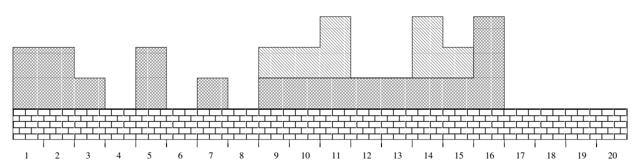
(b) buffer occupancy after forward scan at layer 2



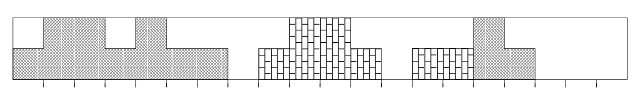
(c) buffer occupancy after backward scan at layer 2



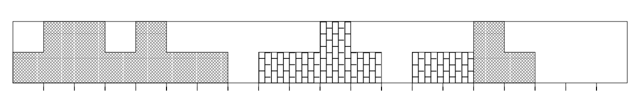
(d) selected frames up to layer 2



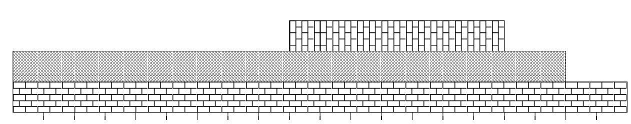
(e) available bandwidth for layer 3



(f) buffer occupancy after forward scan at layer 3



(g) buffer occupancy after backward scan at layer 3



(h) selected frames up to layer 3

Fig. 6. Illustration of the operation of MAXAVGRUN algorithm

filled whenever additional bandwidth available and kept as full as possible.

There are two improvements possible to the sequence generated by forward scan. First, the length of each run may be extended at the front since in forward scan buffer might have been built up earlier than necessary. In this example, the run of layer 2 can be started at slot 1 itself instead if at slot 2. Second, larger residual buffer can be made available to higher layers by buffering frames of a layer only when needed. The layer 2 requires 2 units of buffer only at the end of slot 16 which can be built up in that slot instead of building it too early by slot 10. This would increase the residual buffer for layer 3. This is precisely what is done during a backward scan. The buffer occupancy and the selected layer sequence after the backward scan on layer 3 are shown in Figures 6(c) and 6(d) respectively. The bandwidth utilized by layer 2 and the residual bandwidth for layer 3 are shown in Figure 6(e).

Figures 6(f) and 6(g) give the buffer occupancy at the end of forward scan and backward scan respectively on layer 3. Once again the run of layer 3 is extended at the front during the backward scan. It can also be seen that by making larger amount of residual buffer available to layer 3, its run is continued for longer period. Otherwise, without the backward scan on layer 2, layer 3 would have two shorter runs of length 1 and 3 instead of a single run of length 7. The final resulting layer sequence given by MAXAVGRUN algorithm is shown in Figure 6(e).

It is quite obvious from the description and illustration that this algorithm ensures that neither buffer nor bandwidth constraint is violated at each slot. Hence it selects only feasible subset of frames. Furthermore, since during the forward scan a run is started only after the buffer is full and it is ended only when the buffer is empty, the length of is each run (except the very last run) is guaranteed to be at least buffer size. The backward scan attempts to further extend the length of each run. It can be easily shown that MAXAVGRUN algorithm minimizes the number of runs and hence maximizes the average run length.

### B. Maximizing the minimum run length

The algorithm described above yields the minimum number of runs with the length of each run being at least the size of buffer. However there could be some variation in relative lengths of these runs. One way to maximize the minimum run length would be to reduce the variance among the runs. In other words, we can pull up the minimum run length in the overall sequence by growing the shorter runs while shrinking their longer neighboring runs.

Given a MAXAVGRUN sequence, it is not possible to grow a run at the end since the buffer is empty at that point. However a run can be made longer by selecting a chunk of frames just before the run and discarding an equal number of frames from the end of a previous run. The extent of growth is limited obviously by the maximum buffer and also the buffer accumulation pattern right before the run. It should be noted that though a sequence with higher than the minimum number of runs may also have the largest minimum run length, it is possible to find a sequence that has the same minimum run length but with lesser number of runs. So starting with a MAXAVGRUN sequence we can find a MAXMINRUN sequence by readjusting the lengths of each run without increasing the number of runs.

The MAXMINRUN algorithm first applies the MAXAVGRUN algorithm on the unsmoothed sequence to generate a sequence that has the minimum number of runs. It then considers each pair of consecutive runs in order and tries to bring their lengths as close as possible. Lets say their lengths are $n_k$ and $n_{k+1}$. Also, let $x_{k+1}$ is the limit on the length by which run $k+1$ can be grown. If $n_k \leq n_{k+1}$, continue with the next pair. Otherwise select $min((n_k - n_{k+1} + 1)/2, x_{i+1})$ number of frames before the beginning of run $i + 1$ and discard same number of frames from the end of run $k$. Accordingly adjust the counter $x_{k+1}$. We are not done with just one iteration. This procedure is somewhat similar to bubble sort. It proceeds with another iteration if there was any change to any of the runs in the previous iteration. Otherwise the procedure terminates.

Figure 3(c) shows the resulting MAXMINRUN sequence for a simple case of unsmoothed sequence given in Figure 3(a) and the corresponding MAXAVGRUN sequence in Figure 3(b). By applying the above procedure the minimum run length at layer 3 is increased from 1 in unsmoothed sequence, 3 in MAXAVGRUN sequence to 4 in MAXMINRUN sequence. While this heuristic algorithm works towards maximizing the minimum run length, further investigation is needed to develop a provably optimal algorithm.

### C. Maximizing the expected run length

It is quite clear that the longer the runs are in a sequence, the higher it's expected run length is. So in order to maximize the expected run length we need to make each run as long as possible. Furthermore, extension of a longer run contributes more towards the expected run length than that of a shorter run. Hence, a reasonable heuristic approach for maximizing the expected run length is to start with a MAXAVGRUN sequence and to make the longer sequences even longer while further shortening the shorter runs.

The MAXEXPRUN algorithm first applies the MAXAVGRUN algorithm. It then considers each pair of consecutive runs in order and tries to grow the longer run even more. Let the length of two consecutive runs be $n_k$ and $n_{k+1}$. Also, let $x_{k+1}$ is the limit on the length by which run $k + 1$ can be grown. If $n_k - x_{k+1} > n_{k+1} + x_{k+1}$, continue with the next pair. Otherwise select $x_{i+1}$ number of frames before the beginning of run $i + 1$ and discard same number of frames from the end of run $k$. Unlike in MAXMINRUN case, MAXEXPRUN terminates with just one scan over each of the runs given by MAXAVGRUN. Figure 3(d) shows the sequence with maximum expected run length corresponding to the unsmoothed one in Figure 3(a).

## V. Adaptive Layer Selection

The algorithms discussed so far assume that the bandwidth availability for the entire duration of the video is known *a priori*. Based on the insights gained from these algorithms, in this section we develop an adaptive scheme for layer selection that assumes no knowledge about future bandwidth availability. However, we assume the presence of a bandwidth estimator that gives the precise current bandwidth in each frame slot.

The key questions that needs to be addressed by any layer selection scheme for smoother quality are: 1) which layers and

2) which frames of a layer be prefetched; when to initiate a run for a layer and where to position the run. The *off-line*[5] algorithm addresses these issues by taking advantage of the complete knowledge about future bandwidth availability. With the aid of a forward scan and a backward scan for each layer, it prefetches the frames of layers from lower to higher as late as possible and only when needed. But when future availability of bandwidth is not known, the decision on which layer and frame to transmit next has to be made in an online fashion. In such a case it is not possible to precisely compute the buffer needed for each layer and prefetch just in time. In the following we present an *adaptive algorithm* to predict the buffer requirements of a layer to tide over the potential future bandwidth droughts and to position the runs of a layer to accumulate sufficient cushion before it is displayed.

The adaptive algorithm attempts to estimate future bandwidth availability based on the past history. It associates a *target buffer cushion* ($\tilde{B}^j$) with each layer $j$ and adjusts this value dynamically based on the bandwidth availability. The target buffer cushion of a layer $j$ corresponds to the number of frames of layer $j$ that need to be prefetched in order to continue a run at layer $j$ under the current bandwidth conditions. If the amount of cushion is too less, even small dips in available bandwidth could cause discontinuity of a run at this layer. If it is too much, this may result in inefficient use of buffer which otherwise could help cushion other layers. In the case of off-line algorithm, the forward scan ensures that the prefetched amount is not too less and the backward scan ensures that it is not too much. In the absence of knowledge about future bandwidth availability, the key task of an adaptive algorithm is to estimate the minimum amount of buffer cushion that is sufficient for a layer.

A run of layer $j$ can be sustained only if the average available bandwidth is greater than $j$ and the amount of buffer available for layer $j$ is sufficient to cushion the variations in bandwidth. It is possible to estimate the buffer requirements for an uninterrupted run of a layer $j$ by keeping track of how often the available bandwidth goes below $j$ and how long it stays below $j$. By monitoring the fluctuations in available bandwidth, we can identify a *critical layer* ($\hat{j}$), i.e., the highest layer that can be subscribed but may not have sufficient buffer available to cushion the bandwidth fluctuations. Given a critical layer $\hat{j}$, ideally we would like to protect the runs of all the lower layers up to $\hat{j}-1$ from bandwidth droughts and extend the run at layer $\hat{j}$ as long as possible. We may not initiate a run at layer $\hat{j}$ unless it can be sustained for a certain minimum period. In the following, we present a simple adaptive algorithm that addresses these issues in an indirect way and adjusts the target buffer cushion for each layer by tracking only buffer usage.

The adaptive algorithm dynamically adds layers and allocates the buffer among them based on how buffer is built and drained. The target buffer cushion for a layer is increased if the current target cushion was found to be inadequate to avoid discontinuity in the run due to a bandwidth drought period. It is decreased if the current target buffer cushion was filled and remained undrained for a certain observation window period. Figure 7 shows the procedure for adjusting cushion at slot $i$.

[5]Hereafter we refer to the MAXAVGRUN algorithm described in the previous section as the off-line algorithm.

1. `PROCEDURE` ADJUST-CUSHION($i$)
2.   `If` $B_k = \hat{B}, i - \tau + 1 \leq k \leq \tau$
3.     `For` $j = 1$ `to` $\hat{j}$
4.       $\tilde{B}^j = \beta \tilde{B}^j$, i.e., decrease target cushion
5.       $X^j = i$
6.
7.   `For` $j = 1$ `to` $\hat{j}$
8.     `If` $B^j_i < \delta \tilde{B}^j$ and $B^j_k >= \tilde{B}^j$ for some $k$, $X^j < k < i$
9.       $\tilde{B}^j = \alpha \tilde{B}^j$, i.e., increase target cushion
10.       $X^j = i$
11. `END PROCEDURE`

Fig. 7. The cushion adjustment at time $i$

1. `PROCEDURE` ADD-LAYER()
2.   `If` $B^j \leq \tilde{B}^j, 1 \leq j \leq \hat{j}$ and $\bar{C} \geq \hat{j} + 1$
3.     `If` $\tilde{B}^{\hat{j}+1} < \tilde{B}^{\hat{j}}$
4.       $\tilde{B}^{\hat{j}+1} = \tilde{B}^{\hat{j}}$
5.     $\lambda^{\hat{j}+1} = i + \tilde{B}^{\hat{j}+1}$
6.     $\hat{j} = \hat{j} + 1$
7. `END PROCEDURE`

Fig. 8. The procedure to add a new layer

The target buffer cushion for all the active layers is decreased by $\beta$ if the buffer was full for a certain duration $\tau$ (lines 2-5). There are two reasons for doing this. First, the buffer being full for sustained period indicates that bandwidth is certainly sufficient to accommodate $\hat{j}$ layers and hence the lower layers can be protected with much less cushion. Second, it also implies that buffer is scarce and hence we need to use it more efficiently by prefetching less conservatively for lower layers. The target buffer cushion $\tilde{B}^j$ for an active layer $j$ is increased by $\alpha$ if the current target cushion was filled earlier after the last cushion update time $X^j$ and then gets drained below a threshold $\delta$ (lines 7-10). This way we react in advance for any onset of drought period by conservatively increasing the target cushion even before it is drained completely.

The adaptive algorithm starts a new run at an higher layer $\hat{j}+1$ only if there is sufficient cushion for all the lower layers up to $\hat{j}$ and sufficient bandwidth to accommodate one more layer. The procedure to determine whether to add a new layer and where to start the run is given in Figure 8. A new run at a layer $\hat{j}+1$ is initiated only if the target buffer cushion is already filled for all the layers up to $\hat{j}$ and the long term bandwidth is sufficient to accommodate layer $\hat{j}+1$ also (line 2). The long term mean bandwidth ($\bar{C}$) is measured by exponential averaging the current and past bandwidth values. The target buffer cushion for $\hat{j}+1$ is ensured to be at least as large as the layer below it (lines 3-4). This is because higher layers need more cushion to sustain a run than lower layers. The new run is positioned at frame $\lambda^{\hat{j}+1}$, i.e., a distance of $\tilde{B}^j$ away from the current frame slot $i$ (line 5). This is intended to allow sufficient time for the cushion buffer to be filled before the new layer gets played back at the client. The new layer $\hat{j}+1$ thus becomes the critical layer (line 6).

The adaptive scheme determines which layer to be transmitted next based on the target buffer cushion and the current buffer cushion values for each layer. The corresponding procedure is given in Figure 9. A frame of a layer $j'$ is transmitted only if the current prefetched amount for each layer up to $j'-1$ is greater than the corresponding target buffer cushion value (line 2-5). In

1. PROCEDURE NEXT-TO-XMIT()<br>
2.   For $j = 1$ to $\hat{j}$<br>
3.     If $B^j < \tilde{B}^j$<br>
4.       $j' = j$<br>
5.       Return<br>
6.<br>
7.   $B_{min} = \hat{B}$<br>
8.   For $j = 1$ to $\hat{j}$<br>
9.     If $B^j - \tilde{B}^j < B_{min}$<br>
10.       $B_{min} = B^j - \tilde{B}^j$<br>
11.       $j' = j$<br>
12. END PROCEDURE

Fig. 9. The procedure to determine the layer for transmission

other words the lowest layer with cushion less than its target is chosen for transmission. If all the active layers have their target cushions built up, one more layer may be added using the procedure described in Figure 8. When there is additional bandwidth available even after filling up target cushions of all the active layers, the layer with the least additional cushion above its target buffer cushion is chosen for transmission (lines 7-11). In other words the extra bandwidth is shared fairly between all the active layers.

## VI. Experimental Results

We conducted simulations to study the performance of off-line and adaptive algorithms described above. As mentioned in Section IV we assume that the video is CBR with linearly spaced layers, i.e., the size of all frames is the same and is scaled to be 1 unit. Correspondingly the client buffer and the network bandwidth are scaled to be integers. The video consists of 4 layers. The length of the sequence is 30000 frames. The playback rate is set to 30 frames/sec and hence the whole video lasts for a period of 1000 secs.

The bandwidth is varied such that the mean bandwidth was 3.5 during the first 10000 frame slots, 2.5 during the next 10000 slots and 4.5 during the last 10000 slots. The faster time scale variation around these mean values is modeled using a Markov chain. The resulting bandwidth curve used in our experiments is shown in 10(a).

We study the performance of off-line and adaptive schemes under the above bandwidth conditions by varying the amount of client buffer. The configurable parameters for the adaptive scheme were set as follows. The target cushion increase and decrease factors $\alpha$ and $\beta$ were set to $2.0$ and $0.75$ respectively. In other words, the target cushion is doubled whenever the current target was found to be too less and is decreased by a quarter if it was found to be too much. The threshold $\delta$ to trigger cushion increase was set to $0.5$, i.e., whenever the current buffer for a layer drains below half its target buffer, the target cushion is increased. The observation window period $\tau$ is set to 300, i.e., the target cushion values are decreased if the buffer was full for 10 secs. These values are set such that the adaptive scheme is conservative in protecting the lower layers by maintaining higher cushions.

Figure 10 compares the number of layers selected, and hence the corresponding smoothness achieved by the off-line and adaptive schemes. The output of the off-line scheme with a small client buffer of 30 is shown in Figure 10(b). We expect that this relatively unsmoothed stream is similar to one generated by a greedy layer selection scheme that adds a new layer as and when bandwidth is available. Clearly such a sequence with frequent transitions between layer levels is undesirable. Figures 10(c) and 10(d) show the layer selection by off-line and adaptive schemes respectively when the client buffer is 300. It is very reasonable to expect a buffer at the client that can accommodate up to 10 secs of one layer of the video. Even with not so large a buffer the off-line algorithm yields a considerably smoother sequence. Though the adaptive scheme generates a less smoother sequence than the off-line scheme, it is still significantly better than the result of a greedy selection scheme.

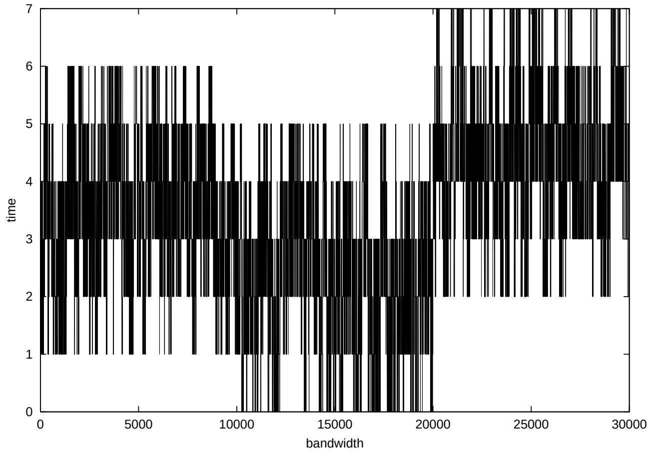
When the buffer is increased to a size of 900 that can accommodate 10 secs of 3 layers of the video, both the schemes produce much smoother sequences as shown in Figures 10(e) and 10(f). In particular, the output of the off-line algorithm is very smooth. This is because it has complete knowledge and utilizes the buffer in the most efficient manner. The output of adaptive scheme, while it resembles that of the off-line algorithm in terms of layer subscription level, is not as smooth: there is fluctuation about the critical layer. However, the increased buffer does improve the output sequence of the adaptive scheme. In particular, the segment between 10000 and 20000 is much smoother and displays more layers.

It is worth noting that the sequence in Figure 10(f) would appear much smoother if the fluctuations about the critical layer could be avoided. For example, consider the segment between frame slots 21000 and 25000. This could have been improved by not selecting layer 4 when its run cannot be sustained for a certain minimum duration. Currently the adaptive scheme, though it attempts to fill the cushion for a layer before the start of a run, still does not attempt to avoid potentially short runs. We need to further improve the algorithm to estimate the length of a run and initiate one only when it is likely not to be short-lived.

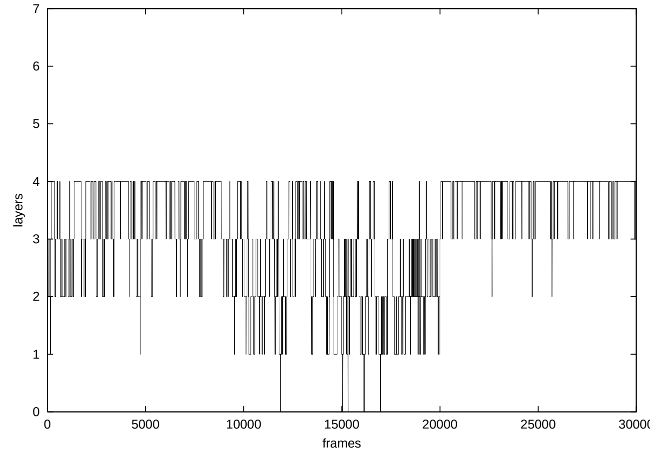
The off-line algorithm also exhibits a similar behavior in that it doesn't explicitly avoid short runs. For example, in Figure 10(c) at around frame slot 12500, layer 4 is chosen briefly. This would not contribute to quality viewing and discarding that run would make the sequence smoother. This behavior can be attributed to the work-conserving nature of the off-line algorithm: available bandwidth is always utilized and thus higher layers are selected momentarily when the buffer is already full. The algorithm needs to be modified to avoid shorter runs even by resorting to not being work-conserving. Further, we need to alter the *exprun* metric to discount runs shorter than a certain length.

We now describe the operation of the adaptive scheme given the bandwidth curve shown in Figure 10(a). Since the adaptive scheme is not aware of future bandwidth availability it has to dynamically adjust the target buffer cushion for each layer based on past history on the variability in available bandwidth. This cushion adaptation process is illustrated in Figure 11(a). It shows how the target buffer cushion is adjusted over time for the layers 1 and 2. It also gives the ideal amount of buffer required for these layers as computed by the off-line algorithm. Figure 11(b) shows the actual buffer occupied by layers 1 and 2 in the adaptive scheme. It also shows the total buffer occupied by all the layers.

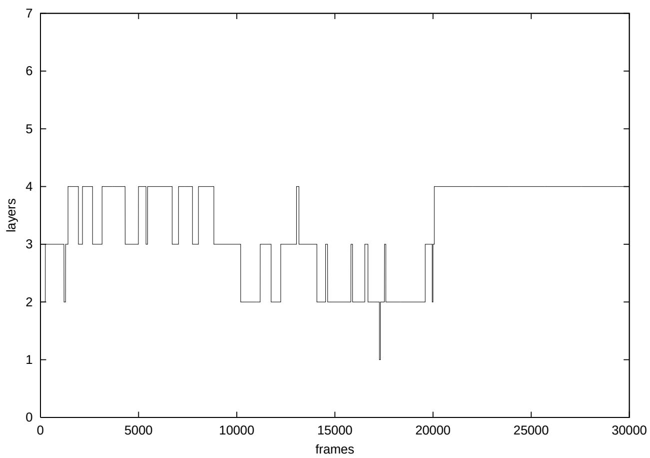
By contrasting the Figures 11(a) and 11(b) we can see the



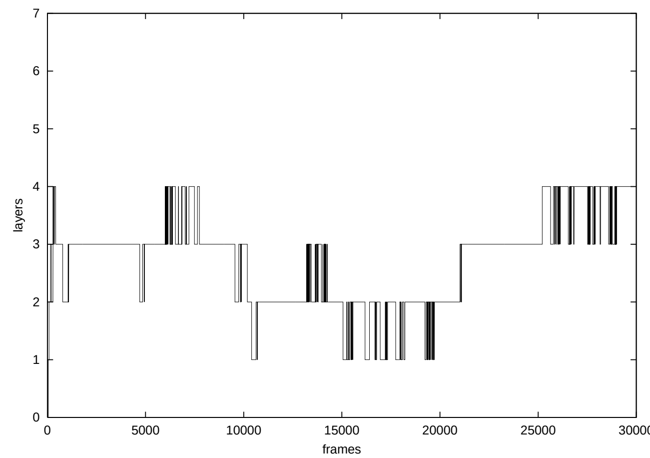

(a) varying bandwidth



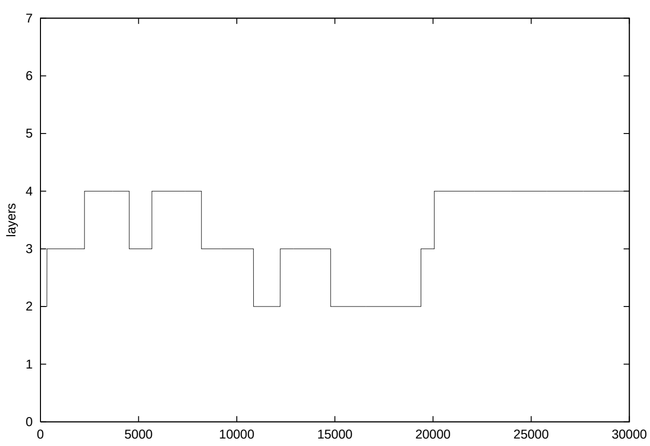

(b) off-line: buffer of 30



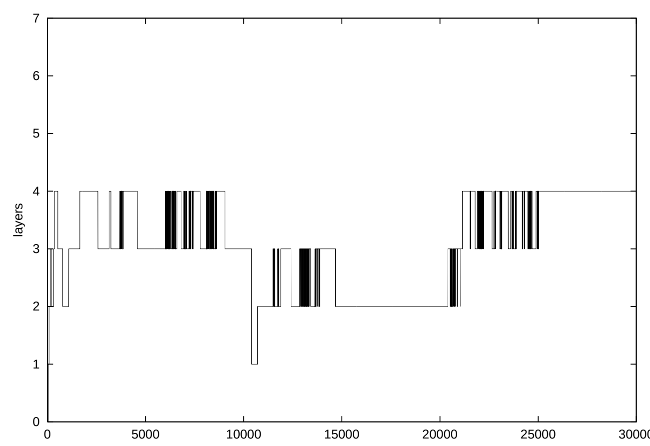

(c) off-line: buffer of 300




(d) adaptive: buffer of 300

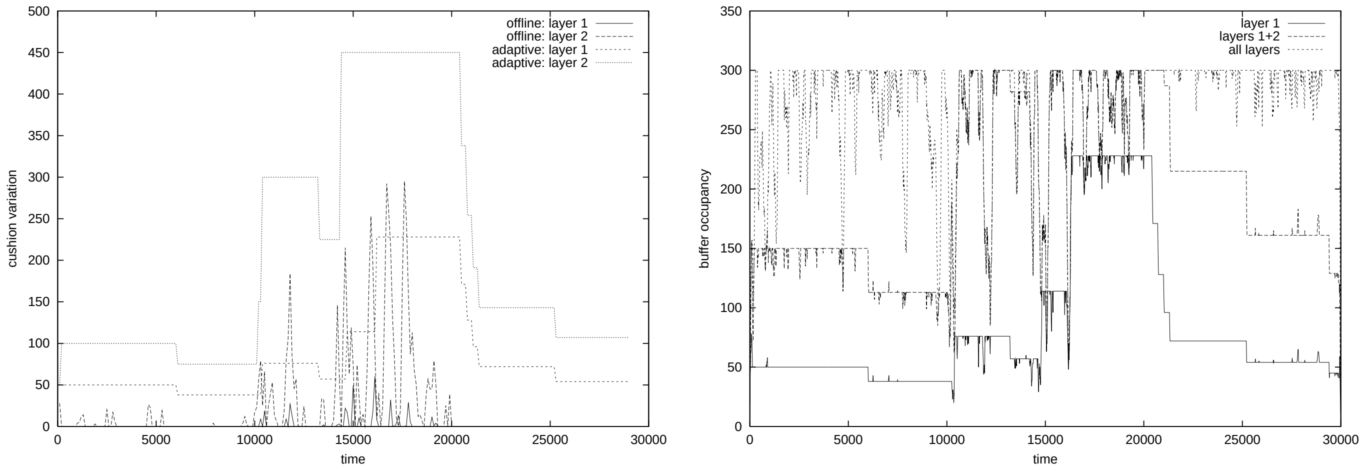



(e) off-line: buffer of 900




(f) adaptive: buffer of 900

Fig. 10. The comparison of smoothness achieved by off-line and adaptive schemes




(a) target buffer cushion

(b) buffer occupancy

Fig. 11. The illustration of cushion adaptation process

times when the adaptive scheme changes the target buffer cushion values. For example, at around frame slot 10350, the actual buffer occupied by layer 1 frames is less than half its target cushion of 38. In response to this, the adaptive scheme doubles the target cushion of layer 1 . Similarly the layer 2 target cushion is also doubled. At around 20400, the actual buffer stayed full for at least 300 frame slots, hence causing the target cushion for both layers to be decreased. It can be seen that target buffer cushion values in the adaptive scheme have the same trend as the ideal buffer cushions of the corresponding layers. However, the adaptive scheme is more conservative while being reactive.

We also measured, using the average run length and expected run length metrics, the relative smoothness of sequences resulting from these algorithms. Figure 12 shows the performance of these schemes, as measured by these metrics, under various buffer settings. The average run lengths for layers 1, 2 and 3 are shown in Figure 12(a) and expected run lengths in Figure 12(b).

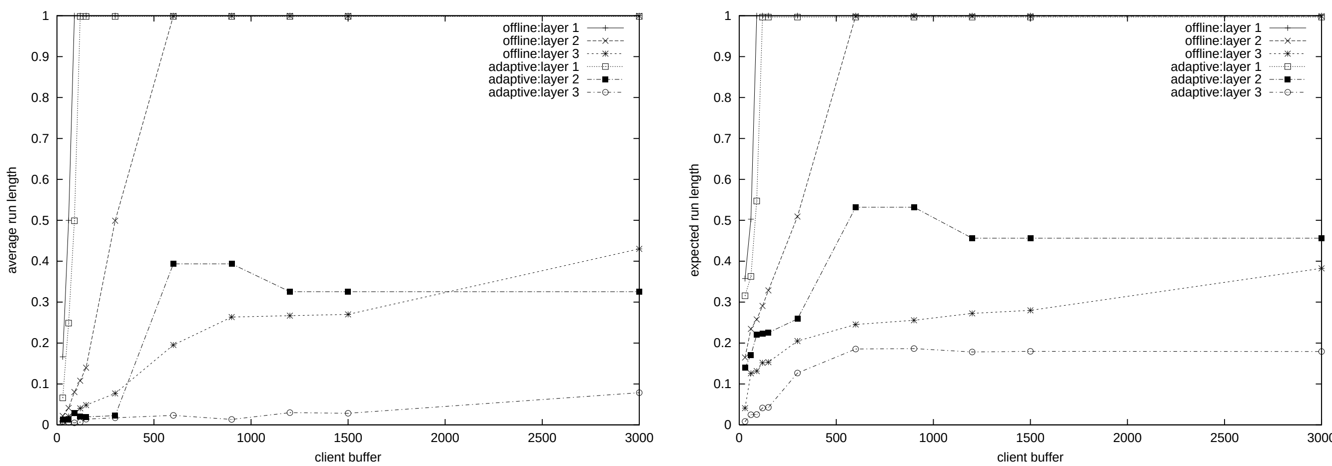
It can be observed that metrics for layer 1 under both the algorithms approach unity even at very small buffer sizes. With the off-line algorithm, the metrics for layer 2 also approach unity for moderate buffer sizes. On the other hand, the metrics for layer 2 in the adaptive scheme increase up to a point as the buffer increases and stagnate after a certain buffer size. This is because the discontinuity at around frame 10350 in the run of layer 2 persists even with large buffer sizes (as evident in Figure 10(f)). This discontinuity is an artifact of the adaptive scheme. Though the adaptive schemes reacts in advance to an onset of congestion, it still may not be able to build sufficient cushion to tide over a sudden but prolonged dip in available bandwidth. Further, in an attempt to use the buffer and bandwidth more effectively, the adaptive scheme chooses not to be too conservative in prefetching and thus may not build a larger cushion even with a larger available buffer. This does not, however, fully explain the decrease in the metrics with increased buffer sizes beyond 900 and requires further investigation. The effect of larger buffers is less pronounced with higher layers since their selection is limited more by the lack of bandwidth than the lack of buffer space.

It is evident that metrics improve faster with the off-line scheme, reflecting its efficient use of buffer. As can be seen, both metrics capture the relative smoothness: the sequences generated by the off-line algorithm score higher than the corresponding sequences from the adaptive algorithm. Moreover, the graphs of both the metrics appear quite similar. This indicates that a work conserving algorithm that maximizes average run length would also come close to maximizing the expected run length metric.

## VII. Related Work

The problem of layer selection for delivering layered video has received a lot of attention recently. One of the earliest works in this area presented in [5] proposes a receiver-driven layered multicast (RLM) protocol for transmitting layered video to heterogeneous receivers. The number of layers subscribed by a receiver is dynamically varied based on the perceived loss rate and thru join experiments. The later studies [3] have shown that receiver-driven schemes such as RLM exhibit significant instability. There were some proposals [1], [4] on addressing this problem inside the network by assigning higher priority to lower layers and providing priority based dropping at routers in case of congestion. These approaches introduce some additional complexity at core routers.

In [8] the available bandwidth is modeled as a stochastic process and optimal allocation of bandwidth between base and enhancement layers is studied. It was assumed that client buffer is unlimited. The work most relevant to ours was reported in [7]. They assume that TCP-friendly congestion control [6] was employed and hence the available bandwidth curve has a sawtooth shape. They address the problem of buffer allocation between layers such that it is used efficiently in absorbing the short-term fluctuations in bandwidth. Though our work is quite similar in spirit to theirs, our approach has been quite different. Our focus has been on designing metrics to capture smoothness criteria and on developing algorithms to maximize these metrics.




(a) average run length

(b) expected run length

Fig. 12. The performance of off-line and adaptive schemes

## VIII. Conclusions and Future work

In this paper we addressed the issue of layer selection for maximizing the perceived video quality under given resource constraints. We defined smoothness criteria and designed metrics namely, *avgrun*, *minrun*, and *exprun* for measuring smoothness. We then developed an optimal offline algorithm MAXAVGRUN to find a sequence with maximum average run length when the network conditions are known *a priori*. We also presented heuristic algorithms, MAXMINRUN and MAXEXPRUN for maximizing the minimum and expected run lengths respectively. We then described a simple adaptive algorithm for providing smoothed layered video delivery that doesn't assume any knowledge about future bandwidth availability. We conducted simulations to study the performance of these schemes and shown that even with a small client buffer it is possible to provide significantly smoother quality video playback.

There are several simplifying assumptions made in this preliminary work on providing smoother quality layered video stream. Specifically, in the adaptive layer selection scheme, we assume the presence of a bandwidth estimator that gives the precise current bandwidth in each frame slot. It is likely that there would be some amount of error in the estimation of available bandwidth. This could lead to packet losses. The packet losses can be handled using retransmissions. Since the frames are prefetched there is sufficient time for error recovery through retransmissions. The trade-off between retransmission of the lost packets and the prefetching of new frames should be investigated.

The schemes, as presented, provide smoother video by favoring longer runs in a layer. They need to be enhanced to further smoothen the video streams by also avoiding short runs. Similarly, the *exprun* metric also needs to be refined to discount short runs. Real experiments need to be conducted to ascertain the relative merit of the proposed metrics and effectiveness of the proposed algorithms. Finally, the schemes presented here work on layered CBR video streams, and need to be extended for VBR video streams too.

## References

[1] S. Bajaj, L. Breslau, and S. Shenkar, "Uniform versus Priority Dropping for Layered Video", Proc. ACM SIGCOMM'98, Sep 1998.

[2] W.-C. Feng, and J. Rexford, "A Comparison of Bandwidth Smoothing Techniques for the Delivery of Compressed Prerecorded Video", Proc. IEEE INFOCOM'97, Kobe, Japan, Apr 1997.

[3] R. Gopalakrishnan, J. Griffoen, G. Hjalmtysson, and C. Sreenan, "Stability and Fairness Issues in Layered Multicast", Proc. NOSSDAV'99, Jun 1999.

[4] R. Gopalakrishnan, J. Griffoen, G. Hjalmtysson, C. Sreenan, and S. Wen, 'A Simple Loss Differentiation Approach to Layered Multicast", Proc. IEEE INFOCOM'00, Tel-Aviv, Israel, Mar 2000.

[5] S.McCanne, V. Jacobson, and M. Vetterli, "Receiver-Driven Layered Multicast", Proc. ACM SIGCOMM'96, Oct 1996.

[6] R. Rejaie, M. Handley, and D. Estrin, "RAP: An End-to-End Rate-based Congestion Control Mechanism for Realtime Streams in the Internet", Proc. IEEE INFOCOM'99, Mar 1999.

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

[8] D. Saparilla, and K. Ross, "Optimal Streaming of Layered Video", Proc. IEEE INFOCOM'00, Tel-Aviv, Israel, Mar 2000.

[9] Z.-L. Zhang, S. Nelakuditi, R. Aggarwal and R.P. Tsang, "Efficient Selective Frame Discard Algorithms for Stored Video Delivery across Resource Constrained Networks," Proc. IEEE INFOCOM'99, New York, Mar 1999.