Critical Bandwidth Allocation Techniques for Stored Video Delivery Across Best-Effort Networks

Wu-chi Feng, Ming Liu
Department of Computer and Information Science
The Ohio State University
Columbus, OH 43210
{wuchi,mliu}@cis.ohio-state.edu

Abstract: In this paper, we propose two new techniques for the delivery of compressed prerecorded video streams across best-effort networks like the Internet. Current approaches for the delivery of stored video across best-effort networks typically alter the quality of the video frames, the frame rate delivered to the user, or a combination of both. By using network feedback, these algorithms continually adjust the video quality to fit within the available network resources. These approaches, however, do not take advantage of the a priori information available from stored video streams, namely the frame sizes that the movie consists of. We will show how monitoring the a priori information and actively monitoring a client-side buffer can help *smooth* the video frame rate delivered to the user, providing for a more consistent quality of video.

1. Introduction

The efficient streamed delivery of compressed digital video has received considerable attention from the research community. There have been two main thrusts of this research: adaptable video transmission techniques and bandwidth smoothing techniques. For adaptable video transmission techniques across best-effort networks, the application (or transport layer) modifies the video stream to fit within the available bandwidth to provide efficient sharing of network resources as well as to avoid congestion collapse. For the delivery of stored video streams, bandwidth smoothing techniques have been developed to take advantage of the *a priori* information available from stored video streams, allowing metrics such as the peak bandwidth requirements to be minimized. Bandwidth smoothing techniques, however, assume that network resources can be reserved; a situation that may not be possible in best-effort networks such as the Internet. For the delivery of *stored* video streams, both of these techniques have limitations.

The adaptable video streaming protocols that have appeared in the literature typically use network feedback to alter a video's frame rate, frame quality, or a mixture of the two in order to reduce the network resource requirements of the stream. Many of the systems that have been developed (as will be discussed in Section 2.1) do not take advantage of the *a priori* information available from stored video streams, resulting in sub-optimal delivery of the video stream. As an example, consider a video stream that has a large number of small size frames followed by a large number of large size frames. While the small frames are being delivered, these adaptive video streaming protocols do not take into consideration that a large number of large frames will be required in the future. As a result, this stream will typically have a larger number of frames dropped when the large frames need to be played back.

Bandwidth smoothing techniques, unlike the adaptable video streaming techniques, create a *plan* for the delivery of stored video streams by using the *a priori* information available from stored video streams (namely the frame sizes within the movie). Using a client-side buffer, these algorithms prefetch large bursts of frames ahead of time by prefetching them into the client buffer. Then, during the playback of these frames network bandwidth can be supplemented with bandwidth out of the client buffer, reducing the network resource requirements. Given a fixed client-side buffer plans for the delivery of stored video can be made that minimize the peak bandwidth requirements as well as other metrics such as the total number of rate changes or the variability of bandwidth requirements. The advantage of these algorithms is that they take advantage of the inherent knowledge available from stored video streams. The main disadvantage, however, is that they require the ability to reserve network and server resources and do not adaptively shape the transmission of the video stream across best-effort networks.

In this paper, we introduce two new techniques for the adaptable streaming of stored video data across best-effort networks. The first technique modifies bandwidth smoothing plans to work with best-effort networks. That is, it creates a bandwidth smoothing plan for the delivery of the streamed video as if rate reservations are possible. Then during transmission, the data stream is adapted on the fly. The second technique extends the notion of critical bandwidth smoothing techniques to deliver data across best-effort networks in a more integrated fashion. The advantage of this approach is that it smooths the frame rate delivered to the user by acting *globally* in the delivery of the data stream instead of always locally adapting the video quality to the available resources. By taking advantage of global information (i.e. the frame sizes of the video), we will show how the video quality delivered to the user can be held more constant. Our results show that this technique can result in more constant video quality delivered to the user under changing network conditions.

In the next section, we describe the relevant background including a discussion of adaptable video delivery techniques for stored video as well as a discussion of bandwidth smoothing techniques. In Section 3, we present our new delivery techniques for the delivery of stored video over best-effort network. In Section 4, we compare and contrast the algorithms with a simple adaptation techniques and discuss metrics such as average frame rate, minimum frame rate, and frame rate variability that these algorithms achieve. Finally, we conclude with a summary and directions for future work.

2. Background and Related Work

In this section, we review the pertinent background information required for this paper. We break our discussion into two main parts: adaptable video techniques (including video streaming protocols) and bandwidth smoothing algorithms. Finally, we provide a listing of the assumptions that we have made for this paper.

2.1. Adaptive Video Transmission Techniques

Several approaches have been proposed for the integrated delivery of streamed video across the Internet from both industry and academia. These systems include

- VIC - a video conferencing tool built for the MBone [11],
- The Continuous Media Toolkit (from the University of California - Berkeley) [16],
- The Quasar Adaptable Streaming Video Player (from the Oregon Graduate Institute) [20],
- Vosaic - University of Illinois' streaming video protocol [2], and
- RealAudio/RealVideo Streaming protocol [15].

VIC is an integrated multicast backbone video conferencing toolkit. This system provides video frame rate and frame quality adaptation in response to changing network conditions. Because the video in VIC is captured on-the-fly, altering the frame quality and frame rate is simpler to accomplish than for stored video sources. VIC provides for congestion management through packet loss monitoring. To avoid congestion collapse within the network, VICs congestion control is *TCP-compatible*. That is, it uses the slow start algorithm of TCP, resulting in the well-known *TCP sawtooth* bandwidth.

The Berkeley Continuous Media (CM) Toolkit is an object-based stored media delivery system using the TCL/TK toolkit. To provide for network adaptation, the CM system uses a logical time system that maps the expected playback times into delivered frames. Using a metafile that is stored with the video stream, adaptation for video streaming is then accomplished through frame rate control. That is, frames are dropped to meet network bandwidth availability. Finally, the CM player uses *cyclic-UDP* for the delivery of the video streams. This transmission protocol provides for congestion management within the network by monitoring the packet cell loss.

The Quasar adaptable streaming player is a sophisticated adaptable streaming video protocol for stored video. The most important property of this player is that it addresses the difficult problem of scaling between frame rate and frame quality. It accomplishes this through *context, view,* and *quality* management. In particular, this system allows users to weigh the relative importance of frame rate and frame quality. The CVQ model then maps this into appropriate frame rate and frame quality operating points. Like the CM toolkit, Quasar uses TCP for control messages and UDP for the video streams. The transport protocol also provides congestion control as in the first two systems. To provide some level of reliability, Quasar retransmits audio frames once but does not retransmit video data. To accomplish frame quality scaling, Quasar stores each video at multiple resolutions along with indices to allow quick access to different quality streams. Finally, Quasar also supports inter-stream synchronization and playback.

The Vosaic media player provides streaming video playback of video through the world wide web browser Mosaic. Each video stream is delivered through their streaming video datagram protocol (VDP). This asymmetric protocol uses an unreliable channel (UDP) for the streaming of the media and TCP based control messages. The rate adaptation algorithm monitors packet loss of the video. When the frame loss is greater than 15%, the quality of the video is scaled down; when the frame loss is less than 5%, the video quality is upgraded. Thus, VDP may not adapt to network congestion as well as the other approaches. Finally, Vosaic uses a demand resend policy to fill in data only if it can be retrieved before its playback deadline.

The RealAudio/RealVideo streaming stored video system uses buffering more actively than the other approaches. In particular, users typically wait at the beginning of playback for some of the data to be prefetched into the client-side buffer. Using measurements of network throughput, the video delivered to the user is matched to an appropriate encoding quality. That is, the system stores each video at several different bit rates. It then delivers the one that most closely matches the available bandwidth. For short running video streams, this approach works well. For long running video streams, however, RealVideo typically results in sudden breaks where the user must wait for more data to be prefetched. The main problem is that RealVideo does not actively monitor the changing network conditions or use the a priori information to adapt the video streams.

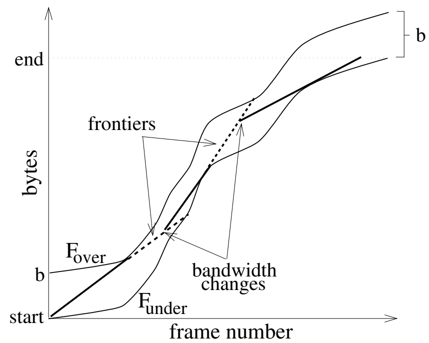


Figure 1: Sample Transmission Plan. This figure shows the buffer underflow and overflow curves for a sample video stream. The plan consists of three constant-rate runs that serve as a schedule for transmitting video frames from the server.

2.2. Bandwidth Smoothing Algorithms

Bandwidth smoothing algorithms *smooth* the bandwidth requirements for retrieval of the video stream from the video server and transmission across the network. By prefetching data ahead of time, a client-side buffer can be used to supplement network bandwidth when the video stream reaches an area of large frame sizes. A compressed video stream consists of n frames, where frame i requires f_i bytes of storage. To permit continuous playback at the client site, the server must always transmit quickly enough to avoid buffer underflow, where

$$F_{under}(k) = \sum_{0}^{k} f_i$$

indicates the amount of data consumed at the client by frame k, where $k = 0,1,...,n-1$. Similarly, the client should not receive more data than

$$F_{over}(k) = b + \sum_{0}^{k} f_i$$

by frame k, to prevent overflow of the playback buffer (of size b). Consequently, any valid server transmission plan should stay within the river outlined by these vertically equidistant functions, as shown in Figure 1. That is, the total amount of delivered data:

$$F_{delivered}(k) = prefetch + \sum_{0}^{k} c_i$$

where c_i is the transmission rate during frame i of the smoothed video stream must adhere to:

$$F_{under}(k) \le F_{deliver}(k) \le F_{over}(k)$$

for every frame within the movie.

The bandwidth smoothing plans that have introduced in the literature create plans that have m consecutive *runs,* each with a constant bandwidth allocation r_j and a duration t_j, where time is measured in discrete frame slots; at time i the server transmits at rate $c_i=r_j$, where slot i occurs during run j. Together, the m bandwidth runs must form a monotonically-increasing, piecewise-linear path that stays between the F_{under} and F_{over} curves. For example, Figure 1 shows a plan with $m=3$ runs, where the second run increases the transmission rate to avoid buffer underflow at the client prefetch buffer; similarly, the third run decreases the rate to prevent overflow.

Given a fixed client-side buffer, these algorithms provide continuous playback of video for the user while

- minimizing the number of bandwidth rate increases required [4, 5],
- minimizing the total number of bandwidth changes required [6],
- minimizing the variability of the bandwidth requirements [17],
- minimizing the buffer utilization of the client-side buffer [7],
- delivering video in constant bandwidth allocation periods [12],
- smoothing across patterns of compressed frame types [18],
- adhering to a buffer residency constraint [9], or
- adhering to on-off bandwidth periods [21].

Many of the above algorithms also minimize the peak bandwidth requirements for the continuous delivery of the video stream given a fixed client side buffers. As an example of how bandwidth smoothing algorithms work, an example bandwidth plan generated by the *minimum changes bandwidth allocation* algorithm for the M-JPEG encoded movie *Beauty and the Beast* is shown in Figure 2. In this figure, we see that the bandwidth smoothing algorithms can result in plans for the delivery of video that have periods of constant bandwidth allocation lasting on the order of 5 to 10 minutes for even a small amount of buffering.

One important theme among these algorithms is that they dictate the bandwidth rates that are required from the underlying network. For dedicated video-on-demand systems, this assumption works fine, allowing the bandwidth smoothing algorithms to minimize the total resources required to deliver the video streams to the users. For best-effort network environments such as the Internet, this assumption *will not allow* the video to be delivered continuously in most circumstances.

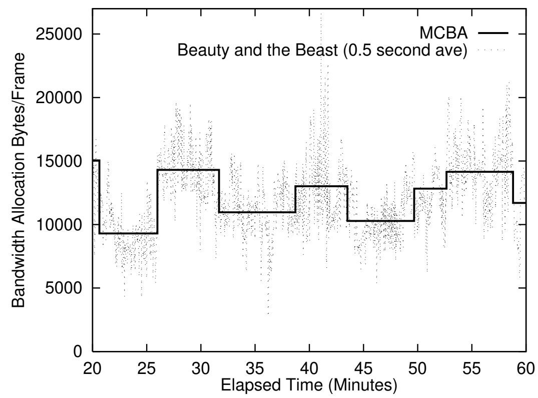


Figure 2: Minimum Changes Bandwidth Allocation Example. This figure shows a sample bandwidth allocation plan generated by the *minimum changes bandwidth allocation* algorithm using a five megabyte buffer for the movie *Beauty and the Beast*. For the given buffer size, the plan generated has, for continuous playback, 1) the minimum peak bandwidth requirement, 2) the maximum-minimum bandwidth, and 3) the smallest number of rate changes possible. The lightly dotted lines represent the half second frame averages.

The notion of bandwidth smoothing, however, can be useful in the delivery of stored video streams. By taking advantage of the *a priori* information available, we can use both the buffer and the bandwidth availability to help determine the frame rate and frame quality that we should be sent. For example, when more network bandwidth becomes available, many of the current adaptable video applications will immediately increase the rate and/or quality of the video delivered to the user. Using a bandwidth smoothing technique, we may keep the frame quality constant while prefetching data of larger frames in the future, ensuring a higher minimum frame rate during playback.

Recently, we proposed a method for the incorporation of bandwidth smoothing techniques for the delivery of stored video[10]. Given a constant bandwidth allocation, this technique uses the *a priori* information from stored video and smooths out the frame drops over time. Because the algorithm works from the end of the movie to the playing point, this algorithm is more suitable for guaranteed bandwidth allocation channels such as ATM's constant-bit-rate (CBR) service or ATM's available-bit-rate (ABR) service. Under changing network conditions, this technique becomes somewhat impractical because of its large computational requirement.

2.3. Assumptions for This Paper

To make the algorithms more understandable, we have made several assumptions throughout the paper. First, we assume that sufficient client-side buffer has been set aside for the removal of delay jitter that the lower layer protocols have. That is, we have assumed that in changing network conditions enough buffer-

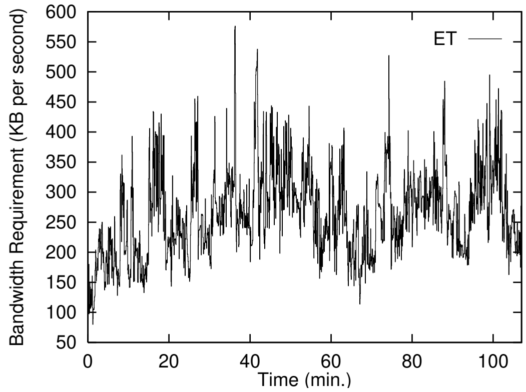


Figure 3: E.T. Example. This figure shows the *3* second frame averages for the M-JPEG encoded movie *E.T.*

ing has been added to smooth the bandwidth availability over second-long periods. This particular requirement is possible because we are delivering stored video streams. Second, we focus our attention to the problem of maximizing the minimum frame rate of video delivered. It is expected that in an implementation of these algorithms that the user would have the ability to set the parameters for the relative importance of the frame rate versus the quality of the video received. Finally, recognizing that the full frame rate of the video can vary, we assume that the full frame rate is 30 frames per second to make the discussion simpler.

3. Adaptable Bandwidth Allocation Algorithms

In this section, we describe three algorithms that can be used for the delivery of stored video streams across best-effort networks. We first describe a simple adaptive algorithm that continually and immediately alters the frame rate of the video stream to fit within the available resource (similar to the techniques currently used). We then describe our proposed algorithms in Sections 3.2 and 3.3. Throughout the discussion as well as in the experimental section, we will refer to the movie *E.T.* which has been digitally captured and compressed using Motion-JPEG (similar to all I-frame MPEG streams). We have graphed the *3* second frame averages in Figure 3.

3.1. A Simple Buffer Management Technique

In this section, we describe an algorithm that uses a buffer to aid in the delivery of video data across networks. At a high-level, this method uses a client-side buffer to buffer data when the frame rate is smaller than the bandwidth available. If the buffer fills up, the server "waits" for buffer space to free up and then transmits the frame. When the buffer is empty, the client has no choice but to drop the frame. To model an "optimal" drop policy for frames, we employ the following. If the buffer contains less data than

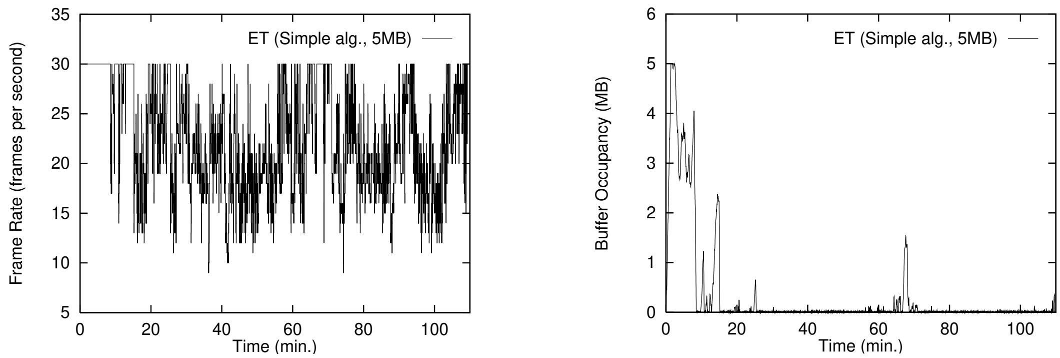


Figure 4: Simple Buffer Management Example. This figure shows the frame rate and buffer occupancy during the playback. The movie used to generate these graphs is *ET* (Quality = 90), and the smoothing buffer used here is 5MB. We used a constant available bandwidth to generate these graphs, which is 70 percent of the average frame size of the whole movie.

the frame we want to playback, we mark the frame as "dropped" and "save" the data as part of the next frame instead of discarding a partial frame. For each frame, we add the maximum bandwidth available into the buffer, assuming that there is room. This approach is similar to the approach used in current adaptable video delivery techniques in that they deliver more frames to the user when the frame sizes are small and deliver fewer frames to the user when frame sizes are large. That is, they use only local information about the bandwidth availability.

An example of this algorithm is shown in Figure 4. The simple algorithm is *greedy* in that it uses all the available bandwidth and does not worry about buffering data for the future. As a result, the only time the client buffer will fill up with data is when the available bandwidth is greater than the bit-rate of the video stream for an extended period of time (see minutes 0-10 in the example). Once the frames get to be larger, the buffer drains and will continue to be drained (from time 10-20 minutes) until an area of small frames allows the buffer to be filled again (time 66 to 70 minutes). The pseudo-code for this algorithm is shown in Figure 5.

3.2. Modifying Existing Bandwidth Smoothing Plans

Our first proposed approach is called the *modified Critical Bandwidth Allocation (CBA)* algorithm. This technique delivers stored video across best-effort networks using a two step process: (1) create a bandwidth plan given a fixed client-side buffer using one of the algorithms presented in Section 2.2 and then (2) modify the plan on-the-fly during transmission. That is, we create a bandwidth smoothing plan and "clip" the regions where the bandwidth requirements are greater than the bandwidth available from the network. This algorithm differs from the algorithm presented in Section 3.1 in an important way, it will not *overflow* the client-buffer when bandwidth availability becomes large compared to the video stream

```

while (not the end of movie) {
     buf += available bandwidth;
     if (buf >= frames[i]) {
         /* We can play frame i */
         play frame i;
         buf -= frames[i];
     }
     else {
         /* We cannot make it */
         Drop frame i;
     }
     i++;
}
```

Figure 5: Pseudo-code for the Simple Adaptive Algorithm. This figure shows the pseudo-code for the simple adaptation algorithm that always (and immediately) alters the frame rate of the video to fit within the available network resources.

bit-rate. Because the first step generated a plan for the delivery of the video stream with the buffer constraint in mind, buffer overflow will not be possible.

To implement this algorithm, we first create the bandwidth smoothing plan using *any* of the algorithms described in Section 2.2. We then deliver the video stream in the following way. First, for each frame i in the movie, we see if c_i is less than the available bandwidth during frame i (see Section 2.2 for the definitions of c_i). If it is, then sufficient network resources are available to transmit *all* of the data originally intended for transmission in the bandwidth smoothing plan. On the other hand, if c_i is greater than the available bandwidth, we must modify the bandwidth to fit within the available resources. To accomplish the rate adaptation, we find the largest frame k that meets the following condition *for the original plans created in step 1*:

$$\max_{0 \le k \le N} (F_{under}(k) \le F_{deliver}(k))$$

This frame k is the last full frame delivered within frame i of the movie *had there been sufficient network resources*. Starting at k, we then start removing frames until the data that must be delivered during frame i has been reduced to the available resources. In essence, this frame dropping simply clips the bandwidth requirement to fit within the available resources. Any excess bandwidth (created from dropping whole frames) is then used to prefetch frame $k+1$. This process is repeated for every frame within the movie as it is being delivered to the user.

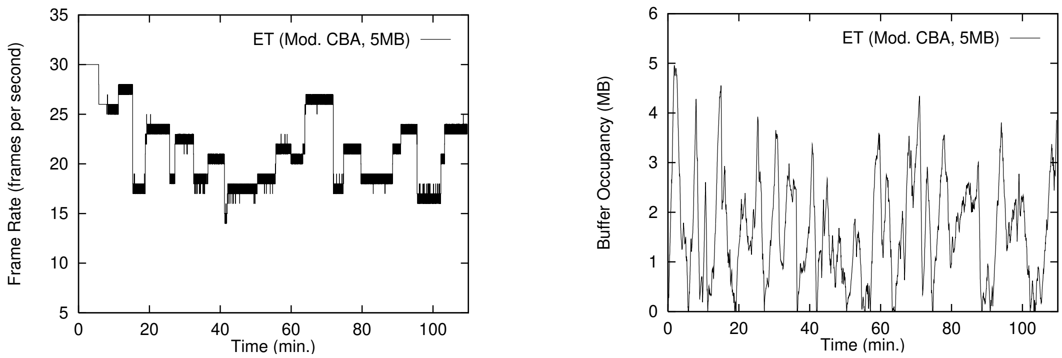


Figure 6: Modified Bandwidth Smoothing Plan Example. This figure shows the frame rate and buffer occupancy graphs for the modified bandwidth smoothing plan for the movie *ET* and a client-side buffer of 5 megabytes.

An example of the delivery of data is shown in Figure 6. As shown in this figure, the modified CBA algorithm results in more stable delivery of the video frame rate compared with the simple adaptive algorithm. In particular, this algorithm results in periods where the buffer occupancy is on the order of a megabyte or more in order to buffer frames for the future. The oscillations between two adjacent frame rates (e.g. minutes 17-21) are the result of rounding errors from dropping whole frames and the way we account for the frame rate. Nevertheless, we see that the frame rate is much more constant over the simple adaptation algorithm.

The pseudo-code for this algorithm is shown in Figure 7. In the pseudo-code, phase 1 (lines 0-2) create a bandwidth smoothing plan given a fixed client-side buffer. The bandwidth smoothing plan fills in the array $F_{deliver}[]$ which denotes the cumulative amount of data that needs to be delivered during each slot (frame) *i* within the movie (see Section 2.2). In line 5, we initialize a variable called *excess*. This variable is used to help keep track of the amount of data that we have prefetched on a previous frame. Its use will become clearer after we describe the rest of the algorithm. Under plentiful network conditions, we would deliver $F_{deliver}[i]$ bytes of data (as created by the bandwidth smoothing plan) during slot *i*. Lines 13-19 of the pseudo-code describe the case where the bandwidth available is greater than required ($F_{deliver}[i]$) during frame *i*. Lines 14-16 take care of the situation where there has been a long sustained bandwidth availability greater than required. In this situation, the algorithm only transmits $F_{deliver}[i]$ - $F_{deliver}[i-1]$ bytes of data. Lines 18-19 take care of the situation where we have prefetched some of the data during slot *i* during slot *i-1*. Because we are dropping whole frames of data, the *excess* variable allows us to take care of rounding errors. As an example, suppose our smoothing plan creates a run that calls for the delivery of 40,000 bytes of data per slot. Further suppose the frame data that we are sending are all 25,000 bytes in size. In this region, we would drop one frame *every* slot, wasting 15,000 bytes of data available per slot. To capi-

```
0  /*** Phase 1 ***/
1  Create F_lo and F_over curves;
2  Run a bandwidth smoothing plan using F_lo and F_over;
3
4  /*** Phase 2 - rate shaping ***/
5  excess = 0;    /*** excess is the prefetching for frame ptr+1 ***/
6  For every frame i in the movie
7    {
8    bw_frame = F_deliver[i]-F_deliver[i-1]; /*** bw req. for frame i ***/
9    bw_avail = bandwidth available from the network
10   amt_remove = (bw_frame-bw_avail)-excess;
11
12   /*** if less than 0, don't need to drop any frames... ***/
13   if (amt_remove < 0) {
14     if (bw_frame < bw_avail)    /*** bandwidth is plentiful ***/
15       Send the data from bw_frame;
16       excess = 0;  /*** don't send extra data --> bw plentiful ***/
17     else           /*** bandwidth not as plentiful --> prefetch ***/
18       Send bw_avail bytes;   /*** Send data from F_deliver[i+1] ***/
19       excess = excess - (bw_frame-bw_avail);
20
21   else /*** Need to remove some data ***/
22     Find largest ptr such that F_lo[ptr] < F_alloc[i];
23
24     /*** Now start dropping data ***/
25     while (dropped < remove)
26       dropped += (frames[ptr]);
27       frames[ptr]=0;
28       ptr--;
29
30     /*** we removed the correct amount, now fix excess to ***/
31     /*** prefetch frame k+1 ***/
32     excess = dropped-remove;
33   }
```

Figure 7: Pseudo-code for modified CBA algorithm. This figure shows the pseudo-code for modified CBA algorithm.

talize on this, we send data from the next slot to reduce the amount of bandwidth required during the next slot, increasing our frame rate. Finally, lines 22-32 take care of removing frames of data such that the amount of bandwidth we transmit during slot *i* that fits within the available resources. Again, *excess* is used to take care of prefetching data from slot *i+1*.

The main advantage of the modified CBA approach for adaptable bandwidth smoothing is that it's easy to implement, and also we only need to run the bandwidth smoothing algorithm once. In addition, it does not have to monitor for buffer overflow in the client since the first pass (running the bandwidth smoothing algorithm) ensures that buffer overflow will not occur. This requirement may be important for long-running movies delivered over intranets where bandwidth may be relatively plentiful. The main disadvantage, however, is that during times of low network bandwidth availability, this algorithm may be too conservative in its usage of network bandwidth. By not delivering as much data as it can (i.e. not being greedy), it may suffer from having less bandwidth availability later in time.

3.3. Adaptable CBA

Our second proposed approach modifies the bandwidth smoothing algorithms to work with best-effort networks in an integrated fashion. The *adaptable critical bandwidth allocation* algorithm takes a target bandwidth availability and modifies the frame rate to fit within the target bandwidth. At a high level, it creates critical bandwidth allocation runs and then modifies the frame rate until the run has a bandwidth requirement less than or equal to the target bandwidth. As the network conditions change, the algorithm is re-run to continually update the frame rate using global knowledge. Thus, instead of telling you what bandwidth requirements are needed, the adaptable critical bandwidth smoothing algorithm adapts the frame rate to fit within the available network resources.

Before we can describe the algorithm, it is imperative that the reader understand the notion of critical bandwidth allocation. We will briefly describe it here, but refer readers to the original paper for a more detailed description of the algorithm [5]. The critical bandwidth algorithm creates a plan for the delivery of stored video by creating *runs* of constant bandwidth allocation. Given a fixed client-side buffer, the $F_{under}()$ and $F_{over}()$ curves, and a starting point within the *river* defined by $F_{under}()$ and $F_{over}()$, the critical bandwidth allocation run creation finds a run that allows the most frames to be played back without buffer underflow or buffer overflow using *a single constant bandwidth allocation*. A sample *run* creation is shown in Figure 8.

<u>Calculation of a single adaptable critical bandwidth run:</u> The main part of this algorithm is the creation of single critical bandwidth runs that fits within available network resources. We will first describe how to calculate a frame rate that fits within the available network resources and then will describe how to continually alter the frame rate to adapt to available network resources.

Given an initial buffer occupancy (amount of data sitting in the client buffer) and a starting frame, we first calculate the critical bandwidth run assuming the full frame rate (30 frames per second). See Figure 9(a). We then compare the bandwidth rate required by the critical bandwidth run creation. If it is greater than the target rate available (Figure (b)), the frame rate must be reduced. To accomplish the frame dropping, we set the frame size of a dropped frame to 0 and space the dropped frames out over every 30

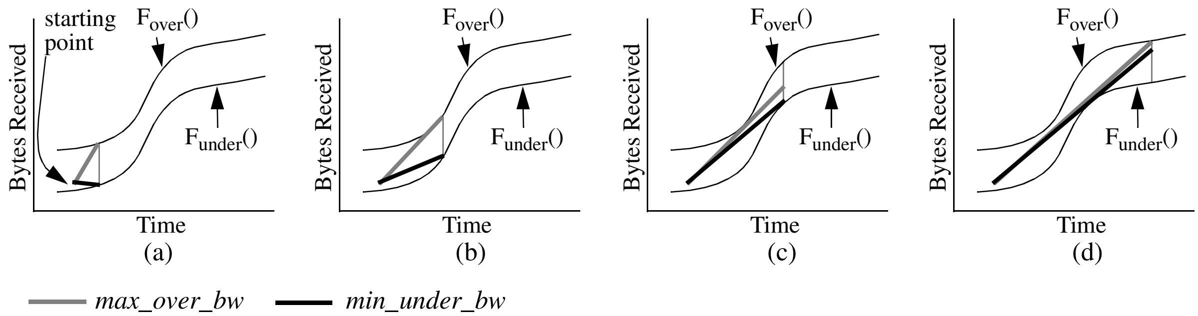


Figure 8: Critical Bandwidth Run Calculation. This figure shows an example critical bandwidth allocation run calculation. Figure (a) shows the early calculations where a large set of possible bandwidth runs exist (between *min_under_bw* and *max_over_bw*). As the calculation advances through the frames, *min_under_bw* and *max_over_bw* continue to converge (Figures (b) and (c)). Figure (d) shows the state of the calculation just prior to the end of the critical bandwidth calculation, and, hence, the selected bandwidth requirement.

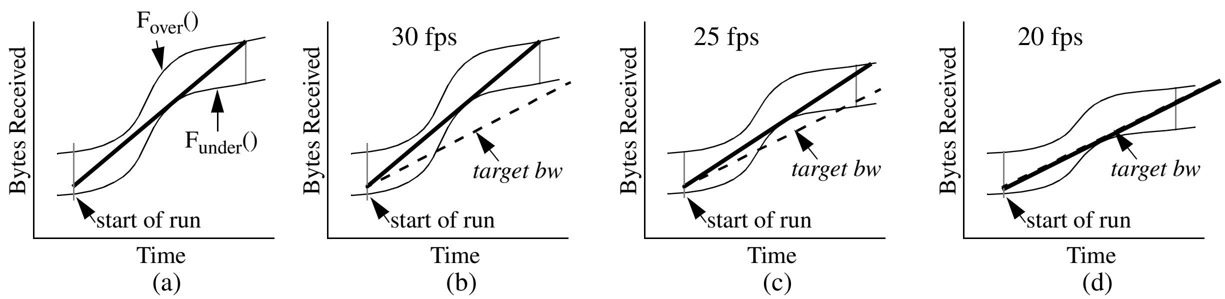


Figure 9: Adaptive Critical Bandwidth Run Creation. This figure shows how the adaptable critical bandwidth allocation algorithm alters the frame rate of the video to fit within the available resources. Figure (a) shows the original bandwidth calculation at the full frame rate. Figure (b) is the same as (a) except that we've shown the target bandwidth that we are trying to aim for. Figure (c) shows a check at 25 frames per second. Note: the distance between the rivers has remained constant, thus, the end of the run is able to extend farther to the right as the frame rate is reduced. Finally, in (d) the target bandwidth has been reached by reducing the frame rate to 20 frames per second.

frames. As an example, at 28 frames per second, we drop every 15th frame and set it to 0. Thus, dropping a frame results in the previous frame longer to fill in the missing frame. We continually reduce the frame rate until the critical bandwidth run algorithm returns a bandwidth requirement less than or equal to the target bandwidth requirement. Once this target frame rate is found, we start transmitting frames at that rate.

<u>Adapting to changing bandwidth availability:</u> In the previous sub-section we described how to alter the frame rate given a single target bandwidth requirement. As the network bandwidth availability changes, the critical bandwidth run calculation may need to be re-evaluated to maximize the frame rate delivered to the user. Because some frame data is typically sitting in the client's buffer, particular care must be taken to ensure that the re-evaluation is done properly. This particular point is best illustrated by example.

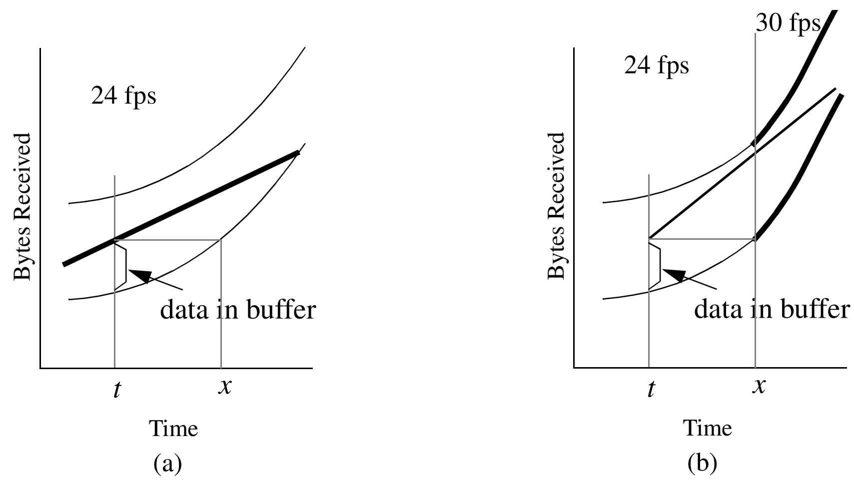


Figure 10: Adapting Critical Bandwidth Allocation to Change. In this figure, we have an example run calculation.

Suppose we have the situation in Figure 10(a). Previous to time *t*, we have calculated that the rate that best matches network resources at 24 frames per second. Suppose we are at playback instant *t* as shown in figure (a) and that more bandwidth has become available. We would like to re-evaluate the frame rate to match the new bandwidth that has become available. *However*, the data sitting in the buffer has been delivered at a rate of 24 frames per second so we cannot simply start the calculation of the new run at frame *t+1* (See figure (b)). Instead, we first find the point on $F_{under}()$ that matches the last frame that we sent to the client (point *x*) in figure (a). Next, we assume that the river from *t* to *x* is fixed. We then "attach" the original $F_{under}()$ and $F_{over}()$ to point *x* (See figure (b)). Using this newly constructed river, we run the critical bandwidth run algorithm. If the rate is greater than the target, we reduce the frame rate *for only the frames after x*. As it turns out, the algorithm we have just described is also the same for when the bandwidth availability decreases, which will presumably have a smaller frame rate than the current frame rate.

The pseudo-code for this algorithm is shown in Figure 11 and an example of this algorithm is shown in Figure 12. In this figure, we see that the adaptable CBA algorithm for a single constant bandwidth allocation results in a much smoother video frame rate delivered to the end user. This is primarily due to the fact that the adaptable CBA algorithm uses the global knowledge of the frame sizes to determine the best frame rate to transmit at any given time. We also see that the adaptable CBA algorithm results in more instances where the buffer is nearly full when compared with the two previous algorithms (shown in Figure 4 and Figure 6).

Implementing the Adaptable Critical Bandwidth Algorithm: There are several implementation details that are worth mentioning. We have assumed in the discussion that the frames do not depend on

```
/*** phase 1 ***/
Create F_under and F_over curves;

/*** Phase 2 - Video Adaptation ***/
For every n frame interval {
  target = available bandwidth;   /*** rate we're shooting for ***/
  Calculate the critical bandwidth run;

  fps = 30;
  while (critical bandwidth > target){
      /*** must reshape the stream ***/
      fps = fps - 1;

      Create New F_under' and F_over' curves after frame x in the movie
          according to buffer occupancy and fps (see Figure 10)

      Calculate the critical bandwidth run for new curve
      } /*** End of while ***/

  Deliver frames at fps frames per second to the user;
  } /*** End of for ***/
```

Figure 11: Pseudo-code for Adaptable CBA algorithm. This figure shows the pseudo-code for adaptable CBA algorithm.

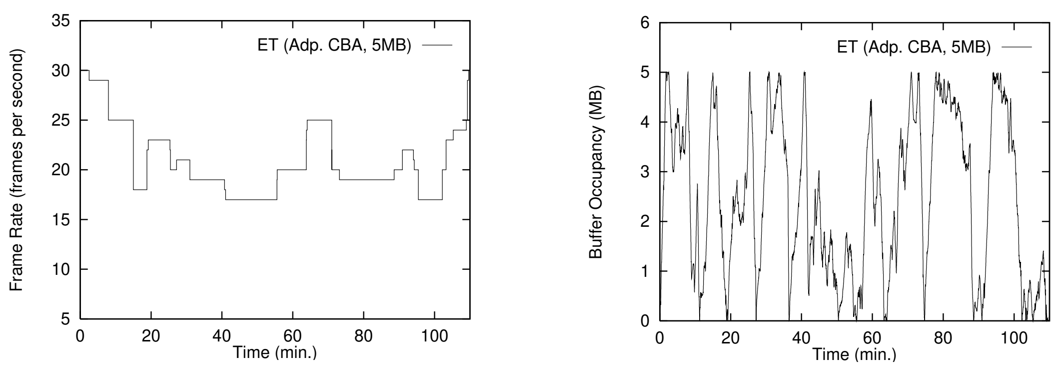


Figure 12: Frame Rate and Buffer Occupancy graph for Adaptable CBA algorithm. This figure shows the frame rate and buffer occupancy of the movie *ET* (Quality = 90) using modified CBA algorithm with 5MB smoothing buffer, and the available bandwidth is constantly at 70 percent of the average frame size.

each other. To allow for MPEG encoded video streams, the frame dropping algorithms would have to be modified to drop the frames using a priority ordering. That is, B-frames first, followed by P and I frames being dropped. Other than the frame dropping algorithm, the adaptable critical bandwidth algorithm will remain the same. We have also assumed that sufficient buffering has been employed to allow for a somewhat stable bandwidth target to shoot for. Because we are delivering stored video, the additional buffer

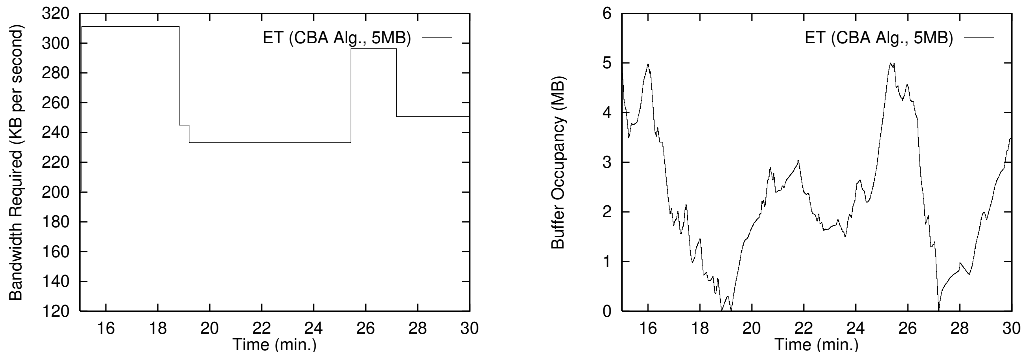


Figure 13: Bandwidth requirement and buffer occupancy graph of the original CBA. This figure shows the bandwidth requirement and buffer occupancy graph of the original CBA algorithm. The movie used is *E.T.* and the smoothing buffer size is 5MB. This graph only shows the time period from 15 minute to 30 minute, that's also the period we used in the experiment.

requirement to somewhat smooth the network bandwidth available (over periods of a second) will not significantly impact the video delivered to the user.

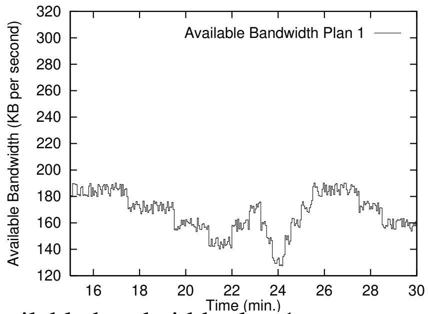
4. Experimentation

In this section, we compare and contrast the proposed video delivery techniques. We first show several example adaptable bandwidth delivery graphs where the bandwidth availability changes over time. We then compare the algorithms on several metrics including: average frame rate delivered, minimum frame rate delivered, and the variability of the frame rate delivered.

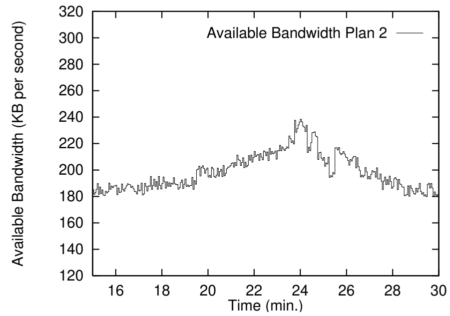
In these experiments, we have chosen to use the movie *E.T. - The Extra Terrestrial* with a quality index of 90. This quality index results in a compression value of 0.85 bits/pixel (corresponding to "good/excellent" picture quality [19]). For the examples in the following sections, we have used *E.T.* along with a client-side buffer of 5 megabytes. For comparison purposes, we have graphed the bandwidth requirement graph and the buffer occupancy graph of the original critical bandwidth allocation algorithm in Figure 13. For the figures, we have displayed the original CBA algorithm for the time frame 15-30 minutes which will match the examples in the following sections.

4.1. Video Adaptation Examples

To highlight the differences between simple frame adaptation and the two proposed algorithms that we have introduced. We have constructed two such sample bandwidth availability traces: one with relatively diminishing bandwidth requirements and the other with relatively increasing bandwidth requirements. We have graphed these available bandwidth graphs in Figure 14. The curve shown in (a) has a lowest band-



a) Available bandwidth plan 1



b) Available bandwidth plan 2

Figure 14: Available bandwidth plans used in the experiments. This figure shows the two different available bandwidth plans used in the experiments. These two plans can show the performance of the algorithms in both high and low bandwidth availability conditions.

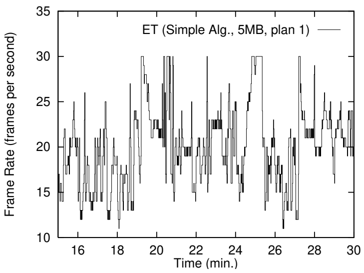
width availability of 50% (132K bytes/second) of the average bandwidth requirement for the whole movie. In the curve shown in (b), the bandwidth availability starts at approximately 70% and has a maximum bandwidth availability of 90% (238K bytes/second) of the average bandwidth requirement for the movie.

Video Adaptation Under Plan 1

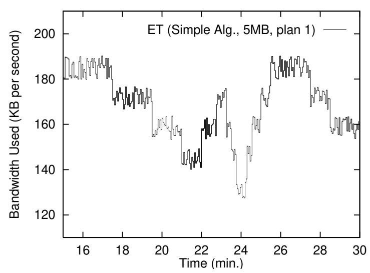
Using the bandwidth availability graph shown in Figure 14(a), we have shown the frame rate, bandwidth, and buffer occupancy graphs for the three video adaptation algorithms in Figure 15

Figures (a) through (c) show the graphs for the simple video adaptation algorithm. Because the bandwidth availability is never greater than the frame rate for extended periods of time, very little buffering of data occurs (see (b)). We also see that the simple adaptation algorithm has significant burstiness in the frame rate delivered to the end user, ranging from 11 frames per second to 30 frames per second. The frame rate increases to 30 frames per second only when the frames being transmitted are smaller than the available bandwidth. Finally, because the rate adaptation occurs immediately, we see that the frame rate delivered to the user reflects this situation.

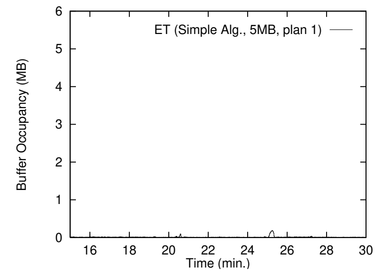
Figures (d) through (f) show the graphs for the modified CBA algorithm. Recall, the modified CBA algorithm works in two passes, first creating a bandwidth smoothing plan and then modifying the bandits smoothing plan to fit within the available network bandwidth. There are several interesting points worth discussion. First, the frame rate of video delivered to the user is much more consistent that of the simple adaptation algorithm. Because the modified CBA algorithm is altering an *already* smooth delivery plan, it tends to space out the frames that it is dropping to meet the available bandwidth requirement. Second, we see in figure (e) that the bandwidth usage for the modified CBA and the simple algorithm are exactly the same, as expected. Finally, as shown by figure (f), we see the advantage of the modified CBA algorithm



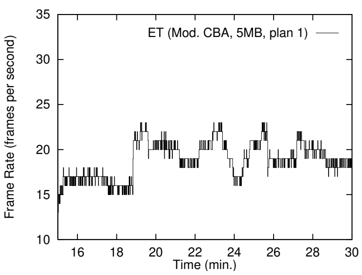
a) Frame Rate (Simple adapt. alg.)



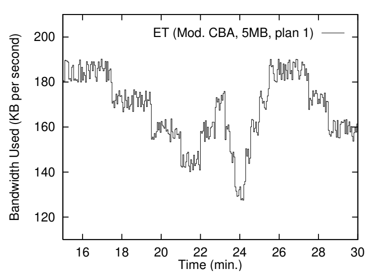
b) Bw. Usage (Simple adapt. alg.)



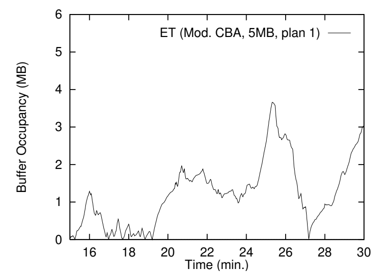
c) Buffer Occ. (Simple adapt. alg.)



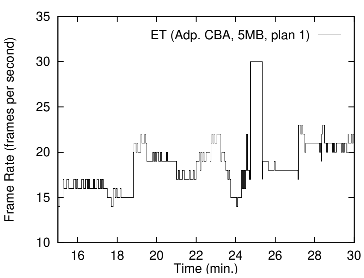
d) Frame Rate (Modified CBA)



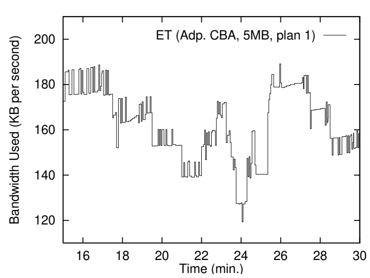
e) Bw. Usage (Modified CBA)



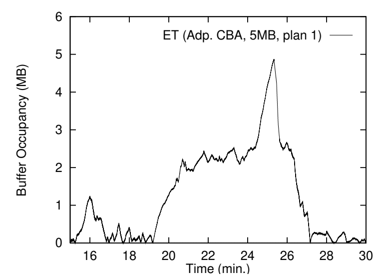
f) Buffer Occupancy (Modified CBA)



g) Frame Rate (Adaptable CBA)



h) Bw. Usage (Adaptable CBA)



i) Buffer Occupancy (Adaptable CBA)

Figure 15: Comparison of the Adaptable CBA, the Modified CBA and the Simple Adaptation Algorithm Under Bandwidth Plan 1. This figure shows the frame rate, bandwidth requirement and buffer occupancy graph of the simple adaptation algorithm, the modified CBA and the adaptable CBA running with the available bandwidth plan 1.

over the simple adaptation algorithm in that it actively buffers data ahead of time. In this particular case, we see that the maximum buffer occupancy is approximately 4 megabytes (~ minute 26), while the original CBA algorithm called for the buffer to be full at this point (see Figure 12 (b)). This highlights the fact that the modified CBA algorithm produces a conservative usage of the buffer. That is, the only time the buffer will be full is under plentiful network bandwidth availability.

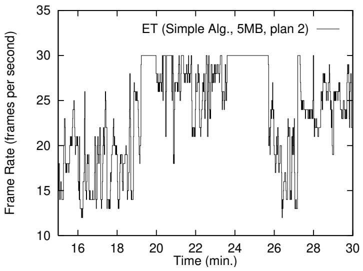
Figures (g) to (i) show the graphs for the adaptable CBA algorithm. In figure (g), we see that the adaptable CBA algorithm results in slightly more consistent frame rates delivered to the user than the modified CBA algorithm and much more consistent frame rates than the simple adaptation algorithm. Of particular interest, however, is the small peak in frame rate from minute 25 to 25.5 in the movie. At approximately that time, two important things happen: the available bandwidth increases as shown by figure (h) and the buffer is nearly full. Before minute 25, the adaptable CBA algorithm had made plans for the delivery of the video stream at a *low* bandwidth availability. As a result, it started to buffer data ahead of time for the larger frames in the future. Once more bandwidth became available (after minute 25), the adaptable CBA algorithm had a nearly full buffer. To use the bandwidth, it went to 30 frames per second, knowing that the buffer would not build up any more. In (h), the bandwidth usage is not the same as the other algorithms. Because the adaptable CBA algorithm bases it's bandwidth usage on the actual movie data and shapes the stream by dropping integer frames (i.e. whole frames), some rounding in the calculations occur leading to bandwidth that is not always used.

Video Adaptation Under Plan 2

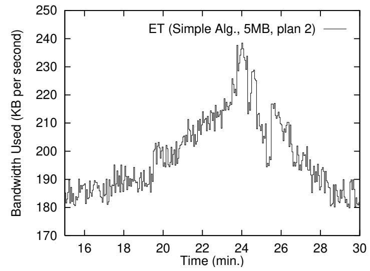
Using the same region within the movie, we have graphed the same plots from Figure 15 using the bandwidth availability plan 2 in Figure 16. Because of the greater bandwidth availability, we see that the frame rate delivered to the user for all the algorithms is higher than those in Figure 15. For figures (a) to (c), we see that the simple adaptation algorithm still results in significant frame rate variation. We also see that the only time buffer builds up under simple adaptation is when the frame rate delivered to the user is 30 frames per second (minutes 23 - 25) *and* the bandwidth availability is greater than the frame rate for an extended period of time.

Figures (d)-(i) are very similar to those in the other bandwidth availability example. There is, however, one point worth mentioning. In figure (d) and (g), we see that around minute 25, the adaptable CBA algorithm has a more variation in frame rate than the modified CBA algorithm. This again highlights the fact that the adaptable CBA algorithm can, at time, be somewhat unstable when the bandwidth availability rapidly changes and the buffer is nearly full.

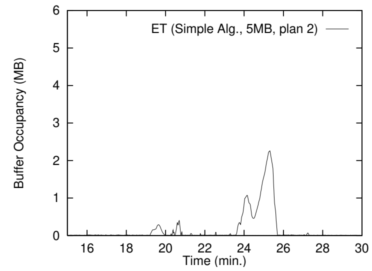
From these two experiments, we can see the differences among the three algorithms. The simple adaptation algorithm has the most bursty frame rate and does not actively use the client-buffer under smaller available network bandwidth. The two algorithms based on the CBA algorithm have similar performance with the modified CBA algorithm being slightly more conservative than the adaptable CBA algorithm. Finally, in all cases, the minimum frame rate delivered to the user is higher using the critical bandwidth based approaches than using the simple adaptation algorithm. In the next subsections, we explore the effect of buffer size and available bandwidth on the frame rate and frame rate variability for the different algorithms.



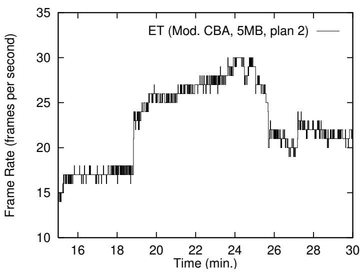
a) Frame Rate (Simple adapt. alg.)



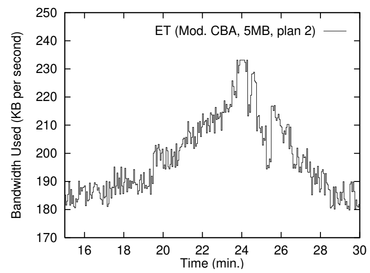
b) Bw. Usage (Simple adapt. alg.)



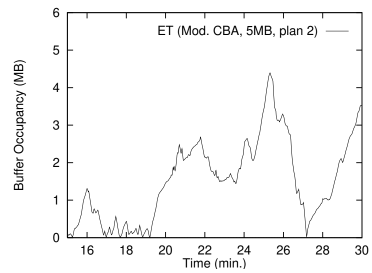
c) Buffer occ. (Simple adapt. alg.)



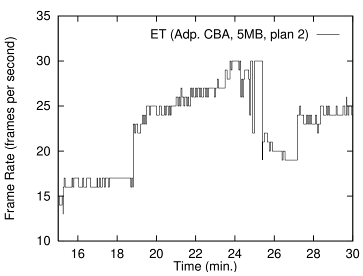
d) Frame Rate (Modified CBA)



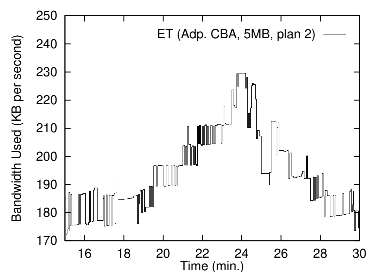
e) Bw. Usage (Modified CBA)



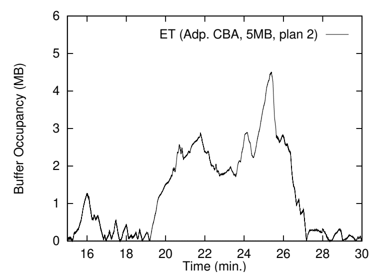
f) Buffer occupancy (Modified CBA)



g) Frame Rate (Adaptable CBA)



h) Bw. Usage (Adaptable CBA)

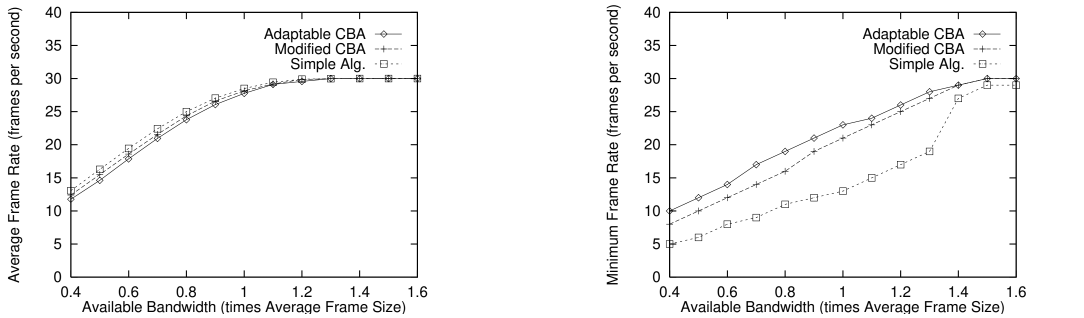


i) Buffer occupancy (Adaptable CBA)

Figure 16: Comparison of the Adaptable CBA Algorithm, the Modified CBA Algorithm and the Simple Adaptation Algorithm Under Bandwidth Plan 2. This figure shows the frame rate, bandwidth requirement and buffer occupancy graph of the simple adaptation algorithm, the modified CBA and the adaptable CBA running with the available bandwidth plan 2.

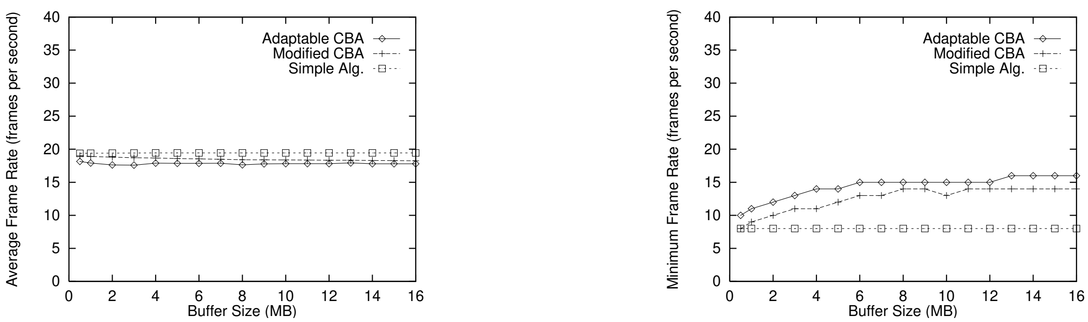
4.2. The Effect of Available Bandwidth

To show the effect of available bandwidth on the frame rate delivered to the user, we have run simulations using a fixed 5 megabyte client-side buffer for varying available bandwidth. For each point in the graph, we have assumed that the bandwidth availability was *constant* for the playback of the entire movie. By holding the bandwidth allocation constant, it allows us to more accurately compare the performance of the various algorithms. The average and minimum frame rate graphs for the various algorithms is shown in Figure 17. In figure (a), we see that the algorithms provide similar average frame rates with the simple adaptation algorithm having the highest frame rate and the adaptable CBA algorithm having the lowest. Figure (b) shows why this happens. The adaptable CBA algorithm achieves the highest minimum frame



a) Average frame rate on different bandwidth b) Minimum frame rate on different bandwidth

Figure 17: Effect of Available Bandwidth on Frame Rate. This figure shows the average and minimum playback frame rate of three algorithms under different available bandwidth or buffer size conditions.



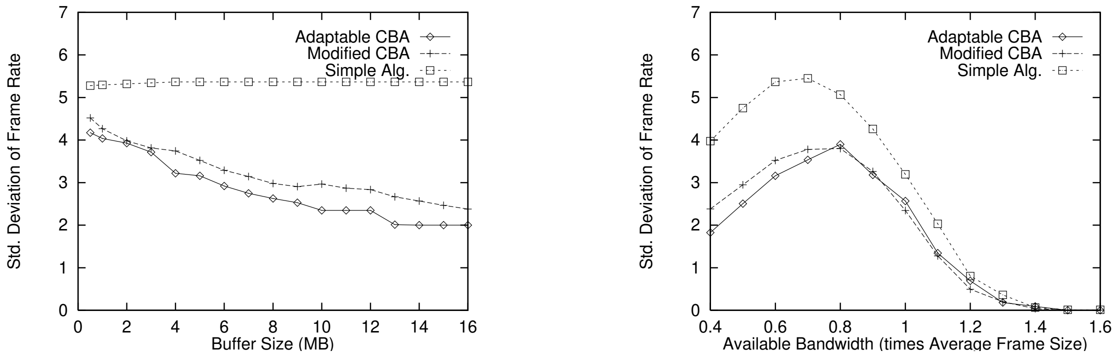
a) Average frame rate on different buffer size b) Minimum frame rate on different buffer size

Figure 18: Effect of Buffer Size on the Frame Rate. This figure shows the effect of changing the client-side buffer size for the algorithms using a *constant* bandwidth allocation of 60% of the average frame rate for the whole movie (158.6K bytes/second).

rate delivered to the user. Because this algorithm tends to buffer larger frames to ensure a higher minimum frame rate, it does not deliver as many of the smaller frames. Therefore, the adaptable CBA algorithm achieves a slightly lower average frame rate for a higher minimum frame rate delivered to the user. Finally, as expected, when the bandwidth availability continues to increase, the frame rate approaches 30 frames per second (i.e. the full frame rate).

4.3. The Effect of Available Bandwidth

To show the effect of buffering on the frame rate delivered to the user, we have run simulations using a constant bandwidth allocation of 60% of the average frame rate for the entire movie and varied the client-side buffer size. The results of these experiments are shown in Figure 18. As shown in figure (a), all the algorithms achieve nearly constant average frame rates. The adaptable CBA algorithm slightly decreases the average frame rate as the buffering increases. The main reason for this is the same as in the last section,



a) Std. deviation of frame rate on different buffer sizes

b) Std. deviation of frame rate on different bw.

Figure 19: The standard deviation of the frame rate under different conditions. This figure shows the frame rate standard deviations of three algorithms under different available bandwidth or buffer size conditions.

the algorithm tries to prefetch larger frames to ensure a higher minimum frame rate, rather than maximizing the average frame rate delivered to the user. In figure (b), we see that the adaptable CBA algorithm achieves the highest minimum frame rates, as expected. For large buffer sizes, the adaptable CBA algorithm has a much higher minimum frame rate delivered to the user. The simple adaptation algorithm has a constant minimum frame rate. This is due to the fact that the bandwidth availability is constant in the experiments as well as the fact that the minimum frame rate is determine by a single area within the movie for the simple adaptation algorithm.

4.4. Variability in Frame Rate

The minimum and average frame rate measurements presented in the two previous sections give us an indication of the number of frames delivered to the user but do not tell us about how variable the frame rate was over time. In Figure 19, we have graphed the variability in frame rates delivered to the user for the same experimental conditions as found in the previous two subsections. In figure (a), we see that the CBA-based approaches yield much smaller variation in frame rates. In particular, for larger buffer sizes the frame rate becomes more constant for the CBA-based approaches, resulting in smaller variation in frame rates. In (b), we see that the CBA-based approaches also deliver a much smaller frame rate variability than the simple adaptation algorithm as the bandwidth availability varies. When the available bandwidth becomes large (> 1.4 times the average frame size), all the algorithms approach 0. This happens because all the algorithms deliver all the frames and hence 30 frames per second for the whole movie. Under smaller available bandwidth requirements, the adaptable approaches perform much better than the simple adaptation algorithm.

5. Acknowledgments

We would like to thank (in advance) any useful comments the reviewers can give us concerning this document. This work was supported in part by grants from the Ohio State University Research Foundation and Ameritech.

6. Conclusion and Future Work

In this paper, we have proposed two algorithms for the adaptable streaming of stored video across best-effort networks. While many end-to-end streaming video applications and protocols exist, they typically use a buffer to smooth out network jitter only, continuing to adapt to network bandwidth as it become available or goes away. By taking advantage of the *a priori* information available from stored video streams, it is possible to analyze, given the current network bandwidth availability, whether one should increase the quality of the video stream when more network bandwidth becomes available relative to the video frame sizes or to hold the frame rate steady to ensure a reasonably constant frame rate in the future. By taking advantage of this information, it is possible to *smooth* out the frame rate received from the video server. It is the belief o the authors that users are more tolerable of a lower quality stream as long as the frame rate and quality are held more constant, rather than having large fluctuations in frame rate and video quality.

The modified CBA algorithm is a simpler two-pass adaptation algorithm that is more conservative than the adaptable CBA algorithm which integrates the smoothing of video with the shaping of the video stream. Both of the CBA algorithms were shown to have smaller frame rate variation than simple adaptation of the video stream. These algorithm also had slightly smaller average frame rates compared to the simple adaptation algorithm in order to ensure higher minimum frame rates. That is, they buffered fewer *larger* frames instead of more *smaller* frames to achieve more constant frame rates.

7. References

[1] Shanwei Chen, Johnathan Walpole, Calton Pu, "Flow and Congestion Control for Internet Media Streaming Applications", In *Proceedings of SPIE Multimedia Computing and Networking 1998*, Kevin Jeffay, Dilip Kandlur, Timothy Roscoe, Editors, Proceedings of SPIE Vol. 3310, pp 250-264.

[2] Zhigang Chen, S.M. Tan, R. Campbell, Y. Li, "Real Time Video and Audio in the World Wide Web", in *Fourth International World Wide Web Conference*, Boston, Massachusetts, December 1995.

[3] Nelson Manohar, Marc Willebeek-LeMair, "Applying Statistical Process Control to the Adaptive Rate Control Problem", In *Proceedings of SPIE Multimedia Computing and Networking 1998*, Kevin Jeffay, Dilip Kandlur, Timothy Roscoe, Editors, Proceedings of SPIE Vol. 3310, pp 42-56

[4] Wu-chi Feng, S. Sechrest, "Smoothing and Buffering for Delivery of Prerecorded Compressed Video", In Proceedings of *IS&T/SPIE Multimedia Computing and Networking* , Feb. 1995, San Jose, CA, pp. 234-242.

[5] Wu-chi Feng, S. Sechrest, "Critical Bandwidth Allocation for the Delivery of Compressed Prerecorded

Video”, *Computer Communications*, Vol. 18, No. 10, Oct. 1995, pp. 709-717.

[6] Wu-chi Feng, F. Jahanian, S. Sechrest, “An Optimal Bandwidth Allocation Strategy for the Delivery of Compressed Prerecorded Video”, *ACM/Springer-Verlag Multimedia Systems Journal,* September 1997.

[7] Wu-chi Feng, “Rate-Constrained Bandwidth Smoothing for the Delivery of Stored Video”, in *Proceedings of the 1997 SPIE Multimedia Computing and Networking Conference*, San Jose, CA, Feb. 1997.

[8] Wu-chi Feng, Jennifer Rexford, “A Comparison of Bandwidth Smoothing Techniques for the Delivery of Compressed Prerecorded Video”, *Proceedings of IEEE INFOCOM ‘97*, Kobe, Japan, pp. 58-66, April 1997.

[9] Wu-chi Feng, “Time-Constrained Bandwidth Smoothing for Interactive Video-on-Demand Systems”, *Proceedings of the 13th International Conference on Computer Communications 1997*, Cannes, France, pp. 291-303.

[10] Wu-chi Feng, Brijesh Krishnaswami, Arvind Prabhudev, “Proactive Buffer Management for the Delivery of Stored Video Across Best Effort Networks”, *to appear ACM International Conference on Multimedia*, Bristol, UK, September 1998.

[11] Steve McCanne, V. Jacobson, “VIC: A Flexible Framework for Packet Video”, Proceedings of *ACM Multimedia 1995*, November 1995.

[12] Jean McManus, K. W. Ross,“Video on demand over ATM: Constant-rate transmission and transport,” In *Proceedings of IEEE INFOCOM*, pp. 1357-1362, March 1996.

[13] Pramod Pancha, M. El Zarki, “Prioritized Transmission of Variable Bit Rate MPEG Video”, *Proceedings of IEEE GLOBECOM 1992*, pp 1135-1139.

[14] S. Ramanathan, P.V. Rangan, H.M. Vin, S.S. Kumar, “Enforcing Application-Level QoS by Frame-Induced Packet Discarding in Video Communications”, *Computer Communications*, Vol. 18, No. 10, October 1995, pp 742-754.

[15] RealNetworks - http://www.real.com

[16] L.A. Rowe, K. Patel, B.C. Smith, K. Liu, " MPEG Video in Software: Representation, Transmission and Playback," *Proc. of IS&T/SPIE 1994 Int'l Symp. on Elec. Imaging: Science and Technology*, San Jose, CA, February 1994.

[17] J.D. Salehi, Z.L. Zhang, J.F. Kurose, D. Towsley, “Optimal Buffering for the Delivery of Compressed Prerecorded Video”, in *Proceedings of ACM SIGMETRICS*, pp. 222-231, May 1996.

[18] H.M. Smith, M.W. Mutka, “Pattern Smoothing for Compressed Video Transmission”, *In Proceedings of IEEE International Conference on Communications*, June 1997.

[19] Gregory Wallace, “The JPEG Still Picture Compression Standard”, *Communications of the ACM*, Vol. 34, No. 4, April 1991, pp. 30-44.

[20] Jonathan Walpole, Rainer Koster, Shanwei Cen, et. al, "A Player for Adaptive MPEG Video Streaming Over The Internet", *Proceedings 26th Applied Imagery Pattern Recognition Workshop*, SPIE, Washington DC, October 15-17, 1997.

[21] J. Zhang, J. Hui, “Traffic Characteristics and Smoothness Criteria in VBR Video Traffic Smoothing”, In *Proceedings of IEEE Multimedia Systems 1997*, June 1997.