Multimedia Systems (1997) 5:297–309

An optimal bandwidth allocation strategy for the delivery of compressed prerecorded video

Wu-chi Feng, Farnam Jahanian, Stuart Sechrest

Software Systems Research Lab, Department of Electrical Engineering and Computer Science, University of Michigan, Ann Arbor, Michigan, USA
e-mail: {wuchi,farnam,sechrest}@eecs.umich.edu.

Abstract. The transportation of prerecorded, compressed video data without loss of picture quality requires the network and video servers to support large fluctuations in bandwidth requirements. Fully utilizing a client-side buffer for smoothing bandwidth requirements can limit the fluctuations in bandwidth required from the underlying network and the video-on-demand servers. This paper shows that, for a fixed-size buffer constraint, the critical bandwidth allocation technique results in plans for continuous playback of stored video that have (1) the minimum number of bandwidth increases, (2) the smallest peak bandwidth requirements, and (3) the largest minimum bandwidth requirements. In addition, this paper introduces an *optimal bandwidth allocation* algorithm which, in addition to the three critical bandwidth allocation properties, minimizes the total number of bandwidth changes necessary for continuous playback. A comparison between the optimal bandwidth allocation algorithm and other critical bandwidth-based algorithms using 17 full-length movie videos and 3 seminar videos is also presented.

Key words: Video-on-demand – Bandwidth allocation – MPEG – Video compression – Smoothing

1 Introduction

Video applications, such as video-on-demand services, rely on both high-speed networking and data compression. Data compression can introduce burstiness into video data streams which can complicate the problem of network and server resource management. For live-video applications, the problem of video delivery is constrained by the requirement that decisions must be made on-line and that the delay between sender and receiver must be minimized. As a result, live-video applications may have to settle for weaker guarantees of service or for some degradation in quality of service. Work on problems raised by the requirements of live video includes work on statistical multiplexing [2, 11], smoothing in exchange for delay [8], jitter control [12, 13], and adjusting the quality of service to fit the resources available [10]. Stored video applications, on the other hand, can take a more flexible approach to the latency of data delivery. In particular, they can make use of buffering to smooth the burstiness introduced by data compression. Because the entire video stream is known *a priori*, it is possible to calculate a complete plan for the delivery of the video data that avoids both the loss of picture quality and the loss of network bandwidth due to overstatement of bandwidth requirements, making network and server scheduling easier.

The utility of prefetching is quite simple to explain. Since the bytes for any given frame can be supplied either by the network or by a prefetch buffer, the burstiness of the network bandwidth requirement can be controlled by filling the prefetch buffer in advance of each burst by delivering more bytes across the network than needed, and draining it in the course of the burst. The size of the prefetch buffer, of course, determines the size of burst that can be averaged out in this way. With a small buffer, only a limited amount of data can be prefetched without overflowing the buffer, so the bandwidth required of the network will remain relatively bursty. With a larger buffer, on the other hand, there is the possibility that most of the burstiness of a video clip can be eliminated through prefetching. This, however, requires a plan for prefetching the data that ensures that the large buffer is filled in advance of bursts which place high demand upon the buffer.

In a previous paper, we introduced the notion of *critical bandwidth allocation* (CBA), which provides plans for smoothing the network bandwidth by utilizing a buffer of a fixed size [6]. The critical bandwidth algorithm allows for long sequences of monotonically decreasing bandwidth requirements, such that, at any point during playback, the bandwidth allocated is the minimum constant bandwidth necessary to play back the video without buffer overflow or underflow. In this paper, we first show that the CBA algorithm results in plans for the continuous playback of video that have (1) the minimum number of bandwidth increases, (2) the smallest peak bandwidth requirements, and (3) the largest minimum bandwidth requirements. We then introduce (and prove) an optimal bandwidth allocation algorithm for stored video which, in addition to the properties of the

Correspondence to: W.-c. Feng

CBA algorithm, minimizes the total number of bandwidth changes for continuous, uninterrupted video playback. That is, the algorithm guarantees that the plan does not cause the available buffer to underflow or to overflow while producing the fewest possible changes in bandwidth.

In the following section, a description of the original critical bandwidth algorithm is presented along with an optimal version of the algorithm. A proof of optimality is also presented for the optimal bandwidth algorithm. The evaluation section compares and contrasts the alternative smoothing algorithms using 17 full-length Motion-JPEG-encoded movies. Finally, a summary and conclusions about the importance of smoothing to the design of network services is presented.

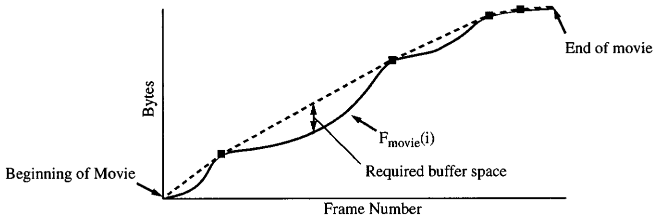


Fig. 1. Critical bandwidth allocation example. The *solid line* in the graph shows a possible graph for $F_{movie}(i)$, while the *dotted line* shows a plan determined by the CBA algorithm. The *dotted set of lines* show the CBA algorithm's bandwidth plan that requires four decreases in bandwidth, while the *squares* on the dotted lines show the junctures between runs. The slope of each dotted line is the bandwidth requirement for that run. The minimum buffer size is represented by the maximum vertical distance between the critical bandwidth allocation plan and the function $F_{movie}(i)$

2 Bandwidth allocation algorithms

In dealing with compressed video data, smoothing techniques attempt to remove the burstiness with appropriate prefetching and delay. An understanding of how burstiness is introduced into a video stream can provide insight into the effectiveness of the various smoothing algorithms. For prerecorded streams compressed with algorithms such as MPEG [1, 9], burstiness occurs at two scales: in small runs of frames as a result of the pattern of frame types (I, P, or B) used in compression and at a larger scale with variations in scene content. As was originally shown, the CBA algorithm can be an effective algorithm in smoothing variations in both pattern and scene content burstiness [7]. In this section, we review the critical bandwidth algorithm and prove several of its key results. In addition, we introduce and prove the optimal bandwidth allocation algorithm.

2.1 Critical bandwidth allocation

The CBA algorithm without regard to the smoothing buffer size creates a bandwidth allocation plan for video data which contains *no increases* in bandwidth requirements for continuous playback and does not require any prefetching of data before playback can begin. By calculating such a bandwidth plan, admission control is greatly simplified. That is, the network manager needs to only ask – "*Is there enough bandwidth to start the flow of data?*". Because the CBA algorithm calculates the minimum bandwidths that are necessary for continuous playback, the buffer size requirement for continuous playback, may be fairly substantial for long video clips. The buffer size requirement, however, is not as large as one required by a single constant bandwidth allocation for the entire video. For clarity, we say that a bandwidth allocation plan consists of runs of constant bandwidth allocations.

We can describe the intuition behind the CBA algorithm with a geometric model. Given any map of frame sizes for a particular movie, a graph can be drawn that has the following function:

$$F_{movie}(i) = \sum_{j=1}^{i} FrameSize_j \ .$$

This function is the running summation of frame sizes for the movie, and must be a monotonically increasing function (see Fig. 1). To avoid buffer underflow, any correct plan must have the total bandwidth received (TBR) at frame i, such that the following condition holds for all frames, i, within the movie:

$$F_{movie}(i) \leq TBR(i) \ .$$

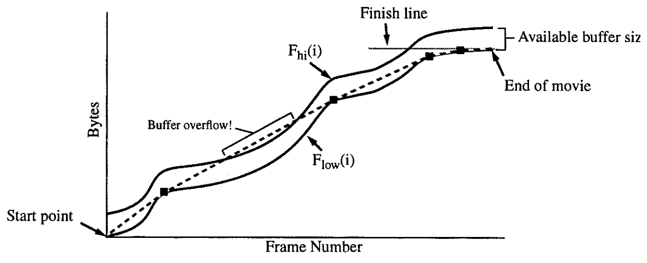
The CBA algorithm allocates a decreasing sequence of constant bandwidths at the minimum bandwidths necessary to play back the video without buffer underflow. This corresponds to creating a convex arc from the beginning of the movie to the end of the movie with each run starting and ending on the function Fmovie(i), where the slope of each line (run) determines the bandwidth allocation that is required for that run (see Fig. 1). As a result, the CBA algorithm results in a plan with monotonically decreasing bandwidths [7]. While the CBA algorithm does not observe any limits in available buffer space, it does calculate the minimum necessary buffer to play the video clip with a single monotonically decreasing sequence of bandwidth allocations. The required buffer size is determined by the maximum vertical distance between the bandwidth allocation plan and the function $F_{movie}(i)$. The magnitude of this minimum buffer size may vary for the same clip, depending on the encoding scheme used and the long-term burstiness that results. Note that a plan that has a single constant bandwidth requirement would have a minimum buffer size of at least this amount but is generally much higher.

Formally, let $CB_0, CB_1, \ldots, CB_k$ be the runs created by the CBA algorithms. Then the critical bandwidth CB_0, in bytes per frame, is defined as

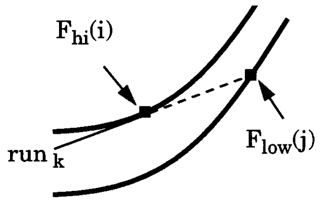
$$CB_0 = \max_{1 \leq i \leq N} \left(\frac{\sum_{j=1}^{i} frame_j}{j} \right) \ ,$$

where N is the number of frames in the video clip and $frame_j$ is the size in bytes of frame number j. Thus, the critical bandwidth is determined by the frame, i, for which the average frame size for i and all prior frames in the video clip is maximized. We call frame i, which sets the critical bandwidth, the *critical point* in the video clip, or CP_0. In the case where the maximum is achieved multiple times, we choose CP_0 to be the last frame at which it is achieved.

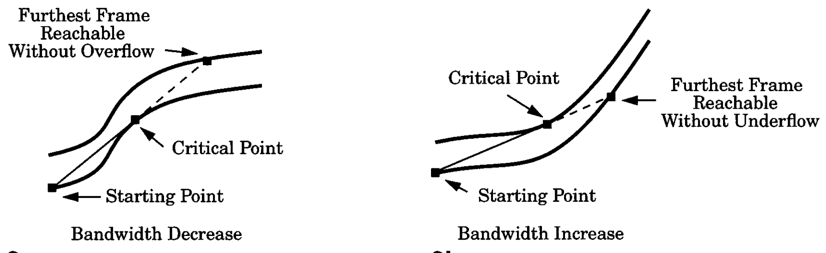
Starting at frame $CP_0 + 1$, we apply the definition of the critical bandwidth to the rest of the clip, resulting in CB_1 and



2



4



3a 3b

Fig. 2. Critical bandwidth allocation with a maximum buffer constraint. This figure shows a possible graph for $F_{hi}(i)$ and $F_{low}(i)$. The total delivered bandwidth (up to some frame i), must lie between $F_{hi}(i)$ and $F_{low}(i)$ or buffer overflow or underflow will occur. The *dotted lines* represents a critical bandwidth allocation plan that does not consider buffer overflow. Any time the allocation plan goes above $F_{hi}(i)$, buffer overflow will occur

Fig. 3a,b. Critical bandwidth and critical point example. On the *left* is an example calculation of a critical point using the maximum buffer constraint in which a bandwidth decrease is required in the next run, while on the *right* is a similar calculation for the case where a bandwidth increase is required in the following run. The *frontiers* of the runs are shown with *dashed lines*, while the hubs are represented by the *thin solid lines*

Fig. 4. Bandwidth increase search example. For the calculation of run $k+1$ a search is performed on the line (frontier) connecting $F_{hi}(i)$ and $Flow(j)$ to find a starting point such that the critical point for the next run is as far out in time as possible

CP_1. Thus, the critical bandwidths, CB_n, are determined by a sequence of critical point CP_n, where

$$CB_n = \max_{CP_{n-1}<i\le N}\left(\frac{\sum_{CP_{n-1}}^{i} frame_j}{j - CP_{n-1}}\right) .$$

The use of the CBA algorithm is an effective technique to use for systems that have appropriate amounts of buffering for several reasons. First, the playback of the video can commence immediately. Second, the admission control algorithm is simple – *Is there enough bandwidth to start the channel*? Third, these bandwidths are the minimum constant bandwidth necessary for continuous playback without requiring an increase in the bandwidth allocation. Finally, we note that is possible to reduce the beginning bandwidth requirement by prefetching data for the initial run.

2.2 Critical bandwidth allocation with maximum buffer constraint

Using the critical bandwidth algorithm results in the calculation of the minimum buffer size necessary to treat the entire video clip as a monotonically decreasing sequence of bandwidth allocations. In the event that this minimum buffer size exceeds the buffer space available, then the client must increase the bandwidth in the middle of the video clip, substituting increased network bandwidth for missing buffer bandwidth. The CBA with a maximum buffer constraint has the same properties as the CBA algorithm but increases bandwidth only when necessary. As a result, the CBA algorithm with a maximum buffer constraint (referred to from now on as the CBA algorithm), results in a plan that

1) requires no prefetching of data before playback begins
2) has the minimum number of bandwidth increase changes
3) has the smallest peak bandwidth requirement
4) has the largest minimum bandwidth requirement

Using our geometric model, we can graph the functions, $F_{hi}(i)$ and $F_{low}(i)$, where $F_{low}(i)$ is the same as $F_{movie}(i)$ from the last section and $F_{hi}(i)$ is $F_{low}(i)$ offset by the buffer size (see Fig. 2). That is,

$$F_{hi}(i) = \left(\sum_{j=1}^{i} FrameSize_j\right) + BufferSize$$

and

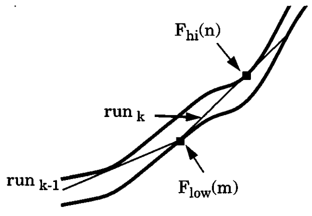
$$F_{low}(i) = \sum_{j=1}^{i} FrameSize_j .$$

Any valid plan must have the following condition hold for all frames, i, within the movie:

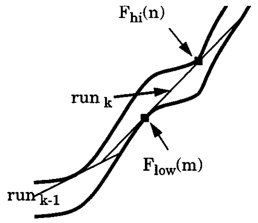
$$F_{low}(i) \le TBR(i) \le F_{hi}(i) .$$

Thus, any bandwidth allocation plan must stay between $F_{low}(i)$ and $F_{hi}(i)$ to ensure that the buffer neither overflows nor underflows in the course of playing the video. In the presence of a maximum buffer constraint, the CBA plans must be modified in the runs that violate the buffer limitations.

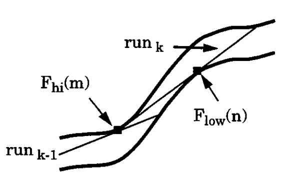
In our discussion, we modify the definition of critical points and critical bandwidths to work with a maximum buffer constraint. Given some starting point and the buffer occupancy at the starting point, the critical bandwidth is the



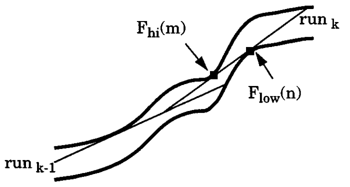
5a



5b



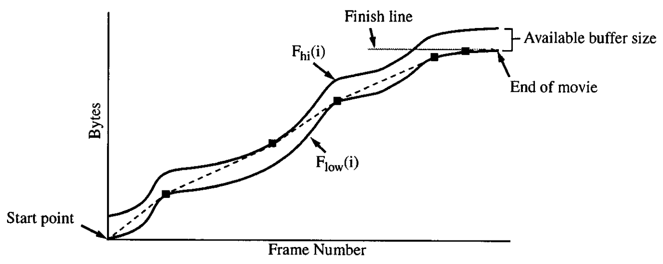
6a



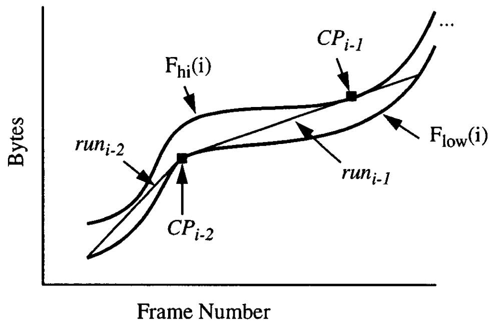
6b

Fig. 5a,b. Bandwidth search example. This figure shows the two representative cases that arise when the search for run k results in another bandwidth increase required in run $k + 1$. In (**a**), $F_{low}(m)$ lies on the line created by run $k - 1$. In (**b**), $F_{low}(m)$ does not lie on run $k - 1$

Fig. 6a,b. Bandwidth search example. This figure shows the two representative cases that arise when the search for run k results in a bandwidth decrease required in run $k + 1$. In (**a**), $F_{low}(m)$ lies on the line created by run $k - 1$. In (**b**), $F_{low}(m)$ does not lie on run $k - 1$



7



8

Fig. 7. Critical bandwidth allocation with maximum buffer example. This figure shows a sample construction of the CBA algorithm with a maximum buffer constraint. The *dotted lines* represent the runs within the bandwidth plan

Fig. 8. This figure shows a run $i-1$ that (1) requires a decrease in bandwidth from the previous run and (2) requires an increase in the next run

bandwidth such that the buffer limitations are not violated for the largest number of frames. This results in a unique run, given the initial starting conditions. Figure 3 shows two representative examples of critical points for runs that require a decrease and increase in bandwidth in the following run and have an initial buffer occupancy of 0 bytes. For a run which requires a bandwidth increase in the *next* run, the critical point is determined by a point on $F_{hi}(i)$, while for a run which requires a bandwidth decrease in the next run, the critical point is determined by a point on $F_{low}(i)$. Finally, as shown in Fig. 3, we use the term *hub* to refer to the part of the run that precedes the critical point and the term *frontier* to refer to the trajectory of the run past the critical point.

To create a bandwidth plan, the CBA algorithm starts at frame 0 with no initial buffer and calculates the critical bandwidth and critical point for the next run. Note, we can generally reduce the initially high bandwidth requirement by starting with an initial buffer size > 0, but it requires that the start playback of video to be delayed. If the critical point is on $F_{low}(i)$, then a decrease in bandwidth is required in the next run and the critical point is used as the starting point for the next run. If the critical point for the run is on $F_{hi}(i)$, then an increase in bandwidth is required in the next run. A search on the frontier of the run is performed for a starting point such that the next run results in a critical point that is as far out as possible. Thus, bandwidth decreases result in a convex arc around points on $F_{low}(i)$, while increases involve a slightly more complicated search.

For a run $k + 1$ that requires an increase in bandwidth from the last run k, the bandwidth is allocated at a slope such that the run extends as far out in time as possible. To implement this, a search is performed on the frontier of run k as shown in Fig. 4. The actual search algorithm is not of great importance in the usual case, since these searches are relatively infrequent. One can choose to implement either a linear or binary search. This search results in one of two cases (if it is not the last run in the movie): either an increase or decrease in the bandwidth allocation is required for the next run. Examples of these are shown in Figs. 5 and 6, respectively. This results in the following two bandwidth increase search properties.

Property 1. *For a run k which (1) increases the bandwidth requirement over run $k - 1$ and (2) requires an increase in bandwidth in run $k + 1$, the search for run k results in a unique run which is determined by the slope between the points $F_{low}(m)$ and $F_{hi}(n)$, where $m < n$.*

This property essentially says that the search for run k as shown in Fig. 5, which results in a bandwidth increase in run $k + 1$, is defined by two points, one from $F_{low}(i)$ and the other from $F_{hi}(i)$. In addition, because run $k + 1$ requires an increase in bandwidth, the frontier of run k must end on a point on $F_{low}(i)$.

Property 2. *For a run k which (1) increases the bandwidth requirement over run $k - 1$ and (2) requires an decrease in bandwidth in run $k + 1$, the search for run k results in a run which is determined by a slope between the points $F_{hi}(m)$ and $F_{low}(n)$, where $m < n$.*

This property is the similar result for bandwidth decreases. That is, the search in run k is defined by two points, one from $F_{hi}(i)$ and the other from $F_{low}(i)$. Figure 7 shows a sample construction using the CBA algorithm.

To recapitulate, the CBA algorithm consists of allocating runs at their critical bandwidths. For bandwidth decreases, the end of the run is set to the critical point and the next run is started on the next frame. For bandwidth increases, a search is performed on the frontier of the last run to find a starting point such that the critical point of the next run is maximized. By searching for a starting point such that the critical point is as far out in time as possible for bandwidth increases, an important theorem about the critical bandwidth algorithm can be derived.

Theorem 1. *The critical bandwidth allocation algorithm with a fixed maximum buffer constraint results in a plan for playback of video without buffer starvation or buffer overflow that has (1) the smallest number of bandwidth increases possible, (2) the smallest peak bandwidth requirement, and (3) the largest minimum bandwidth requirement.*

Proof. Let the CBA plan consist of n runs, each with a constant bandwidth allocation. We prove the above theorem by showing that all other plans (1) must have at least as many bandwidth increases, (2) cannot have a smaller peak bandwidth, and (3) cannot have a larger minimum bandwidth.

We first break the n runs into sets of consecutive runs which increase the bandwidth requirements from the previous run in the CBA plan. Let run i be the first run in each set, and let each set be numbered from i to k, $i < k$. Because run i is the first run in a set of bandwidth increases, run $i-1$ must have decreased the bandwidth over run $i-2$. This implies that run $i-2$ is determined by a critical point on $F_{low}(i)$ and that run $i-1$ starts on $F_{low}(i)$. In addition, the critical point for run $i-1$ must be on $F_{hi}(i)$. This situation is shown in Fig. 8.

We now note that because run $i-1$ connects $F_{low}(m)$ and $F_{hi}(n)$ for some $m < n$, *any other* bandwidth plan not co-linear with run $i-1$ must have a run that has a slope less than that chosen by run $i-1$, resulting in a smaller bandwidth. By showing this for all the sets of consecutive bandwidth increases, part 3 of the proof is shown. That is, any other bandwidth plan cannot have a higher minimum bandwidth than the CBA plan.

To show part 1 of the theorem (CBA results in the minimum number of bandwidth increases), we first consider run $i-1$. We note that any other plan not co-linear with run $i-1$ must have a run that crosses run $i-1$ with a lower bandwidth requirement (smaller slope). As a result, any other plan *cannot* have a run that starts on or behind the hub of run $i-1$ and cross the frontier of run $i-1$. Thus, any other bandwidth plan must also increase the bandwidth requirement *before* crossing the frontier of run $i-1$ (see Fig. 8). In the search for a run i, the CBA plan maximizes the distance reachable by run i by performing a search along the frontier of run $i-1$. Because any other bandwidth plan must increase its bandwidth before crossing the frontier of run $i-1$, it *cannot* cross the frontier created by run i, otherwise, the CBA algorithm would have found the same run in its search along the frontier of run $i-1$. Because at each step the other bandwidth plans also require an increase in bandwidth and can never pass the frontier created by that of the CBA plan, the set of consecutive increases is minimum. Applying this to all the sets of consecutive increases allows us to prove part 1 of the theorem. That is, the CBA plan results in the minimum number of bandwidth increases.

Finally, to show that the CBA results in the minimum peak bandwidth requirement, let us examine run k of each set of consecutive bandwidth increases. Because the set of runs are grouped into runs that consecutively increase the bandwidth requirements, run $k+1$ must decrease the bandwidth requirement from run k. Using Property 2, we note that the search for run k is performed along the frontier of run $k-1$, and that run k connects $F_{hi}(m)$ and $F_{low}(n)$ for some $m < n$. Because this run connects $F_{hi}(i)$ and $F_{low}(i)$, *any other* bandwidth plan not co-linear with run k must have a higher slope which crosses run k. By showing for each set of consecutive runs that other plans cannot have a minimum peak bandwidth less than the CBA plan, the CBA plan results in the minimum peak bandwidth requirement, thus, proving part 2 of the theorem.

To algebraically calculate the CBA plan requires the allocation of constant bandwidth runs. The calculation of a run requires the starting point for the run, $Frame_{start}$, and the initial buffer occupancy, $Buff_{init}$, at that starting point. To calculate a run starting from $Frame_{start}$ with initial buffer $Buff_{init}$, let

– $FrameAve_i$ be the average frame size from the beginning of the run to the ith frame within the run. This can be defined as:

$$FrameAve_i = \left(\frac{(\sum_{j=Frame_{start}}^{j+i} FrameSize_j) - Buff_{init}}{i} \right)$$

– $MaxBW_i$ be the maximum average bandwidth sustainable from the beginning of the run to the ith frame that does not overflow the buffer. This can be defined as

$$MaxBW_i = \min_{1 \le j \le i} \left(FrameAve_j + \frac{BufferSize}{j} \right) .$$

Then, the critical bandwidth for a run is defined as a set of k frames such that the following holds for all frames within the run:

$$\max_{1 \le j \le k} FrameAve_j \le MaxBW_k ,$$

and such that

$$\max_{1 \le j \le k+1} FrameAve_j > MaxBW_{k+1} .$$

The critical bandwidth for the run is then

$$CB = \max_{1 \le j \le k} FrameAve_j .$$

To calculate the critical bandwidth plan, we start with the first frame with no initial buffer and then calculate the critical bandwidth for the first run. If the critical point for the first run is along $F_{low}(i)$, then a decrease in bandwidth is necessary or the buffer will eventually overflow. The next run is then started at the critical point with initial buffer 0. If the critical point for the first run is along $F_{hi}(i)$, then a bandwidth increase is necessary for the next run. As described earlier, a search is then performed on the frontier of the run for a starting point that maximizes the point reached by the next run using the above equations. Note that this search involves a fairly trivial calculation to find the appropriate initial buffer for the next run.

Using the critical bandwidth algorithm with a fixed size buffer minimizes the number of bandwidth increases required during the playback of a video clip. In addition, it

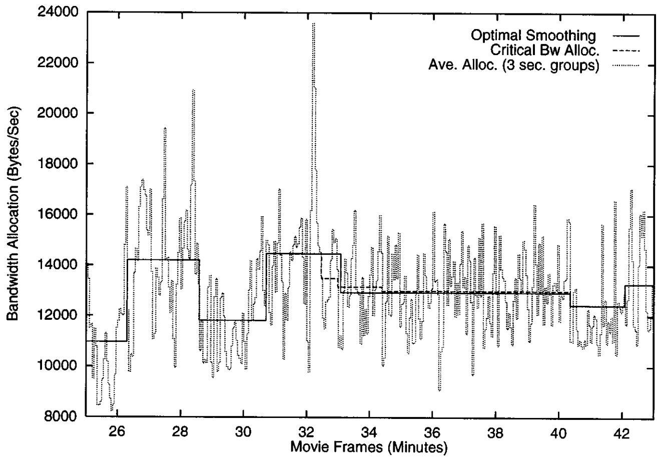


Fig. 9. This figure shows the same sample clip from the movie *Speed*. The *solid line* shows the bandwidth allocation plan using the OBA algorithm and a 2-MB buffer, while the *heavier dotted line* shows the CBA algorithm. The main difference between the algorithms is that the OBA algorithm combines all the bandwidth decreases into a few request

also has the smallest peak bandwidth requirement and the largest minimum bandwidth requirement. An example of CBA smoothing can be found in Fig. 9.

2.3 An optimal bandwidth allocation algorithm

Using the critical-bandwidth-based algorithm, it is possible to minimize the total number of bandwidth increases for the continuous playback of video. The CBA plans, however, may require many adjustments that decrease the bandwidth requirement. For networks that place a premium on interacting as little with the clients as possible, the CBA can be extended to have all the properties from Theorem 1 while also minimizing the total number of bandwidth changes required. The *optimal bandwidth allocation* (OBA) algorithm results in the same number of increases in bandwidth, the same smallest peak bandwidth, and the same largest minimum bandwidth as the CBA algorithm. The OBA algorithm differs from the CBA algorithm by not returning bandwidth to the network as soon as it has passed the critical point that required the bandwidth. Instead, the OBA algorithm may hold the bandwidth past the critical point for a run in order to reduce the number of decreases in bandwidth required, and, hence, minimizes the interactions with both the network and server. As a result of this, the OBA algorithm has very few changes in bandwidth for a moderately sized buffer. In the rest of this section, we describe the OBA strategy for the delivery of prerecorded video. We continue to use the definitions of critical bandwidths and critical points stated in the last section.

For our geometric model, the OBA algorithm allocates runs by performing a search on the frontier of each run such that the critical point for the next run is maximized. As a result, the OBA algorithm attempts to allocate runs such that each run maximizes the point reachable. At the end of a particular line (run), there are two possibilities for the next run, either increase or decrease the bandwidth requirement. For our discussion, we ignore a run which can reach the end of the movie. The actual bandwidth used for the last run can be chosen so that it falls within the range of bandwidth allocations already used, or it can be chosen in such a way as to minimize the bandwidth or minimize the allocation time of the channel.

For runs which require a bandwidth increase in the next run, the same search is performed as in the CBA algorithm, resulting in a search on a line connecting $F_{hi}(m)$ and $F_{low}(n)$, with $m < n$ (see Figs. 5 and 6). For runs which require a bandwidth decrease in the next run, a search at the end of the current run results in four other possible outcomes. These representative outcomes, which are essentially mirror images of the four bandwidth increase cases, are shown in Figs. 10 and 11. For a run k which decreases the bandwidth from the last run, a search along a line that touches $F_{low}(m)$ and $F_{hi}(n)$, with $m < n$ is performed to find a starting point for the next run that maximizes the point reachable for run k. This new line segment maximizes the critical point for the next run, while providing a transition from the last run to the current run. This leads to two bandwidth decrease search properties that are parallel to Properties 1 and 2.

Property 3. For a run k which (1) decreases the bandwidth requirement over run $k-1$ and (2) requires an increase in bandwidth in run $k+1$, the search for run k results in a run which is determined by the slope between the points $F_{low}(m)$ and $F_{hi}(n)$, where $m < n$.

This property essentially says that the search for run k as shown in Fig. 10, which results in a bandwidth increase in run $k+1$, is defined by two points, one from $F_{low}(i)$ and the other from $F_{hi}(i)$. In addition, because run $k+1$ requires an increase in bandwidth, the frontier of run k must end on a point on $F_{low}(i)$.

Property 4. *For a run k which (1) decreases the bandwidth requirement over run $k-1$ and (2) requires an decrease in bandwidth in run $k+1$, the search for run k results in a run which is determined by a slope between the points $F_{hi}(m)$ and $F_{low}(n)$, where $m < n$.*

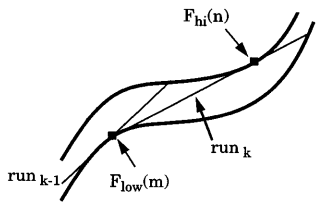
This property is the similar result for bandwidth decreases. That is, the search in run k is defined by two points, one from $F_{hi}(i)$ and the other from $F_{low}(i)$. See Fig. 11. A sample construction is shown in Fig. 12. Using this "greedy" approach in the allocation of each run within the OBA plan results in the following theorem:

Theorem 2. *For video playback allocation plans using a fixed size buffer, for which (a) the bytes deliverable are equal to the aggregate size of the video clip and (b) where prefetching at the start of the movie are disallowed, the OBA algorithm results in (1) the smallest peak bandwidth, (2) the largest minimum bandwidth, and (3) the fewest possible bandwidth changes.*

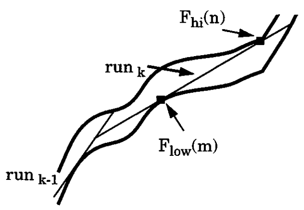
Proof. To prove this theorem, we use the notation

[inc, inc] – for a run which increases the bandwidth from the last run and requires an increase in bandwidth in the next run

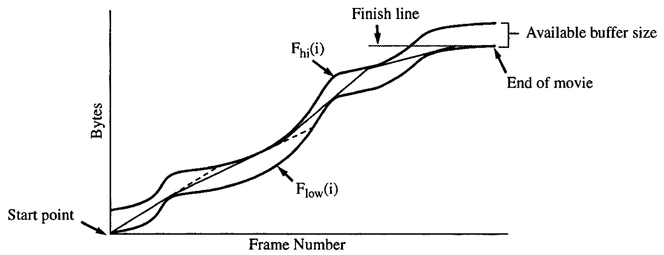
[inc, dec] – for a run which increases the bandwidth from the last run and requires a decrease in bandwidth in the next run



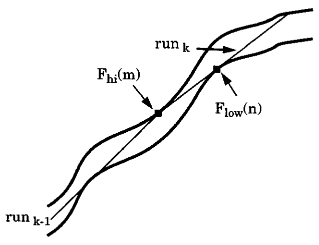
10a



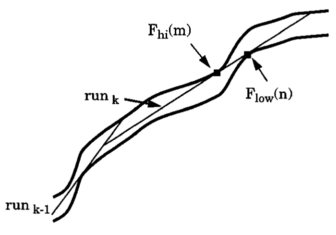
10b



12



11a



11b

Fig. 10a,b. Bandwidth search example. This figure shows the two representative cases that may result for run k which decreases the bandwidth requirement from run $k - 1$, but requires a bandwidth increase for run $k + 1$

Fig. 11a,b. Bandwidth search example. This figure shows the two representative cases that arise when the search for run k results in another bandwidth decrease required in run $k + 1$. In (**a**), $F_{hi}(m)$ lies on the frontier created by run $k - 1$. In (**b**), $F_{hi}(m)$ does not lie on the frontier of run $k - 1$

Fig. 12. Optimal bandwidth allocation construction example. This figure shows a sample construction of the optimal bandwidth allocation algorithm. Note, this plan includes only four runs, while the same plan in Fig. 7 required six runs using the CBA algorithm. The *heavy solid lines* show $F_{hi}(i)$ and $F_{low}(i)$, while the *light solid lines* show the slopes (bandwidths) selected by the optimal critical bandwidth allocation algorithm. The *dotted lines* show the lines along which the searches were performed to maximize the critical points of the following runs

[dec, inc] – for a run which decreases the bandwidth from the last run and requires an increase in bandwidth in the next run

[dec, dec] – for a run which decreases the bandwidth from the last run and requires a decrease in bandwidth in the next run

To prove part 1 of the theorem (smallest peak bandwidth), let us consider all of the [*inc, dec*] runs within the OBA plan. That is, the runs that are surrounded by runs of smaller bandwidths. Let the [*inc, dec*] run be run i. By Property 2, run i is determined by a hub that runs from $F_{hi}(m)$ to $F_{low}(n)$ for some $m < n$. We then note that any other plan that is not co-linear with run i, must have a run that crosses the hub of run i. Because this slope must be greater than that from $F_{hi}(m)$ to $F_{low}(n)$ in order to cross it, no other run can have a smaller bandwidth requirement that crosses the hub of run i.

To prove part 2 of the theorem, the mirror of part 1 is used. Let us consider all of the [*dec, inc*] runs within the OBA plan. That is, the runs surrounded by runs of larger bandwidths. Let the [*dec, inc*] run be run i. By Property 3, run i is determined by a hub that runs from $F_{low}(m)$ to $F_{hi}(n)$ for some $m < n$. We then note that any other plan that is not co-linear with run i, must have a run that crosses the hub of run i. Because this slope must be smaller than that from $F_{low}(m)$ to $F_{hi}(n)$ in order to cross it, no other run can have a larger bandwidth requirement that crosses the hub of run i.

To prove part 3 of the theorem, we show by contradiction that the OBA algorithm results in the minimum number of bandwidth changes. Suppose the OBA algorithm creates a bandwidth plan, $plan_{opt}$, that has X bandwidth changes in it. Further, suppose that this plan is not optimal in the number of bandwidth changes. Therefore, another plan, $plan_{better}$, must exist that has fewer than X bandwidth changes in it. As a result, there must exist at least one run in $plan_{better}$ that spans greater than one run from $plan_{opt}$. As will be shown, this cannot happen.

For algorithms that do not allow prefetching, both bandwidth plans must start on the first frame and have nothing in the smoothing buffer. As a result, $plan_{opt}$, whether it requires an increase or decrease in bandwidth in the next run, results in a plan that has a critical point greater than or equal to the first run in $plan_{better}$. If an increase in bandwidth is required in the next run, $plan_{opt}$ picks the bandwidth such that any more bandwidth would result in buffer overflow before the critical point of the first run in $plan_{opt}$. Any less bandwidth results in a critical point that is before the critical point of the first run in $plan_{opt}$. If a decrease in bandwidth is required in the next run, then by definition, $plan_{opt}$ has chosen the minimal bandwidth necessary without overflow resulting in the furthest critical point possible. Thus, $plan_{better}$ cannot have a critical point that is further out than $plan_{opt}$ for the first run, and hence, cannot cross the frontier of the first run in $plan_{opt}$ in the first run.

For each run after the first run, $plan_{opt}$ starts by examining the frontier of the last run and finds a starting frame that maximizes the critical point of the current run. This search is always performed a line connecting $F_{low}(i)$ and $F_{hi}(i)$ OR $F_{hi}(i)$ and $F_{low}(i)$. Because this search is on a line that connects Fhi and Flow which $plan_{better}$ must cross, $plan_{better}$ cannot pick a next run that is longer than the one chosen by $plan_{opt}$. Otherwise, $plan_{opt}$ would have found it in its search. We continue this process for all runs within the

$plan_{opt}$. Because every ith run in $plan_{better}$ cannot have a critical point further than the ith run in $plan_{opt}$, $plan_{better}$ must have at least as many runs as $plan_{opt}$. Therefore, $plan_{opt}$ results in the fewest number of bandwidth changes.

In order to algebraically construct the OBA plan, we use the same algorithm for finding the critical bandwidth and critical point for a run as defined in Sect. 2.2. The OBA plan then consists of three types of allocations: the beginning run, a run that decreases the bandwidth allocation, and a run that increases the bandwidth allocation.

The beginning run does not have any prefetch in order to minimize the latency between channel set-up and the beginning of playback. Therefore, the first run, is set to the critical bandwidth and critical point for the run starting at the beginning of the movie with an initial buffer of 0. Next, if the critical point for the run lies on $F_{low}(i)$ a bandwidth decrease is required in the second run. If the critical point for the run lies on $F_{hi}(i)$ a bandwidth increase is required in the second run.

In the calculation of a run that decreases the bandwidth allocation, a search on the frontier of the previous run is performed to determine how long the bandwidth should be held past the previous run's critical point (See Figs. 10 and 11). The search finds a frame, j, such that using the same bandwidth allocation from the end of the last run results in the critical point in the current run to be as far out as possible. The bandwidth for the run is then set to the critical bandwidth of the last run up to, and including, frame j, while the bandwidth from frame $j + 1$ to the critical point is set to the critical bandwidth for the run starting on frame $j + 1$, with the appropriate initial buffer.

In the calculation for a run that increases the bandwidth allocation, a search on the frontier of the previous run is performed to find a frame, k, such that the current run reaches as far a frame as possible. The current run is then started on frame k and has its bandwidth allocation set to the critical bandwidth starting from frame k.

For each subsequent run, we simply apply the same algorithm to determine which of the calculations to use (whether for increasing or decreasing the bandwidth). A sample allocation plan is shown in Fig. 9.

3 Evaluation of algorithms

From the point of view of network and server management, load estimation and admission control are crucial to providing guarantees of service. These can be greatly simplified if all channels exhibit constant behavior. In the absence of an entirely constant bandwidth allocation, the amount each channel strays from this constant allocation will determine the network's performance. Several measures that influence this performance are the frequency of requests for increased bandwidth, the size of these increase, and the peak bandwidth requirements. The frequency and size of decreases can be interesting as well if the network management makes some provision for lowering a bandwidth reservation.

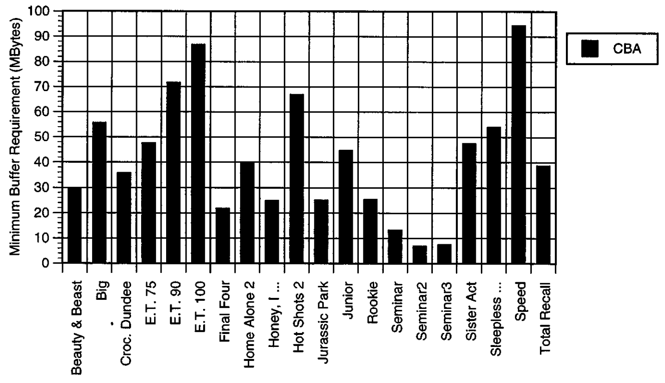
To compare and contrast the differences between the critical bandwidth allocation based approaches, we digitized and Motion-JPEG compressed 17 full-length movies and 3 seminars presented in our department to use as test data. Our experiments confirmed our theoretical hypothesis and showed that for a small amount of client-side buffering we can indeed reduce the number of bandwidth changes necessary to a small number. To our surprise, we found that having 20 MB of buffer space on the client-side could reduce the total number of changes for any of the clips tested to less than 10. We expect that with tighter video encodings that take advantage of temporal redundancy between frames, such as MPEG, that similar numbers can be achieved with smaller buffers. In the rest of this section, we describe our experimental set-up and the video clips that were digitized. In addition, a more in-depth look at the performance of the critical bandwidth allocation based algorithms is presented.

3.1 Experimental set-up

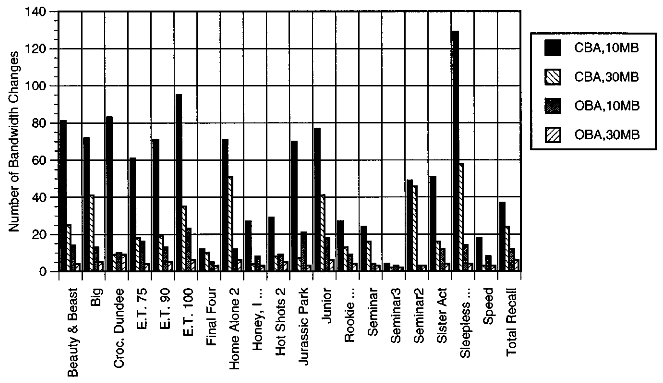
To test the effectiveness of the various smoothing algorithms, we needed to digitize a few full-length movies. In previous work, we found that using the critical bandwidth allocation approach on smaller clips created plans that required no increases in bandwidth and required small amounts of buffering to achieve this. To get a better representation of how the algorithms perform, we needed to capture a larger set of full-length movies.

Our PC testbed consists of a Pioneer Laser Disc player, a MiroVideo DC1tv capture board, and a Pentium 90 processor with 32 MB of memory. The MiroVideo Capture board is a Motion-JPEG compression board. We do not have the equipment to perform rapid MPEG encodings. Because the basic routine for encoding I-frames within an MPEG video are derived from the JPEG compression standard, the frame sizes for our experimental video data are roughly equivalent to all I-frame encoded MPEG video movies. The CBA and OBA algorithms are most sensitive to scene content changes and not pattern burstiness, however, the size of the resulting streams strongly affects the buffer requirements. MPEG encoded video could achieve the same performance with buffers that are approximately 4-6 times smaller. We, therefore, expect that the results presented here are somewhat conservative compared to a system using MPEG as its compression standard. The Miro Video board digitized the movies at 640×480 and then subsampled them to 320×240 with guaranteed VHS picture quality.

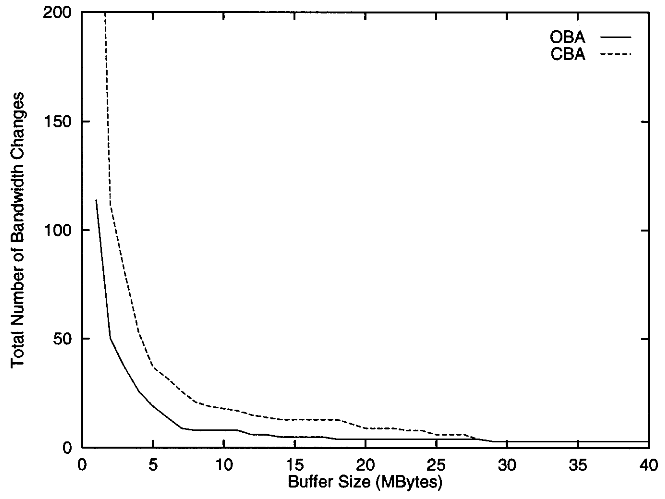
Using our testbed, we captured 17 full-length movies at a range of 0.85–1.61 bits per pixel. The statistics for these videos are shown in Table 1. In digitizing the video data, we attempted to capture a variety of different movies in order to examine the effects each had on the smoothing algorithms. The *Beauty and the Beast* video is an animated Walt Disney movie, resulting in scenes with a lot of high-frequency components as well as scenes that had large areas of constant color. The *1993 NCAA Final Four* video is a documentary describing the *NCAA Final Four* basketball tournament, resulting in many of the scenes with lots of detail. As a result, the *1993 NCAA Final Four* video had the highest average bit rate. The rest of the movies are a mix of conventional entertainment containing a wide range of scene content, including digital effects and animations. The *Seminar* videos, as previously mentioned, contain single scenes and, thus, have the smallest variation in frame sizes.



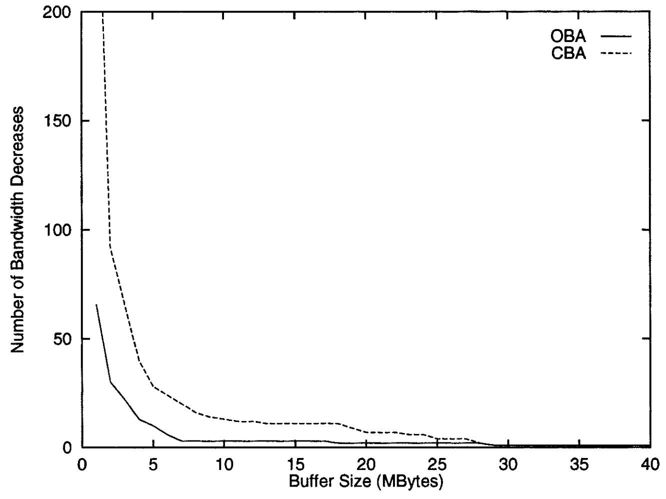
13



15



14



16

Fig. 13. Critical bandwidth allocation minimum buffer requirement. This figure shows the maximum buffer requirements for the movies encoded using the CBA algorithm with no buffer limitation

Fig. 14. Bandwidth change requests. The graphs show the total number of required bandwidth allocation change requests for the movie *Speed*. The OBA and CBA algorithms were run on the entire video clip for varying buffer sizes in 1-MB increments

Fig. 15. Bandwidth changes for all movies. This graph show the number of bandwidth changes required for all of the sample movies using a 10- and 30-MB buffer and the OBA and CBA algorithms

Fig. 16. Bandwidth decrease requests. The graphs show the total number of bandwidth allocation decrease requests for the movie *Speed*. The CBA and OBA algorithms were applied to the movie segment with different buffer capacities

3.2 Critical bandwidth allocation without buffer constraint

When sufficient buffering is available, allocating the bandwidth plan using the critical bandwidth algorithm without a buffer constraint is useful because, in a system where all streams are using this algorithm, admission control becomes trivial. The admission control algorithm simply sees if there is enough bandwidth to start the flow of data. Recall that the critical bandwidth algorithm results in a monotonically decreasing sequence of bandwidth allocations. In addition, the playback of the video can start once the video has been accepted to the network.

As shown in Fig. 13, the amount of buffer required to play back the sample videos varies quite a bit. The total amount of buffering depends primarily on three factors, the average size of the frames, the length of the video, and the long-term burstiness of the video. If the movie has a sustained area of smaller frames followed by a sustained area of larger frames, the amount of buffering tends to be much higher. As examples, the seminar videos require much smaller buffer sizes, because both the size and variation of frames sizes are small in these videos. The *E.T.* videos require proportionately larger buffers as the quality is increased. Because these videos exhibit the same long-term burstiness, the differences are due mostly to the increase in the frame sizes within the videos. Just the average size of frames, however, is not indicative of the minimum buffer requirements. The movie *Speed* requires a larger buffer than the *E.T.* (Quality 100) video even though both the variance in frame sizes and the average frame sizes are smaller in the *Speed* video. Thus, the buffer size is primarily due to the long-term burstiness, and to a lesser degree, the length and average frame sizes of the videos.

Table 1. This table shows the statistics of the Motion-JPEG video clips that were digitized with the MiroVideo capture board and used in the evaluation of the critical bandwidth algorithms

Video clip name	Quality	Length (min)	Ave. Bit Rate (Mbits/s)	Max frame size (bytes)	Min frame size (bytes)	Std. dev. (bytes)
Beauty and Beast	90	80	3.04	30367	2701	3580
Big	90	102	2.96	23485	1503	2366
Crocodile Dundee	90	94	2.59	19439	1263	2336
ET 75	75	110	1.51	14269	1153	1840
ET 90	90	110	2.17	19961	2333	2574
ET100	100	110	3.78	30553	6827	3294
Home Alone II	90	115	2.73	22009	3583	2480
Honey, I Blew Up the Kid	90	85	3.32	23291	3789	3183
Hot Shots, Part Duex	90	84	3.06	29933	3379	3240
Jurassic Park	90	122	2.73	23883	1267	3252
Junior	90	107	3.36	25119	1197	3188
Rookie of the Year	90	99	2.98	27877	3531	2731
Seminar	90	63	2.07	10977	7181	592
Seminar2	90	68	2.12	12309	1103	608
Seminar3	90	52	2.26	11167	7152	690
Sister Act	90	96	2.86	24907	1457	2608
Sleepless In Seattle	90	101	2.28	16617	3207	2459
Speed	90	110	2.97	29485	2741	2707
Total Recall	90	109	2.88	24769	2741	2692
1993 Final Four Video	90	41	3.95	29565	2565	4138

3.3 Bandwidth changes

As Fig. 14 shows, the OBA algorithm results in a fewer number of bandwidth changes than the CBA algorithm for a given buffer size, as expected. As an example, using the *Speed* video, a 5-MB smoothing buffer, and the OBA algorithm results in 21 bandwidth changes over the 110-min movie, while the critical bandwidth algorithm requires 37 changes in bandwidth. On average the optimal bandwidth allocation algorithm requires a bandwidth change approximately every 5 min. After the initial start of the movie, the *Speed* movie using the OBA algorithm has a minimum run length of approximately 1 min and 45 s and a maximum run length of approximately 14 min. By using a 10-MB buffer for buffering, the shortest and longest run lengths grow to 13 min and 25.5 min, respectively. As a result, a modest amount of buffering can reduce the number of interactions required from the network to the order of tens of minutes.

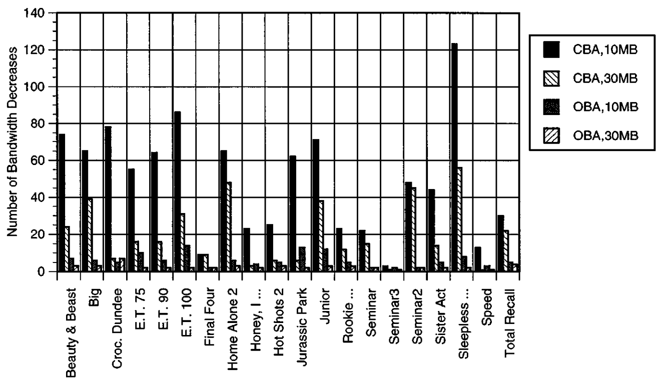
As shown in Fig. 15, the total number of bandwidth changes required for the rest of the video data are relatively small even for a 10-MB buffer with loosely encoded video. For a 10-MB smoothing buffer, the movie *E.T.* (Quality 100) requires 23 bandwidth changes with the OBA algorithm, which is the maximum number of changes required for all the movies using the OBA algorithm. On average, the OBA algorithm results in 73% fewer bandwidth changes than the CBA algorithm for a 10-MB smoothing buffer and 63% fewer bandwidth changes at 30 MB. The distinction between increases and decreases in the bandwidth allocation plan can be useful, because the requests for decreases in bandwidth can generally be satisfied, while increases may require further negotiations with the network. In addition, it highlights the main differences between the various algorithms.

3.4 Bandwidth decrease requests

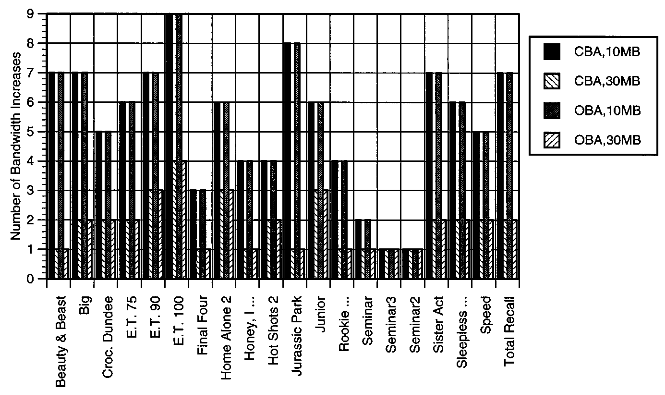
As shown in Fig. 16, the total number of bandwidth decreases for the *Speed* video are similar to the total number of bandwidth change graph (Fig. 14). This is not entirely unexpected because the CBA and OBA algorithms result in the minimum number of bandwidth increases necessary for continuous playback. Thus, a large percentage of the bandwidth changes are due to decreases in bandwidth, which from a network point of view should be easier to satisfy. In comparing the optimal bandwidth algorithm with the critical bandwidth algorithm, we see that the main difference between these algorithms is in the number of bandwidth decreases (as shown by the same relative differences in total bandwidth changes and total number of decreases). The CBA algorithm allocates each run at the minimum bandwidth requirement to avoid underflow, while the optimal bandwidth starts each run at the minimum bandwidth requirement but holds the bandwidth past the critical point of the run to prefetch data for the next run.

3.5 Bandwidth increase requests

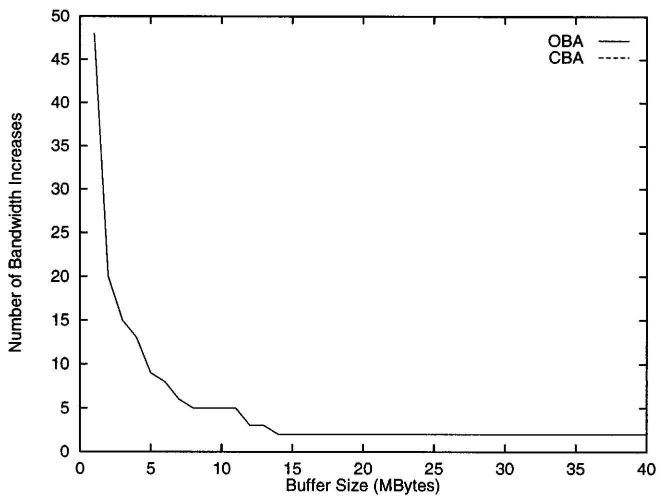
For bandwidth increases, using the CBA and OBA algorithms results in the same number of increases across all buffer size constraints for each movie, as expected. As shown in Fig. 18, the number of increases required for the movie *Speed* drops to five increases for buffers greater than 8 MB and drops to only two increases for buffers greater than 14 MB. As a result, interactions with the network for more bandwidth are required, on average, every 21 and 55 min for an 8- and 14-MB smoothing buffer, respectively! As shown in Fig. 19, all the other movies exhibit similar behavior to the *Speed* video.



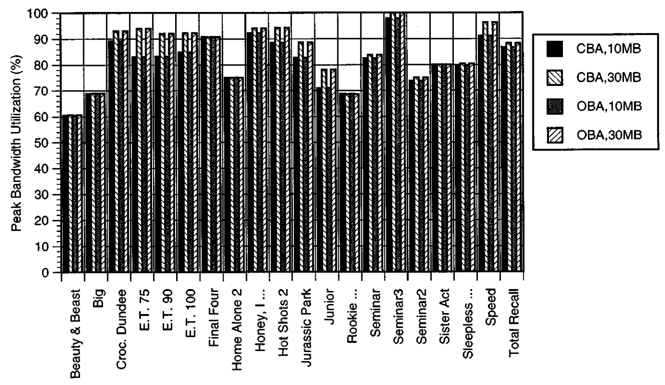
17



19



18



20

Fig. 17. Bandwidth decreases for all movies. This graph show the number of bandwidth decreases required by the OBA and CBA algorithms for 10- and 30-MB buffers for all the sample movies

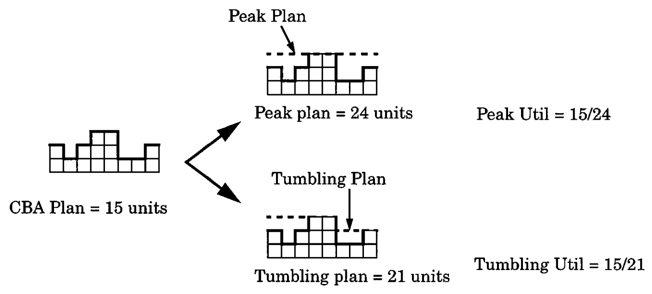
Fig. 18. Bandwidth increase requests. The graph shows the total number of bandwidth allocation increase requests for the movie *Speed*. The CBA and OBA algorithms were applied to the video with different buffer capacities, all resulting in the same number of bandwidth increases

Fig. 19. Bandwidth increases for all movies. This graph shows the number of bandwidth increases required using the OBA and CBA algorithms with a 10-MB and 30-MB buffer

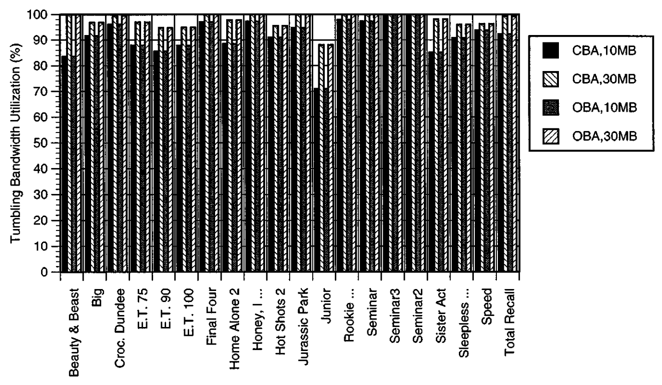
Fig. 20. Peak bandwidth utilization for aAll movies. This figure shows the utilization of resources allocated using the peak bandwidth requirement and the OBA and CBA algorithms (with no initial prefetching at the start of the movie)

3.6 Peak bandwidth requirements

For systems that allocate resources based on the peak bandwidth requirements, the peak bandwidth requirement can be an important measure. While the peak bandwidth requirement gives the amount of resources necessary, it does not give a method for comparing the results of different movies. To show the effects of smoothing on our sample movies, we have assumed that the peak bandwidth requirement is used for the entire movie and then calculated the utilization of the bandwidth reserved. The peak bandwidth utilization measurements are shown in Fig. 20. From this graph, we see that some movies (particularly the ones with low utilization) do not improve their utilization of bandwidth between 10 MB and 30 MB of buffering. The main reason for this is that the optimal bandwidth algorithm may have an initially high bandwidth requirement in order to satisfy a low-latency start of the video, however, once this initially high bandwidth requirement is passed, a lower peak bandwidth requirement results for the rest of the movie. To show this, we have also graphed what we call the *tumbling utilization* measurement, which measures the utilization of the bandwidth reserved in the same way as the peak bandwidth utilization measurement, with one exception. Once the peak bandwidth requirement has passed for the entire movie, the bandwidth can be reduced to the peak bandwidth for the rest of the movie. An example of the tumbling bandwidth utilization calculation is shown in Fig. 21. As Fig. 22 shows, the tumbling bandwidth utilizations are much higher than the peak bandwidth utilization measurements, mostly due to the initially high bandwidth requirement of the OBA plans. The movie *Junior* has the lowest utilization of all the movies. This movie has a large burst of frames at the end of the movie, resulting in the peak bandwidth requirement at the end of the movie, resulting in a lower utilization than exhibited by the other movies. With the exception of *Junior*,



21



22

Fig. 21. Peak utilization vs. tumbling utilization. This figure shows the difference between the peak and tumbling utilization calculations. The CBA plan (*heavy solid line*) requires 15 units of bandwidth (represented by *squares*). The tumbling utilization plan always results in a utilization greater than or equal to the peak utilization

Fig. 22. Tumbling bandwidth utilization for all movies. This figure shows the tumbling utilization for the OBA and CBA algorithms on the sample movies. The tumbling utilization is the same as the peak bandwidth requirement utilization, except once the peak bandwidth requirement is passed, a channel may reduce its peak bandwidth requirement once the peak has passed

it is interesting to note that the optimal bandwidth algorithm result in utilizations between 94% and 100%, with 10 of the movies having greater than 99% utilization.

4 Summary and conclusions

The smoothing of video data will play an important role in the design of video playback systems. Smoothing of compressed video data through prefetching allows the data delivery service to substitute buffer bandwidth for network bandwidth. By smoothing the rate of video transmission through buffering, network and server scheduling can be simplified. In this paper, we have proven that the CBA algorithm results in plans for the continuous playback of stored video that have (1) the minimum number of bandwidth increases, (2) the smallest peak bandwidth requirements, and (3) the largest minimum bandwidth requirements. We have also introduced the notion of the OBA algorithm which is based on the critical bandwidth algorithm. As we have shown, this OBA algorithm, in addition to having the three CBA properties, also minimizes the total number of bandwidth changes required, making the best use of the available buffer bandwidth.

Many live-video applications have fairly consistent bandwidth requirements across time, although compression will lead to pattern burstiness at a small scale. This sort of video stream can be smoothed by an algorithm with a narrow window [8]. On the other hand, streams such as stored movies are inhomogeneous at the level of scenes within the movie. To effectively smooth such streams, it is necessary to examine longer segments within the stream. This is the approach taken by the CBA and the OBA algorithms. Both algorithms aggressively prefetch data to smooth network bandwidth requirements and differ on the approach used in returning bandwidth back to the network. The critical bandwidth algorithm calculates a bandwidth plan such that the bandwidth is returned as soon as the critical point that forced the bandwidth is passed. The OBA algorithm, on the other hand, continues to hold the bandwidth until the optimal amount of prefetch for the next run is obtained, minimizing the number of interactions necessary with the network and server.

The choice between using the OBA algorithm and the CBA algorithm depends on the network cost model. In some cases, it may be more beneficial for the network to have the clients allocate at their minimum constant bandwidth requirements, making the critical bandwidth algorithm more useful. In other cases, it may be more beneficial for the network to have as few interactions with the video application as possible, in which case, the OBA algorithm may be the appropriate choice.

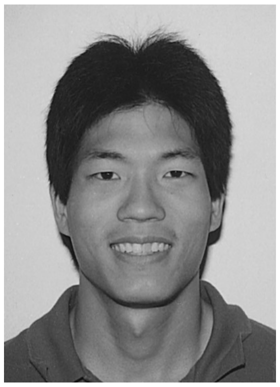
The amount of buffering needed in any system depends upon the size of the data stream and the effectiveness of encoding. The video clips used in our analysis do not take advantage of inter-frame dependencies, however, these inter-frame dependencies tend to reduce the number of bandwidth changes required as long as the pattern of frame types is repeating. Nonetheless, our results are indicative of the resources required for smooth video delivery. These requirements are not large in terms of today's workstations and should be achievable for tomorrow's televisions.

In the playback of video, VCR functions such as stop, pause, rewind, and fast-forward may be required. For accesses that occur around the point of playback such as pause or a short rewind, the buffer can be used to service some of the requests. For these local requests, the only change necessary to the bandwidth plan is that the reservations must be moved forward in time. Because the CBA-based algorithms have nearly constant bandwidth allocations (as exhibited by their 90+be efficiently handled. For accesses that are more random, the use of contingency channels can be used to return the user to the originally agreed upon bandwidth reservation level [3]. A more in-depth handling of VCR functions in bandwidth smoothing environments can be found in [4] and [5].

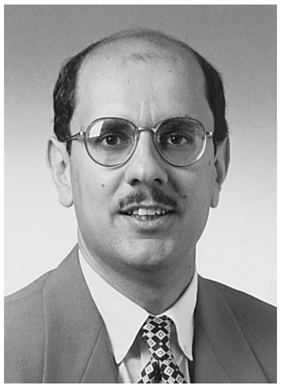
Acknowledgements. This work was supported by the National Science Foundation under grant CCR-9502341.

References

1. CCITT Recommendation MPEG-1 (1993) Coded Representation of Picture, Audio, and Multimedia/Hypermedia Information. ISO/IEC 11172, Geneva, Switzerland
2. Cohen DM, Heyman DP (1993) A simulation study of video teleconferencing traffic in ATM networks. In: IEEE INFOCOM 1993, San Francisco, California, pp 894–901
3. Dan A, Shahabuddin P, Sitaram D, Towsley D (1994) Channel allocation under batching and vcr control in movie-on-demand servers. IBM Research Report RC19588, Yorktown Heights, NY
4. Feng W, Jahanian F, Sechrest S (1995) Providing VCR functionality in a constant quality video-on-demand transportation service. CSE-TechReport 271-95, University of Michigan
5. Feng W, Jahanian F, Sechrest S (1996) Providing VCR functionality in a constant quality video-on-demand transportation service. In: 3rd IEEE International Conference on Multimedia Computing and Systems, Hiroshima, Japan
6. Feng W, Sechrest S (1995) Smoothing and buffering for delivery of prerecorded compressed video. In: Proceedings IS&T/SPIE Multimedia Computing and Networking, San Jose, Calif., pp 234–242
7. Feng W, Sechrest S (1995) Critical bandwidth allocation for the delivery of compressed prerecorded video. Comput Commun 18: 709–717
8. Lam S, Chow S, Yau D (1994) An algorithm for lossless smoothing of MPEG video. In: ACM SIGCOMM Conference Proceedings, London
9. LeGall DJ (1992) A video compression standard for multimedia applications. Commun ACM 34(4): 46–58
10. Pancha P, El Zarki M (1992) Prioritized transmission of variable bit rate MPEG video. In: IEEE GLOBECOM 1992 Proceedings. Orlando, Florida, pp 1135–1139
11. Reininger D, Raychaudhuri D, Melamed B, Sengupta B, Hill J (1993) Statistical multiplexing of VBR MPEG compressed video on ATM networks. In: IEEE INFOCOM 1993, San Francisco, California, pp 919–926
12. Stone D, Jeffay K (1993) Queue monitoring: a delay jitter management policy. In: Proceedings of the 4th International Workshop on Network and OS Support for Digital Audio and Video. Heidelberg, pp. 149–160
13. Zhang H, Ferrari D, Verma DC (1992) Delay jitter control for real-time communications in a packet switching network. Comput Commun 15(6): 367–373



Wu-chi Feng received his B.S. degree in Computer Engineering from the Pennsylvania State University in 1990. He received his M.S. and Ph.D. degrees from the University of Michigan in Computer Science and Engineering in 1992 and 1996, respectively. He is currently a faculty member in the Department of Computer and Information Science at the Ohio State University. His research interests include multimedia computing and networking, and operating system support for multimedia.



Farnam Jahanian received the M.S. and Ph.D. degrees in Computer Science from the University of Texas at Austin in 1987 and 1989, respectively. He is currently a faculty member in the Department of Electrical Engineering and Computer Science at the University of Michigan. Prior to joining the faculty at the University of Michigan in 1993, he had been a Research Staff Member at the IBM T.J. Watson Research Center where he led several experimental projects in distributed and fault-tolerant systems. His current research interests include real-time software systems, fault-tolerant distributed computing, and protocols and tools for wide-area collaborative environments.