On the efficacy of quality, frame rate, and buffer management for video streaming across best-effort networks

Wu-chi Feng
Department of Computer Science and Engineering, OGI School of Science and Engineering, Beaverton, OR 97006, USA
E-mail: wuchi@cse.ogi.edu

Abstract. In this paper, we propose a mechanism that supports the high quality streaming and adaptation of stored, constant-quality video across best-effort networks. The difficulty in the delivery of such traffic over best-effort networks is the fact that both the bandwidth required from the stream and the bandwidth available across the network vary considerably over time. Our proposed approach has a number of useful features. First, it uses the *a priori* information from the video stream to drive that adaptation policy. Second, it does not rely on any special handling or support from the network itself, although any additional support from the network will indirectly help increase the video quality. Third, it can be built on top of TCP, which effectively separates the adaptation and streaming from the transport protocol. This makes the question of TCP-friendliness much easier to answer. Using actual MPEG encoded video data at multiple qualities, we show through experimentation that this approach provides a viable alternative for streaming media across best-effort networks.

1. Introduction

With the rapid advances in the ‘last mile’ networking bandwidth, such as the digital subscriber loop (DSL) and cable modems, the ability to achieve higher quality video networking is becoming more feasible than ever. As a result of moving from 28.8 kbps or 56 kbps coded streams to megabits per second, streaming of higher quality MPEG video is now possible. As the bit-rates for the video streams increases, the bandwidth requirements for the stream become more diverse over time, requiring more intelligent buffering and rate adaptation algorithms to be put in place.

One of the key issues in creating highly scalable networks is the notion of TCP-friendliness, which measures how fairly transport-layer flows are sharing network bandwidth [1]. While various definitions of fairness have been proposed, the key idea is that all the flows share the bandwidth and that they avoid sending the network into congestion collapse. The manifestation of TCP-friendliness in video applications has resulted in techniques such as congestion-sensitive UDP-based flows [2,3]. TCP-flows without retransmission [4], or limited retransmission UDP-flows [5].

Another key design issue is how video streams can be adapted to fit within the available resources of the network. For video conferencing applications and live video applications, the adaptation typically occurs along either adapting the quality of the video, adapting the frame rate of the video, or using a mixture of the two [6,7]. For the delivery of stored video streams, the adaptation really occurs over three parameters: quality, frame rate, and buffering (greater than that needed to remove delay jitter). All of these can be used in isolation or combined together, however, efficient techniques need to be developed for the algorithms to deliver the highest quality video over the network.

In this paper, we propose a system for the delivery of stored video streams across best-effort networks that has a number of unique properties. First, it uses the *a priori* information available from the video (i.e., the frame information) in the decision of streaming. Previous work in this area (including our own) has always used the

0926-6801/02/$8.00 © 2002 – IOS Press. All rights reserved

a priori information in limited ways. Second, it can use the TCP protocol as the underlying transport mechanism. Thus, the question of TCP friendliness is not a problem. In this work, we will also describe a metric we call the *effective frame rate* measure that more effectively captures the smoothness of streaming media. To show the efficacy of our proposed approach, we have conducted simulations of the algorithms and have implemented the algorithms for MPEG streaming applications. Our results show that we can effectively combine frame rate, frame quality, and buffering into a single coherent framework.

In the rest of the paper, we will first describe the background and related work necessary for the rest of the paper. In Section 3, we describe our proposed approach as well as an evaluation metric called the effective frame rate. Section 4 provides the experimental results of our research, including a simulation-based study of the algorithms and an exploration of the effective frame rate measure. Finally, we summarize our findings in Section 5.

2. Background and related work

There a large number of related research ideas that dove-tail to the techniques proposed here. In this section, we briefly highlight some of this work.

2.1. Content distribution networks and proxy-based delivery mechanisms

There are a number of different approaches to the wide-scale distribution of video information. They are distinguished primarily by the way that the data is managed between the server that created the video data and the clients.

Content Distribution Networks (CDNs) focus on distributing the video information, in whole, to caches and proxies that are close to the end clients that require the information. For companies that provide access to web data, such as Inktomi or Akamai, this involves creating large distributed data warehouses throughout the world and using high-capacity, peer-to-peer networks to make sure the distribution infrastructure is up to date. The video data that is delivered to the user typically originates from the cache itself.

Proxy-based dissemination architectures focus on actively managing streaming video content through the proxy. Such techniques include efforts from the University of Southern California (RAP) [8], the University of Massachusetts (Proxy Prefix Caching) [9], and the University of Minnesota (Video Staging) [10]. Unlike the CDNs, these proxy-based mechanisms partially cache video data as it is streamed from the server to the proxy and through to the client. Additional requests to retrieve a video stream from the proxy will result in missing data being filled in at the proxy.

For our work, we assume that the proxies are far enough from the client to achieve a reasonable hit rate to the video data. As an example, the proxy could serve an entire campus or company. The streaming mechanism that we have developed can be used to deliver data from the data warehouses in the CDNs to the clients or can be augmented to include the techniques that have been developed for proxy-based video distribution networks.

2.2. Video streaming protocols

There are a large number of streaming protocols that have been developed for point-to-point video distribution. For sake of brevity, we will purposely omit discussion of techniques developed for the distribution of video over multicast networks such as [17]. For point-to-point video streaming, there are a tremendous number of streaming algorithms including the Windows Media Player, RealPlayer, the Continuous Media Toolkit (CMT) [11], Vosaic [2], the OGI Media Player [12], and the priority-based streaming algorithm [13].

These algorithms are for the most part network congestion sensitive, with some being arguably more TCP-friendly than others. In addition, most of these streaming algorithms focus on local adaptation of video sequences to network bandwidth availability and have either no or limited retransmissions of missing data. The lone exception to these is the priority-based streaming algorithm, which focuses on adaptation at the minute level (but still does

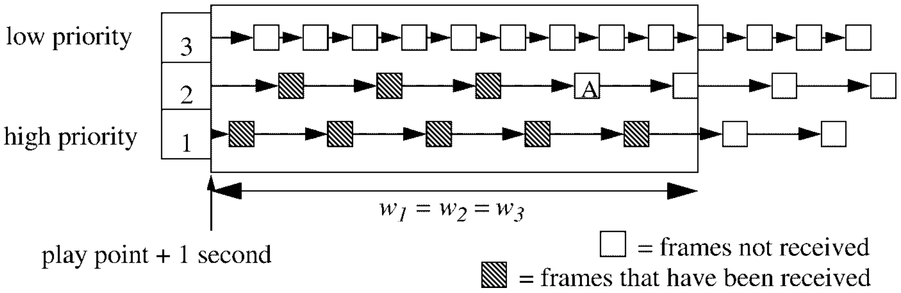


Fig. 1. Priority-based streaming algorithm example: the window keeps track of which frames are buffered in the client. Using this window, the server always attempts to transmit the highest priority frame within the window that has not yet been sent. In this example, the first unsent frame in priority level 2 is transmitted next (labelled A).

not completely use all the *a priori* information in its streaming algorithm). The priority-based streaming algorithm (PBSA) algorithms prioritizes all the frames within a given video. This assignment can be assigned to gracefully degrade the frame rate as bandwidth becomes scarce. It does so by assigning frames and layers of the video to different priorities within a sliding window (just ahead of the playpoint). This window is used to ‘look ahead’ into the movie at least a minute into the future. Using this window, it then delivers all frames in the window at the highest priority before delivering the lower priority frames within the window. As a result it is able to gracefully degrade the quality of video during times of congestion. An example of the PBSA mechanism is shown in Fig. 1.

3. Proposed adaptive streaming mechanism for the delivery of stored content

In this section, we propose a mechanism for the delivery of stored video content across best-effort networks. Before describing the algorithm, we first describe some basics of our approach including (i) some of the underlying assumptions that we make, (ii) a discussion of the trade-off between quality, frame rate, and buffering, and (iii) a metric for the evaluation of streaming media.

3.1. Basics

3.1.1. Underlying assumptions

There are a number of aspects of our approach that we believe are important for the delivery of stored video content over best-effort networks. These characteristics are described below:

- Moving towards more bursty video sources – As the bandwidth over the network increases, the ability to stream higher-bit-rate MPEG streams becomes possible. This increase in bandwidth will create streams of greater bandwidth diversity than we currently see on the Internet.
- Using *a priori* information – We believe that a large portion of the video streamed over the Internet is (and will continue to be) stored video. That is, video that is accessed from a disk subsystem at the server. Because the network bandwidth available over the Internet is time-varying and highly varied, we believe that the knowledge of the content of the stream must be used to aid in its delivery.
- Buffering at the Receiver – We believe that buffering limitations in the client end systems are becoming a non-issue. We believe this because disk and memory capacity is outpacing the growth in bandwidth available to a single flow (not in aggregate). Thus, we can focus on getting the video data into the buffer without worry of buffer overflow.
- TCP transport layer – We believe there are several reasons why stored video streaming should happen over the TCP protocol. TCP is a widely deployed transport protocol. This allows applications that are built on this architecture to be deployed rapidly over a large scale. Second, it allows us to resolve the TCP-friendly issue

rather easily: it is TCP-friendly. Third, we do want flow/congestion control as well as retransmissions. The reason for flow/congestion control is fairly obvious. The reason we want retransmissions for stored video is that we are carefully planning the delivery of the stored sources well in advance (even for B-frames), thus, when we transmit a frame we want to receive it in whole at the receiver. Finally, we note that our algorithms are typically buffering tens of seconds to minutes ahead and thus can deal with the sawtooth nature of TCP and the delay of retransmissions. While the choice of TCP is not necessary to run the proposed algorithm, we believe that the quality delivered over time will be more stable.

3.1.2. Quality and frame rate adaptation

For the delivery of stored video content, there are number of options available for adaptation over best-effort networks. We can control the quality of the video stream, the frame rate of the video stream, and the ordering in which the video data is streamed across the network. In fact, each of these can be adapted in isolation or combined together in the delivery of video content. It is our belief that while any quality video can be used at any frame rate, we believe that there are operating points that will be used to correlate frame rate and frame quality.

Operating points define interesting sets of combinations of frame rate and quality. The streaming algorithms we develop are constrained to move from pre-determined operating points to other pre-determined operating points. Switching between operating points (as the bandwidth availability change over the network) is guided by the principle of correlated priority between frame-rate and quality. In other words, we choose not to compromise one parameter significantly with respect to the other. Therefore any degradation/improvement in the streaming status would trigger comparable degradation/improvement in both the parameters. This is a fairly common approach as found in the Quasar streaming model as well as industrial models such as the Real Networks SureStream mechanism.

The OGI Priority Progress Streaming algorithm (that is part of the Quasar streaming model) has a number of features that dovetail with the work proposed here. First, there is the scalable video format called SPEG [14]. This format is similar to the fine grain scalability option in MPEG-4 which allows ‘layers’ to be shed in case the bandwidth available for streaming is less than that needed by the stream. Second, the quality of video mapper assigns priorities to both frames as well as quality levels. For our purposes here, we use the SPEG streaming format to select an appropriate frame rate and frame quality mixture to maximize the video quality delivered (based upon the users preference).

3.1.3. Effective frame rate

Our proposed video streaming algorithm attempts to maximize the frame rate displayed to the client, while minimizing the variability or the jerkiness in the display frame rate. While we believe that it is ultimately up to the end user to decide how aggressive or conservative the streaming algorithm is (while still being TCP compliant), we believe it is important to define a metric that adequately measures this objective function in order to evaluate the performance of the algorithm.

In the past, researchers have often measured the average and variation in the frame rate delivered to the client as metrics for performance. While there is typically a strong correlation between standard deviation and jerkiness, this may not necessarily hold true. For example, consider a stream displayed at 10 frames per second for half the duration of the video, and at 20 frames per second for the remaining half. Additionally, consider a stream displayed alternately at 10 and 20 frames per second throughout the length of the video. The average frame rate and standard deviation in both cases is the same, but the perceived jerkiness in the former is much less than that in the latter.

As another example, we have graphed the frame rate delivered for a video sequence using the priority based streaming algorithm from reference [13] in Fig. 2a. In Fig. 2b we have merely sorted the frame rate points making up figure (a). Again, it is clearly evident that the user would perceive much more jerkiness with the frame rate delivered in figure (a) than with (b). However, the average frame rate and standard deviation in both cases is the same.

Unfortunately, the user experience will depend upon the user’s preference on how quickly rate adaptations should occur, making it nearly impossible to create a single metric that encompasses these preferences. We propose a

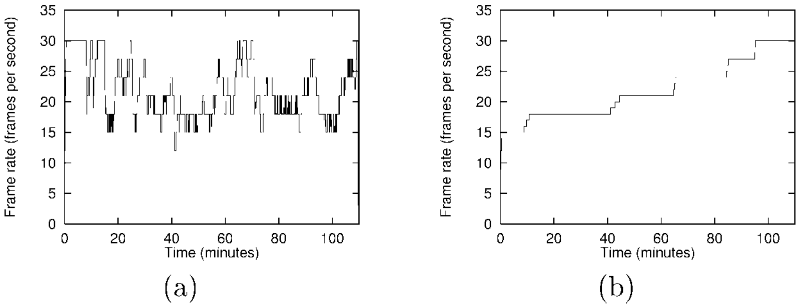


Fig. 2. Frame rate measurement example: (a) shows an example of the frame rate delivered by the priority-based streaming algorithm. (b) shows the same frame rate points but sorted in ascending order. Both figures exhibit the same frame rate and frame rate variation.

metric that accounts for the jerkiness in the display frame rate more appropriately. We define *Effective Frame Rate* (*EFR*) as follows:

$$\mathit{EFR} = \sum_{i=1}^{N-1} \left(\frac{fps_i - P * \mathit{qualitychange}(\max(i - W, 0), i)}{N} \right),$$

where fps_i is the number of frames delivered in the ith second of the video clip, W is the size of the window in which the smoothness metric should be applied, P is a weighting factor for variations in frame rate, and $\mathit{qualitychange}(\max(i - W, 0), i)$ is the number of quality changes in the range $i - W$ to i. The idea behind EFR is that it first measures the average frame rate delivered to the user and then penalizes the EFR for variations in the frame rate and quality delivered.

The penalty in the calculation of the frame rate is determined by two parameters in the metric, W and P. These two terms determine the nature of the penalty to be applied when the frame rate changes. W is the size of the window in seconds in which the smoothness criteria is to be applied. Larger values of W mean that the algorithm will need to deliver a constant operating point over a longer period of time to achieve a higher EFR. The term P is a weighting on how important the changes are to the overall frame rate. For example, if one does not care about the frequency of quality changes, then P can be set to 0. In this case, the algorithm that will do the best simply delivers the most frames regardless of the number of changes. These two terms can be set to user preferences to allow one to evaluate the algorithms based upon their own idea of what makes a good streaming algorithm.

One caveat of the effective frame rate measure is that it does not measure frame quality in any way. Because we are using a set of operating points where the frame quality and frame rate are somewhat tied together, the frame quality is implied by the effective frame rate. One could augment this metric with the peak signal-to-noise ratio (PSNR) for the frames delivered if quality is important. Note that the PSNR measure does not capture frame rate when used alone, thus, one could maximize the PSNR measure by sacrificing frame rate completely.

3.2. Content-based video adaptation

In this section, we describe our proposed content-based video adaptation (CBVA) protocol. We break the discussion into two main parts: a high-level overview that describes the basics of the adaptation algorithm and a lower-level implementation subsection that describes some of the implementation details to make the algorithm work.

3.2.1. Basics of CBVA

At a high level, there are two things that are dynamically changing over time, the bandwidth requirements of the video stream and the data available over the network. The former can be somewhat controlled through careful adaptation policies while the latter is out of our control. Thus, our approach focuses on the actual data that needs to be delivered and then to deal with the variability of the network bandwidth as it occurs. As a simple example of this approach, consider the frame sequence shown in Fig. 3. In analyzing the video data, the bandwidth requirement for seconds 3 and 4 are not nearly as large as those in seconds 5 and 6. Thus, even if sufficient bandwidth is available during seconds 3 and 4 to stream at the highest available quality, the streaming algorithm should use some of the transmission bandwidth for prefetching later data by streaming at a slightly lower quality during seconds 3 and 4.

The overarching goal of the CBA algorithm is to determine a reasonable plan for the delivery of the stored video content. Here, we revisit the notion of critical bandwidth smoothing that was introduced in 1995 [15,16]. The idea here is that every movie exhibits a different critical bandwidth requirement – the bandwidth required to play back the movie without buffer underflow at the full frame rate from a given starting point. This is calculated by finding:

$$\mathit{critBw}(x) = \max_{x \leqslant i \leqslant N} \left(\left(\sum_{j=x}^{i} \left(\frac{\mathit{framesize}_j}{i} \right) \right) - \mathit{prefetchbytes} \right),$$

where x is the starting point and prefetchbytes is the amount of data that we have prefetched already from the point x to the end of the movie. For the beginning of the movie x is 0. For any point else in the movie x is the frame number that has not yet been transmitted to the client. The critical bandwidth for the example in Fig. 3 is 3.29 units (as labelled in the *running average* and occurs in second 6 of the movie. We refer to the point that causes the critical bandwidth as the critical point. In this case, the critical point is second 6 of the sequence shown. The buffer underflow condition can be written as follows:

$$f_{\mathit{underflow}}(j) = \sum_{j=0}^{i} \mathit{framesize}_j.$$

Two additional examples are shown in Fig. 4. The heavy solid lines show the critical bandwidth, while the lighter lines show the buffer underflow curve. This example shows one with no prefetch (a) and the other with prefetch (b). Note, any prefetch (and hence, delay), will help reduce the rate required for streaming it across the network. There

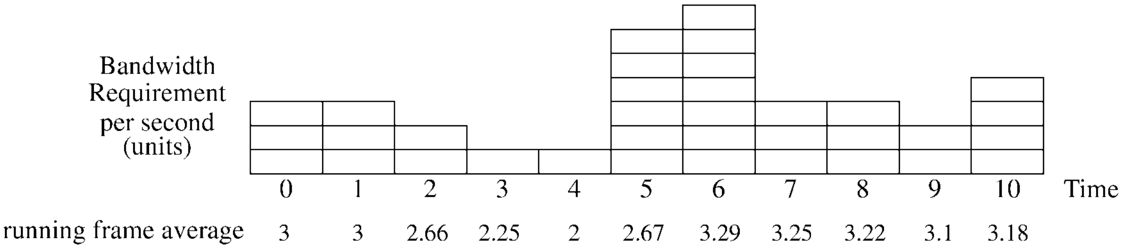


Fig. 3. Critical bandwidth example.

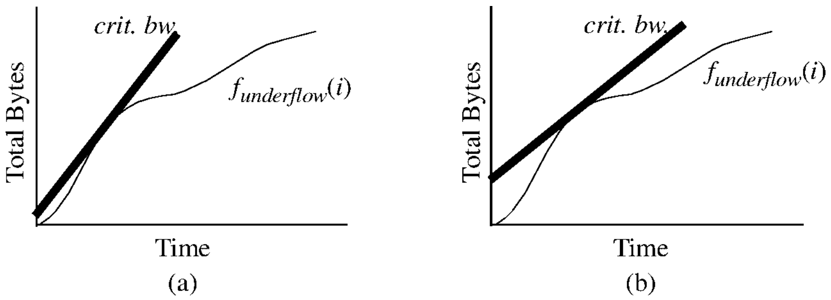


Fig. 4. Operating point example.

is one property of this algorithm that is important to our work here: Once past the point i, which we call the critical point, the bandwidth requirement for the rest of the movie must be monotonically decreasing. That is, if one finds the critical bandwidth from point i to the end of the movie, then it must be less than that from frame 0 to i (see reference [16] for details). For our work here, frame i really describes the region that is the most challenging to deliver across the best-effort network. Once past that point, the bandwidth requirement for the stream will decrease.

In the delivery of video, each operating point will exhibit its own critical bandwidth. To start streaming, we select a conservative (lower-priority) operating point as this allows us to measure the bandwidth available over several seconds and allows us to get a little ahead of the playback point. Using the critical bandwidths of the operating points, we chose the operating point that is closest to the available bandwidth. It is important to remember that the critical bandwidth represents the constant bandwidth required to deliver the video over a larger period of time. If the streaming algorithm gets significantly ahead or behind of schedule or the critical point for the current operating point is passed, the algorithm re-evaluates the critical bandwidth and critical points for the operating points. This process continues until the end of the movie.

3.2.2. Algorithmic details

There are a number of questions about how one actually implements such an algorithm and makes it tractable enough for deployment. In the remainder of this section, we will describe the various parts of the adaptation algorithm that we have implemented.

Setting up the video data for CBVA. To create the operating points and the data for the CBVA algorithm, we assume that either the video is encoded into multiple quality/frame-rate video streams or that the video can be adapted into the multiple quality/frame-rate pairs on-the-fly as needed. The trade-off here is that having more operating points allows for finer granularity adaptation at the expense of higher overhead. Once we have the operating points, we then group the video data into groups of frames. For MPEG-like video streams, we treat each operating point's group of pictures as a single unit (typically this works out to half second group of pictures). The purpose of this is to minimize the overall overhead in running the CBVA algorithm. Thus, for a 90-minute movie we end up with 10 800 points for each operating point that will be used by the CBVA algorithm to determine its adaptation.

Analyzing the data and preparing for transmission. For each operating point, we then calculate its critical bandwidth, its critical bandwidth, and a set of deadlines that provide guidance on the progress of the streaming. We have shown an example with two operating points for a stream in Fig. 5. In this example, the critical points for both operating points is GOP 8 in the movie and the critical bandwidths for the entire stream are 3.6 and 2.33 for the high and low quality streams, respectively. The important calculation that we use to determine how well we are doing is the deadline calculation. If we are streaming at a particular operating point, the deadline tells us when we should expect to have the data transmitted by. For example, let us assume we are transmitting at the low quality stream. After transmitting for the first 2 GOP periods, we expect to have transmitted the first 4 GOPs to the client (deadlines 0 and 1). In essense, we are creating deadlines for the data that are proportional to the amount of data within each GOP.

To calculate the deadline (in GOPs) for a given operating point with no prefetching we use:

$$\mathit{deadline}(i) = \left\lceil \max_{0\leqslant j\leqslant i}\left((\mathit{critBw} * j) \geqslant \sum_{k=0}^{i} \mathit{framesize}_k\right)\right\rceil.$$

If we are transmitting data well in advance of the 'deadline' target, then we may consider increasing the quality because we are relatively ahead in transmission *of the smoothed stream*. Conversely, if we start to fall a little behind in the deadlines (i.e., we are transmitting data after the deadline target) this indicates that we are falling behind and may consider decreasing the quality to stay ahead.

Estimating the average available bandwidth is somewhat useful (but not necessary) for this algorithm. The bandwidth available is estimated by measuring the bandwidth available in the past. The average is calculated

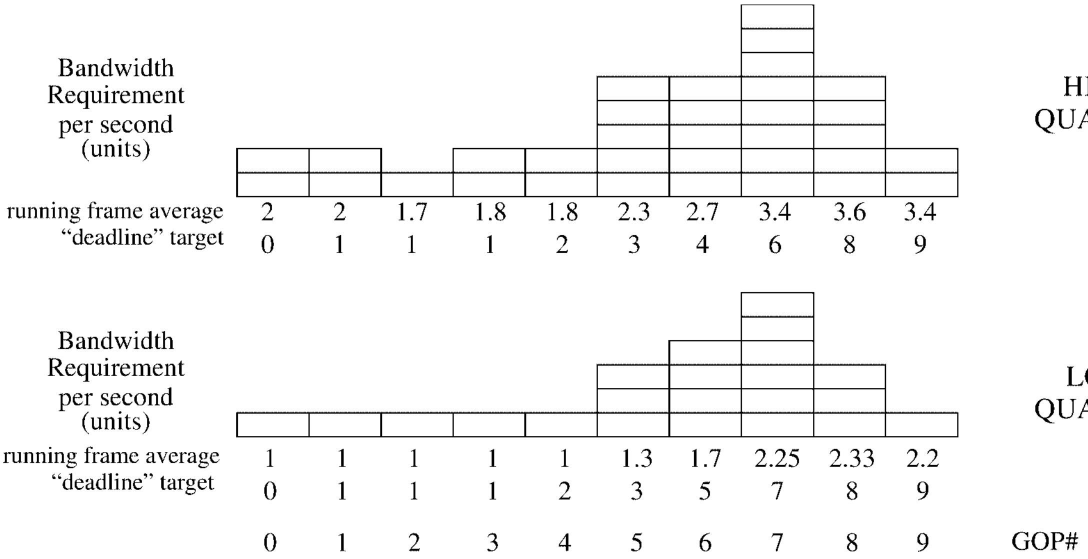


Fig. 5. Operating point example.

over an interval and depends on the environment to some degree. If the bandwidth available is fairly stable then the average can be calculated over small intervals, on the other hand if the available is very erratic, then the average bandwidth should be calculated over larger intervals in order to reduce any error in frame rate estimation which is calculated using this bandwidth. We found that an interval of 2 to 3 minutes for a typical environment works reasonably well as it helps to smooth out the short-term variation from the bandwidth. Finally, we note that the bandwidth estimation algorithm does not have to be as accurate as in algorithms in the past since we are typically working tens of seconds to minutes ahead of the playback point so small errors in bandwidth estimation are typically taken care of by the time the playback point is at the point where the error occurred.

To begin the streaming of data, we initially choose a conservative operating point. During this period an estimation of bandwidth is obtained. We use this estimate of bandwidth to determine the appropriate adjustment in operating point, if necessary.

Adjusting the quality during transmission. Once streaming begins at an operating point, we monitor the difference between the wall clock time and the deadline target time. If we are transmitting ahead of the deadlines and this difference is growing, then we may consider increasing the quality/frame rate of the video. Aggressively increasing the quality of the video may result in larger fluctuations in quality while conservatively increasing the quality may result in a lower quality stream than might have otherwise been possible. There are two limits that we are concerned with. The *decrease limit* determines how close (or behind) the deadline target is allowed to stray before potentially forcing a decrease in the quality level of the stream. The smaller this number is the higher the probability of fluctuations in operating points becomes. The *increase limit* determines how far ahead the transmission time can get in relation to the deadline targets. A larger increase limit smoothes out the stream but may end up delivering a lower quality than might have otherwise been possible. We refer to the difference between the increase and decrease limits we refer to as the *deadline slack*. This slack will ultimately be controlled by the user's preference of how aggressive he/she wants the algorithm to be.

Using the limits, we stream the data and monitor the difference between deadline and transmission times. If either the increase limit or decrease limit is reached, we re-run the CBA algorithms on all the operating points to recalculate the critical bandwidths, the critical points, and the deadlines. Once re-run, we compare the critical bandwidths to the current estimate of bandwidth (as described above) and reselect an appropriate operating point to stream from. We note here that, if one chooses to not calculate bandwidth estimates explicitly, then reaching the decrease and increase limits can force a decrease or increase in the operating point automatically.

The pseudo-code for our algorithm is shown in Fig. 6.

```
critBw[operating point]; // array of critical bandwidths
critPt[operating point]; // array of critical points
                         //  one per operating point
deadGOP [operating point][GOP #] // deadline of GOP

Aggregate frame data into GOPs to reduce computational burden
For all operating points i
Calculate critBw[i], critPt[i], deadGOP[i]

bwTarget = estimate of bandwidth available  //conservative estimate
Choose op such that critBW[op] < bwTarget
Start streaming data

i = first GOP of movie;
While (i < last GOP of movie){
  Transmit GOP deadGOP[op][i]
  deadlineDiff = deadGOP[op][i] - currentGOPTransmissionTime
  if ((deadlineDiff > INCREASELIMIT)||(deadlineDiff < DECREASE_LIMIT))
    Recalculate critBw[i], critPt[i], and deadGOP[i]
    Choose op such that critBW[op] < bwTarget
    op = new operating level
  i++;
}
```

Fig. 6. CBVA pseudo-code.

4. Experimentation

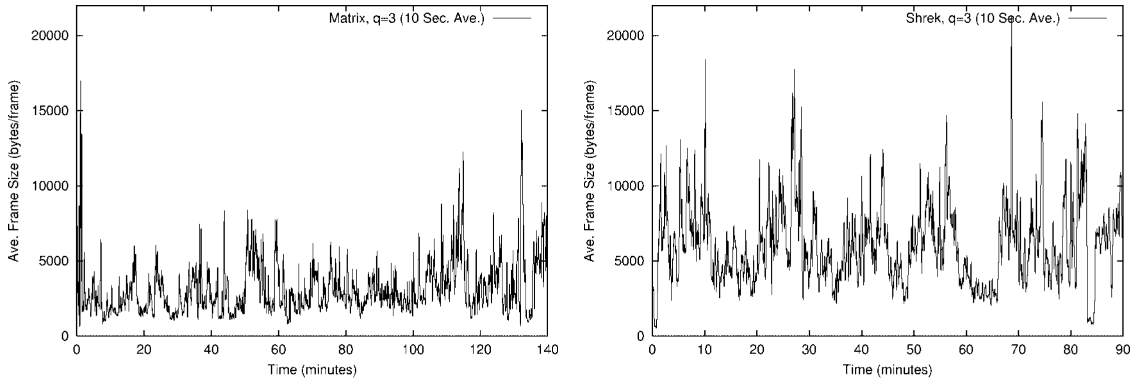
In this section, we describe some of the performance results of our streaming algorithm compared with simple streaming algorithms that perform rate changes based on per-second adaptation and the priority-based streaming algorithm which uses windows that are in the size of minutes. We will also explore the use of the effective frame rate metric for measuring a streaming algorithm's performance. Before we describe our experimental results, however, we will describe our experimental simulation environment and the video data and network traffic trace that we captured.

4.1. Experimental environment

We have converted the movies *Matrix* and *Shrek* from DVD into MPEG-1 constant-quality formats for use in our simulations. Each of the movies was compressed into the following group of pictures pattern:

$$IBBBPBBBPBBB$$

Because the movies were encoded at 24 frames per second, our encoded streams have a rate of two GOPs per second. In addition, we compressed each video using fixed quantization values of 2, 3, 5, and 9. These quantization values led in a somewhat linear degradation in bandwidth between the levels. We have graphed the 10 second frame averages for the quantization $= 3$ movies in Fig. 7. The movie *Matrix* has a particularly large burst of bandwidth required one minute into the movie. As will be shown later, this can cause some problems with limited prefetching time. For *Matrix*, there are only two critical bandwidth points: one approximately 100 seconds into the movie and



(a) (b)

Fig. 7. Movie examples: Figure (a) shows the bandwidth requirements smoothed using a 10 second sliding window for the movie *Matrix* encoded with a fixed quantization level of 3. Figure (b) shows the same for the movie *Shrek*.

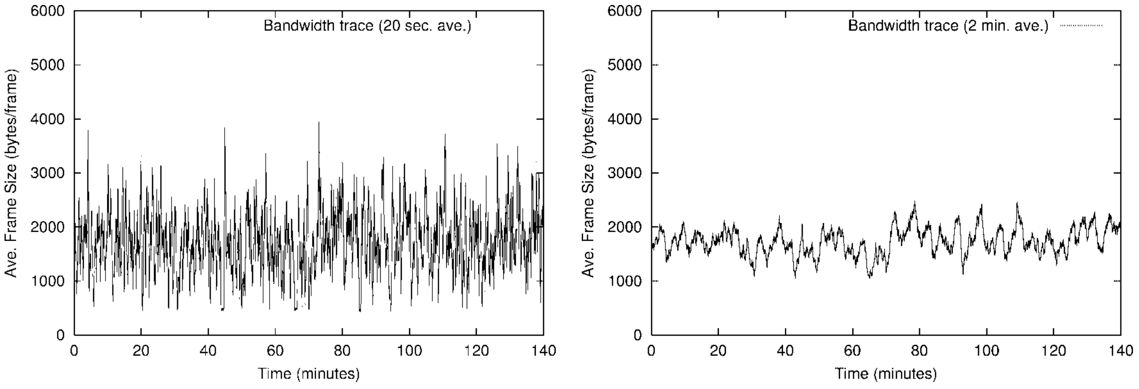
Table 1
Movie statistics: Statistics for the compressed video streams that were used for the simulations

Movie	Ave. Bit Rate (Mbps)	Quant. value	Length (min)	Ave. size (bytes)	Ave. I-frame size (bytes)	Ave. P-frame size (bytes)	Ave. B-frame size (bytes)
Matrix	0.943	2	141	4909	8791	5941	4249
Matrix	0.611	3	141	3182	6889	3872	2617
Matrix	0.370	5	141	1928	4875	2263	1526
Matrix	0.225	9	141	1174	3362	1259	912
Shrek	1.82	2	90	9460	18072	12589	7830
Shrek	1.16	3	90	6065	13715	8222	4734
Shrek	0.678	5	90	3531	9027	4840	2630
Shrek	0.382	9	90	1987	5587	2617	1447

another at the end of the movie. This is underlined by the fact that there are significant bursts of data early in the movie and at the end. For the movie *Shrek*, there are four critical points within the movie: 12.3 minutes, 59.3 minutes, and 84.1 minutes into the movie as well as one at the end of the movie. Finally, the statistics for the video trace data that we used in the experimentation are shown in Table 1.

For the video adaptation experiments, we used these streams to create 12 operating levels for each video. The GOP patterns and quantization values that were used for these levels are shown in Table 2. Note that in Section 4.1.2, we fixed the quantization value for all the various operating points. For *Shrek* we set the quantization to 3, and for *Matrix* we set the quantization level to 5 in order to equalize the average bandwidth requirements somewhat.

To provide deterministic results of the various algorithms, we captured a two and a half hour long TCP trace, which was captured from the University of Michigan to Ohio State. Thus, the comparisons that we draw in this paper are based upon the TCP traces. For actual streaming algorithms that just use UDP with or without flow control and with or without retransmissions may be slightly different than those presented here. We also note that comparing the performance of streaming algorithms that receive partial data (as in a UDP stream) is extremely difficult as it is hard to compute what the relative impact of receiving half a frame of video is for any streaming algorithm. The bandwidth trace (shown at two different smoothing intervals) is shown in Fig. 8. The average bandwidth of this trace is 327 kbps.



(a) (b)

Fig. 8. Bandwidth trace: The bandwidth trace captured from the University of Michigan to Ohio State using TCP. In (a), we have smoothed the bandwidth trace using a sliding window of 20 seconds. In (b), we have smoothed the trace using a 2 minute window to show the longer-term variation in bandwidth from the trace.

Table 2

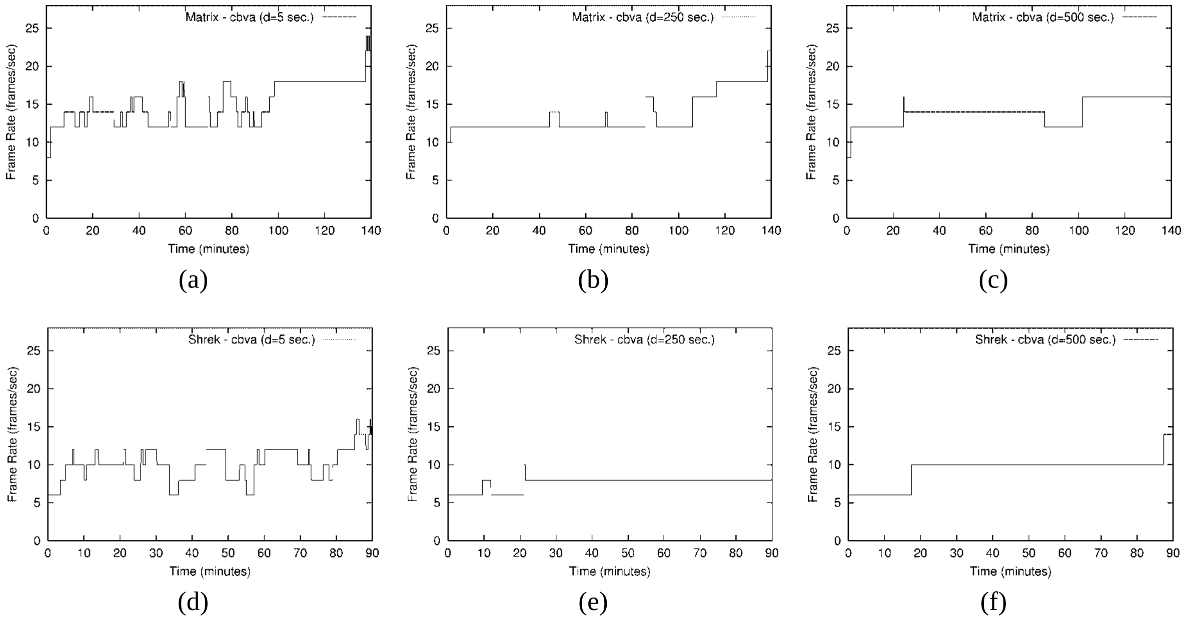
Movie encodings: Various encodings that were used in the experiments

Frame rate	GOP pattern	Quant. value
24	IBBBPBBBPBBB	2
22	IBBBPB BPBBB	2
20	IB BPBBBPB B	2
18	IB BPB BPB B	3
16	IB BP B PB B	3
14	I B PB BP B	3
12	I B P B P B	5
10	I B P P B	5
8	I P B P	5
6	I P P	9
4	I P	9
2	I	9

For comparison purposes, we use two streaming algorithms for our experiments. The first, which we will refer to as the naive algorithm, simply looks to transmit the appropriate quality in order to stay at least two seconds ahead of the playback point. This algorithm is very similar to the algorithms found in the CMT toolkit out of Berkeley [11] or the Oregon Graduate Institute player [12]. The Priority-Based Streaming algorithm is an extension to the basic naive algorithms that uses a much larger lookahead window. Here, the lookahead was set to two-minutes, meaning that all high-priority (I-frames) must be received before the algorithm starts delivering lower priority-frames (P and B). Thus, we expect the priority-based algorithm to provide substantially smoother operation than the naive algorithm. These two algorithms are explained in more detail in reference [13]. Finally, we note that each algorithm was allowed to prefetch 5 seconds of data and was not allowed to use the future bandwidth availability from the network trace.

4.1.1. Effect of deadline slack

For the adaptation of video to changing network resources, the CBVA algorithm's primary mechanism for how aggressive or conservative it is is the amount of *deadline slack* that it has. For a very small deadline slack, the algorithm will react more quickly to increases in available network resources. At the same time, however, it can



(a) (b) (c)

(d) (e) (f)

Fig. 9. Deadline slack: The effect of varying the deadline slack in the CBVA algorithm. Figures (a)–(c) show the effect of increasing the deadline slack for the *Matrix* movie and (d)–(f) for the movie *Shrek*.

end up increasing its quality only to have it decrease it again. The important point, however, is that increasing the quality can only happen if the measured bandwidth will support the new quality until the critical point. In Fig. 9, we have graphed the CBVA delivery the *Matrix* and *Shrek* for the bandwidth trace in Fig. 8 for three different bandwidth slack values.

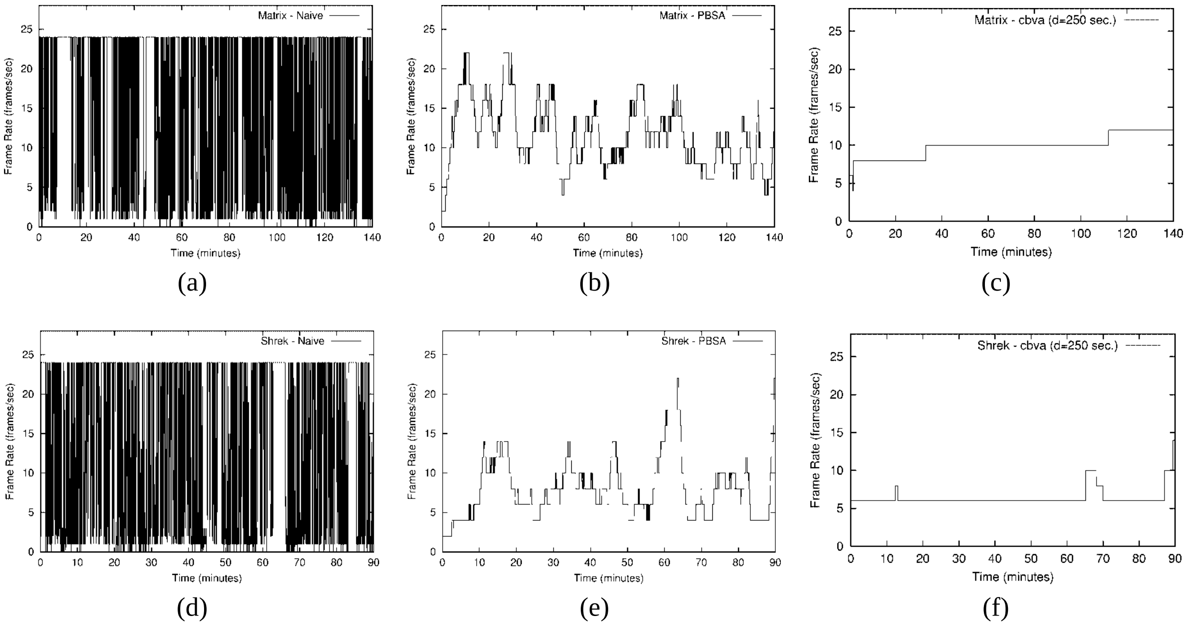
For a deadline slack of 5 seconds, where the transmission of video data at the chosen quality has gotten only 5 seconds ahead of its deadline, the algorithm re-evaluates the critical bandwidths of the various operating points and adjusts its operating point, if possible. While still reasonably smooth, the algorithm varies its operating unnecessarily several times (e.g., minute 20 of the movie *Matrix*). As the deadline slack is increased, the delivery of the streams becomes significantly smoother (and at the same time more conservative it is use of bandwidth). Finally, by definition of the critical bandwidth algorithm, our expectation is that once crossing the critical point, the critical bandwidth requirement must be smaller than before crossing the critical point. As a direct result of this, it is always easier to deliver video data towards the end of the movie than it is at the beginning.

There are two mechanisms to avoid significant increases in the quality of the video throughout playback. First, one can employ a greater amount of prefetching at the beginning of the movie. Because we limited the prefetching in the algorithms shown to 5 seconds of prefetching, the initial operating point is necessarily low, particularly for the *Matrix* video. With an increase amount of prefetching time the initial lower quality video stream can be avoided. Alternatively, one can limit the quality level chosen at the end of the movie (i.e., stop transmitting the video early). The choice of both of these mechanisms is ultimately up to the user.

4.1.2. *Effect of frame-based encoding*

To compare and contrast the various streaming algorithms, we first begin with adaptation in frame-rate only while keeping the quality of encoding the same. By dropping frames only, the bit-rates of all the operating points are closer together. For the movie *Matrix* using a fixed quantization value of 5, the operating points cover 100 to 612 kbps. For the movie *Shrek* using a quantization value of 3, the operating points cover 144 kbps to 678 kbps.

In Fig. 10a and d, we see that the simple local adaptation policies have a hard time stabilizing the frame rate because both the network bandwidth and video bandwidth continue to vary over time. We see that in several places (e.g., minute 62-66 of *Shrek*), that the naive algorithm maximizes its frame rate during periods where the bandwidth



(a) (b) (c)

(d) (e) (f)

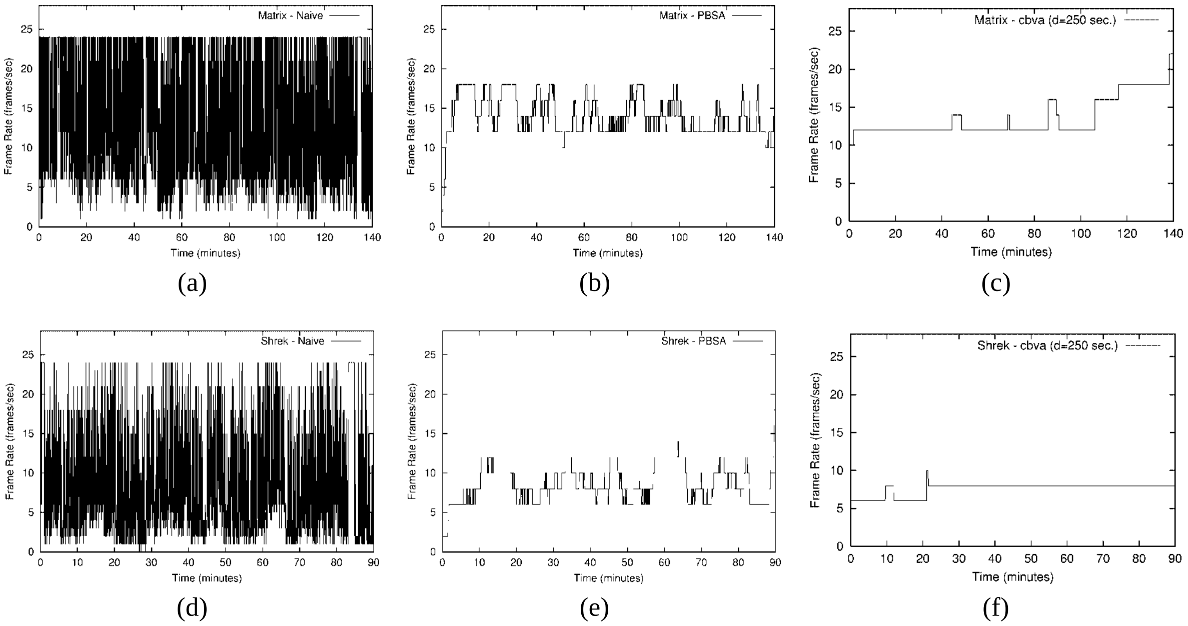
Fig. 10. Frame-based adaptation: The result of running the three streaming algorithms for the movies *Matrix* and *Shrek*. (a) and (d) show the simple adaptation algorithms. (b) and (e) show the priority-based streaming algorithm using a 2.5 minute adaptation window. (c) and (f) show the CBVA streaming algorithm with a 250 second slack window.

requirement of the stream is small relative to the network bandwidth. In Fig. 10b and e, we see that the priority-based mechanism using a smoothing window of 2.5 minutes smoothes out the rate variations overtime by slowly building up and decreasing the quality level. Again, we see that when the network bandwidth increases relative to video data (e.g., minute 62-66 of *Shrek*) that it is reflected in the operating point chosen. In Fig. 10c and f, we see that the CBVA algorithm is significantly smoother than the other approaches. In particular, it is not ‘fooled’ as much into increasing the bandwidth during minutes 62-66 of the movie *Shrek* as the other algorithms.

4.1.3. *Effect of quality/frame-rate encoding*

In this subsection, we compare and contrast the various streaming algorithms using both frame-rate and quality adaptation. There are a couple of differences between this type of adaptation versus simple frame adaptation. First, by adapting both the quality and frame rate, the range of bandwidth that the operating points can cover is larger. For the movie *Matrix*, the range of operating points expands from 100–612 kbps to 53–943 kbps. For the movie *Shrek*, the range of operating points expands from 144–678 kbps to 89–1816 kbps. Second, with the bandwidth requirement between operating points having increased, the adaptation tends to be more stable over time for the naive and priority-based streaming algorithms. Finally, for the priority-based streaming algorithm, the overhead in streaming quality and frame-rate adaptive video becomes extremely complicated due to the fact that each change in quality or frame rate requires the algorithm to keep track of a separate priority during streaming. For the naive and CBVA algorithms, the overhead involved with quality and frame-rate adaptive video is essentially the same as the frame-rate only adaptive video. The reason for this is that the naive and CBVA algorithms do not reorder the data (as in the priority-based streaming algorithm), allowing changes in operating points to be simplified.

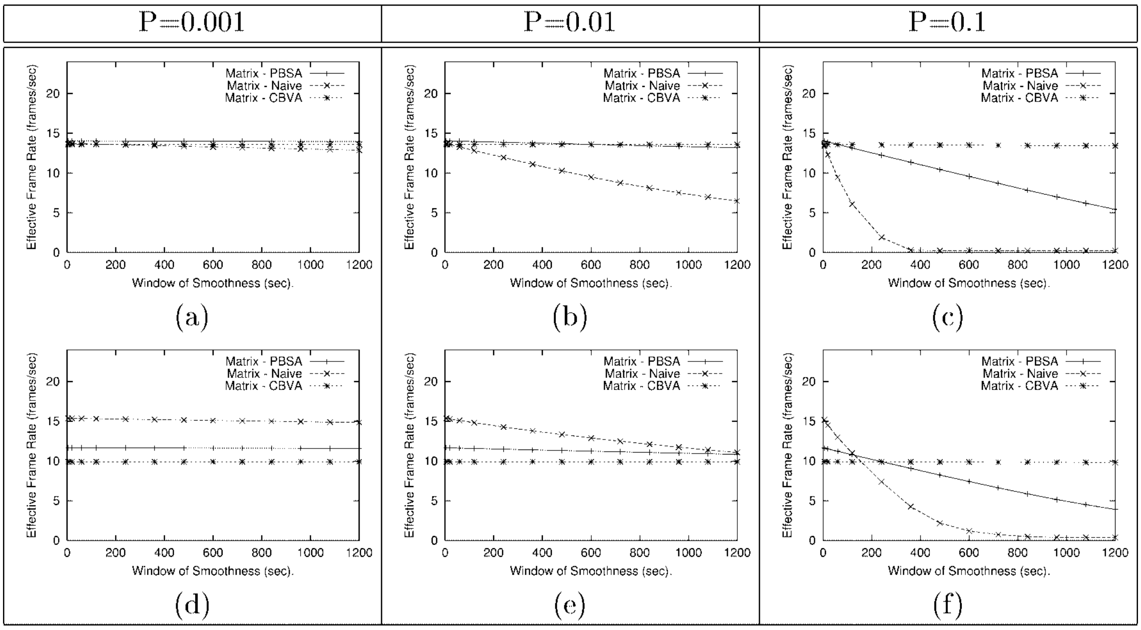
In Fig. 11, we have graphed the frame rate delivered for the various algorithms. Compared with Fig. 10, we see that the priority-based streaming algorithm is more stable over time. One interesting phenomena that we came across was the fact that in total frames delivered the priority-based streaming algorithm was able to deliver more frames when both quality and frame-rate adaptive video was used. The reason for this is that the priority-based algorithm would, at times, deliver a slightly lower operating point *but for a longer period of time*. Because the greedy nature of the naive algorithm, oscillations in network bandwidth causes the algorithm to use all the bandwidth to increase the quality and frame-rate for a shorter period of time.



(a) (b) (c)

(d) (e) (f)

Fig. 11. Quality/frame-rate adaptation: The result of running the three streaming algorithms for the movies *Matrix* and *Shrek*. (a) and (d) show the simple adaptation algorithms. (b) and (e) show the priority-based streaming algorithm using a 2.5 minute adaptation window. (c) and (f) show the CBVA streaming algorithm with a 250 second slack window.



(a) (b) (c)

(d) (e) (f)

Fig. 12. Effective frame rate measure: (a), (b), and (c) show the effective frame rate measure for penalty weightings of 0.001, 0.01, and 0.1, respectively for the quality and frame-rate adaptive video streaming of Matrix shown in Fig. 11a–c. (d), (e), and (f) show the effective frame rate measure for the frame-rate only adaptive algorithms shown in Fig. 12d–f.

4.1.4. Effective frame rate measure

Figure 12 shows the effective frame rate metric for a variety of window sizes for the movie *Matrix*. Figures (a)–(c) are for the frame-rate and frame-quality adaptive video algorithms and figures (d)–(f) are for the frame-rate only adaptive video streaming algorithms. The results for the movie *Shrek* are very similar and have been omitted from this paper. As shown by figures (a) and (c) using a very small weighting places a heavy weighting on maximizing the overall frame rate. As the weighting of smoothness increases (i.e., P increases), a higher and higher weight is placed on overall smoothness of video delivery. As shown in figures (c) and (f), the CBVA algorithm tends to stay constant, meaning that it is smoother over much larger intervals. The naive algorithm exhibits significant burstiness in quality over time resulting in a dramatic decrease in its effective frame rate. Because the priority-based streaming algorithm is fairly stable over larger periods, the effective frame rate metric tends to be better than the naive algorithm but not as good as the CBVA algorithm.

5. Conclusion

In this paper, we have introduced a content-based streaming algorithm that takes advantage of *a priori* information in the streaming of stored video data over best-effort networks. As we have shown, this approach is much more effective smoothing out the variability in frame rate delivered to the user, even across best-effort networks. We have also introduced an effective frame rate metric to evaluate the effectiveness of various stored video streaming algorithms. The key concept here is that instead of measuring just average frame rate and the variability in frame rate delivered, it calculates an average frame rate and then penalizes the average frame rate depending on how quickly the frame rate changes at smaller time scales.

Acknowledgements

This research was funded in parts by National Science Foundation grants NSF ANIR 98-75493 and NSF EIA 01-30344. The results presented in this paper do not necessarily reflect the views of the funding agency.

References

[1] J. Mahdavi and S. Floyd, TCP-friendly unicast rate-based flow control, Note sent to end2end-interest mailing list, January 1997.

[2] Z. Chen, S. Tan, R. Campbell and Y. Li, Real time video and audio in the World Wide Web, in: *Fourth International World Wide Web Conference*, 1995.

[3] S. McCanne and V. Jacobson, VIC: A flexible framework for packet video, in: *Proceedings of ACM Multimedia 1995*, 1995.

[4] S. Cen, J. Walpole and C. Pu, Flow and congestion control for internet media streaming applications, in: *Proceedings of SPIE Multimedia Computing and Networking 1998*, 1998, pp. 250–264.

[5] I. Rhee, Error control techniques for interactive low-bit rate video transmission over the internet, in: *Proceedings of SIGCOMM 1998*, 1998.

[6] P. Nee, K. Jeffay and G. Danneels, The performance of two-dimensional media scaling for internet videoconferencing, in: *International Workshop on Network and Operating System Support for Digital Audio and Video*, 1997.

[7] M. Talley and K. Jeffay, Two-dimensional scaling techniques for adaptive, rate-based transmission control of live audio and video streams, in: *Proceedings of the Second ACM International Conference on Multimedia*, 1994.

[8] E. Ekici, R. Rajaie, M. Handley and D. Estrin, Rap: An end-to-end rate-based congestion control mechanism for realtime streams in the internet, in: *Proceedings of INFOCOM 99*, 1999.

[9] S. Sen, J. Rexford and D. Towsley, Proxy prefix caching for multimedia streams, in: *Proceedings of INFOCOM 99*, 1999.

[10] Z.-L. Zhang, Y. Wang, D.H.C. Du and D. Shu, Video staging: a proxy-server-based approach to end-to-end video delivery over wide-area networks, *IEEE/ACM Transactions on Networking* (August) (2000).

[11] L. Rowe, K. Patel, B. Smith and K. Liu, MPEG video in software: Representation, transmission and playback, in: *Proc. of IS&T/SPIE 1994 Symp. on Elec. Imaging: Science and Technology*, 1994.

[12] J. Walpole, R. Koster, S. Cen et al., A player for adaptive mpeg video streaming over the internet, in: *Proceedings of 26th Applied Imagery Pattern Recognition Workshop*, 1997.

[13] W. Feng, M. Liu, B. Krishnaswami and A. Prabhudev, A priority-based technique for the best-effort delivery of stored video, in: *Proceedings of the SPIE Multimedia Computing and Networking*, 1999.

[14] B. Krasic and J. Walpole, Qos scalability for streamed media delivery, OGI CSE Technical Report CSE-99-011, September 1999.

[15] W. Feng and S. Sechrest, Smoothing and buffering for delivery of prerecorded compressed video, in: *Proceedings of IS&T/SPIE Multimedia Computing and Networking*, 1995, pp. 234–242.

[16] W. Feng and S. Sechrest, Critical bandwidth allocation for the delivery of compressed prerecorded video, *Computer Communications* **18** (October) (1995) pp. 709–717.