VIDEO-ON-DEMAND SERVICES: EFFICIENT TRANSPORTATION AND DECOMPRESSION OF VARIABLE BIT RATE VIDEO

by

Wu-chi Feng

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
(Computer Science and Engineering)
in The University of Michigan
April 1996

VIDEO-ON-DEMAND SERVICES: EFFICIENT TRANSPORTATION AND DECOMPRESSION OF VARIABLE BIT RATE VIDEO

by

Wu-chi Feng

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
(Computer Science and Engineering)
in The University of Michigan
April 1996

To my parents and Carol

ACKNOWLEDGEMENTS

This dissertation represents the final chapter in my official educational journey. Along the way, I have come in contact with countless number of people that have helped shape my education. While it is impossible to individually acknowledge everyone that has directly or indirectly contributed to my travel, I would like to take this opportunity to thank the people who have influenced me the most.

I would like to thank my advisor Professor Stuart Sechrest for his guidance throughout the course of this work. He has always provided me with another view point from which to approach problems. He has also patiently read and corrected my papers, providing the proper frameworks for my research. Similarly, I would like to thank Professor Farnam Jahanian for his guidance during the latter part of my dissertation and for serving as a Co-Chair on this committee. I would also like to express my appreciation to Professor John Coffey, Professor Atul Prakash, and Professor Kang Shin for serving on my dissertation committee.

During my stay at the University of Michigan, I have had contact with many students whom I have bounced ideas off of and received insightful comments from or have made my dissertation experience go a little faster. Among them are Shawn Blanton, Scott Dawson, Wu-chang Feng, Nigel Hinds, Yen-min Huang, Rob Malan, Ashish Mehra, Todd Mitton, Yoonho Park, Jennifer Rexford, and Anees Shaikh.

Finally, I would like to thank my wife Carol for her constant support, patience, and proof-reading throughout my dissertation. I would also like to thank my parents for their support throughout my educational career.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF APPENDICES

CHAPTER 1

INTRODUCTION

"All of the books in the world contain no more information than is broadcast as video in a single large American city in a single year. Not all bits have equal value." - Carl Sagan

1.1 Background

Multimedia has a virtually unlimited number of applications that can significantly affect the lives of people. Applications such as world-wide-web (WWW) browsers (along with the web-servers they access) allow people to share information in a multimedia format that includes text, audio, still images, and to an extent, video. Live-video conferencing and computer supported collaborative work (CSCW) applications allow physically separated people to interactively discuss and share ideas through the exchange of multimedia information. Applications such as digital libraries and video-on-demand services promise the retrieval of video for virtually any topic the user desires. A common thread throughout all these applications is the need for high quality video within limited resource budgets.

The handling of digital video poses a formidable task to virtually any computer or network system. As an example, a one second 640x480 pixel video clip in 24-bit mode requires approximately 27.6 Megabytes (MBytes) of data to be handled in one second. Furthermore, the storage for this uncompressed second of data requires approximately 150 MBytes. Video compression techniques such as the Motion Pictures Expert Group's (MPEG) compression standard or Motion-JPEG compression greatly reduce the required storage capacity for these videos. These compression standards, however, result in variable bit rate video and requires additional processing to display. Thus, these video compression techniques result in data that may be difficult to manage. The difficulty arises from reasons both e*xternal* and *internal* to end host computers. Externally, compressed video data is hard to manage because of the burstiness that is introduced. From a network point of view, it is difficult to make guarantees of network resources because at different times, the compressed video data requires a potentially wide range of different resource requirements. Internally, compressed video is difficult to handle because of the processing requirements for decompression and reconstruction.

1.2 Research Objectives

In this dissertation, we take an in-depth look at how burstiness introduced by compression standards such as MPEG and Motion-JPEG affects the handling of video data. We concentrate on the handling of bursty prerecorded video data that may be found in an interactive educational system or an interactive video-on-demand video delivery system. Our three-fold approach looks at smoothing the requirements for network data, delivering the video data across networks, and finally how end processors can deal with the burstiness.

The use of these video compression techniques can be applied to both live and stored video applications; however, the handling of the data for these two types of applications have different requirements. Live video applications are typically constrained by the need for network bandwidth scheduling decisions to be made on-line and the delay between sender and receiver minimized. As a result, handling compressed live-video typically requires that statistical guarantees be made for both network and system resources or adjusting the picture quality to fit within a fixed size channel. On the other hand, stored video applications have different needs for network and system resources. Because the video is stored, the delay between sender and receiver does not necessarily need to be minimized as long as the data is received by the playback instance. Thus, buffering can be an effective tool for handling of compressed prerecorded video data.

For the delivery of prerecorded video data, we introduce the notion of *critical bandwidth allocation*. This bandwidth smoothing technique creates a bandwidth allocation plan for the delivery of the video data given *a priori* knowledge of the video data. The critical bandwidth allocation technique allows for the retrieval of stored video that does not require any initial prefetching (and hence, delay) and results in a monotonically decreasing sequence of bandwidth allocations. Given some fixed buffer size constraint, the critical bandwidth allocation algorithm creates plans for the continuous playback of stored video that have (1) the minimal number of bandwidth increases, (2) the smallest peak bandwidth requirement, and (3) the largest minimum bandwidth requirement. We extend this idea into an *optimal bandwidth allocation policy* that, in addition to the critical bandwidth algorithm properties, also minimizes the total number of bandwidth changes required for the continuous playback of stored video.

While the use of bandwidth smoothing techniques are effective at removing the peak bandwidth requirements of networks for a single video stream, smoothing of bandwidth through the prefetching of data makes the bandwidth plans somewhat rigid. This, in turn, makes it difficult to support VCR capabilities such as stop, pause, rewind, and fast-forward. To support these interactive functions, we introduce the notion of the *VCR-window*, which allows users to have full-function VCR capabilities within a limited segment of video around the playback point without having to change the level of the bandwidth reservations. For accesses outside the VCR-window re-negotiation of network resource may be necessary. To allow for these accesses, we show how the use of a “contingency channel” for stored video can be used to resynchronize the delivery of video back with the original bandwidth allocation plan. To show the applicability of the VCR-window, we introduce a resource reservation scheme that can be used in conjunction with the VCR-window.

Once the video has been delivered to the end user's machine, the video must be decompressed and played back for the user. We examine the effects that software decompression of MPEG video has on processor performance. The software decompression of MPEG video typically makes poor utilization of processor caching, because the information used across frames in MPEG reconstruction does not typically stay in the cache long enough for the processor to benefit from its reuse. To examine the issues involved with the software decompression of video data, we examine two techniques for improving processor caching: 1) reducing the effective working set size of the decompression algorithm and 2) providing a prefetch instruction and the associated circuitry to "warm" the cache for data that is anticipated to be accessed.

To reduce the working-set, we introduce the notions of *vertical* and *horizontal* striping for the decompression of MPEG video. These traversal algorithms visit (and decompress) the macroblocks of the compressed video stream using a different order than implied by the MPEG standard. As a result, the macroblocks that may be needed by other frames have a higher probability of remaining in the cache and being re-accessed. To keep caches "warm" with macroblock data for video decompression, we examine the feasibility of prefetching data that is expected to be accessed in the near future and to prefetch the data before it is needed. Software controlled prefetching is supported on some current processors.

1.3 Outline of the Dissertation

The structure of the dissertation is as follows. In Chapter 2, we present necessary background for the rest of the chapters. We begin by describing our assumptions about the structure of the video-on-demand system. We also describe the work that has been introduced for video servers and for video retrieval techniques. A discussion of compression technologies and how burstiness is introduced into video streams is presented. Finally, we conclude the chapter with a description of a video capture testbed that we used to capture over 30 GBytes of video data for use as sample clips in this dissertation.

Chapter 3 deals with the problem of delivering a single compressed prerecorded video stream across networks. Specifically, we describe two broad classifications of smoothing techniques: window-based and non-window-based smoothing techniques. We present two window-based techniques that smooth bandwidth requirements based on some maximum delay (window) that each frame must adhere to. We then present the *critical bandwidth allocation* algorithm and the *optimal bandwidth allocation* algorithm, which do not adhere to a window, allowing burstiness to be smoothed over larger stretches of video. Finally, we compare, contrast, and summarize the differences between the various algorithms.

In Chapter 4, we focus on the problem of handling multiple video streams. Specifically, we introduce the VCR-window and describe how burstiness affects the ability to provide VCR functionality. We then describe a contingency channel mechanism that can be used for accesses outside of the VCR-window, where renegotiation of network bandwidth may need to be done. To show how the VCR-window can be used we describe an in-advance resource reservation scheme for scheduling network resources. Finally, we then use our sample video clips to show the effectiveness of the VCR-window and its affect on the underlying network manager.

Chapter 5 deals with the software decompression of MPEG video streams. We describe the following two methods for reducing miss rates of software MPEG video decompression: 1) reduce the working set size and 2) add a prefetch instruction to reduce processor cache misses. To reduce the working set size, we introduce the techniques called *horizontal* and *vertical striping*, which re-order the traversals during the reconstruction of macroblocks, making better use of data that resides in the cache. We then contrast and compare these algorithms based on cache simulations as well as an implementation of these algorithms. The alternative to re-ordering of macroblock traversals is the use of a software-controlled prefetch instruction that fetches data before it is needed. To show the effectiveness of a prefetching instruction for MPEG video decompression, we examine how two different prefetching strategies can be used to reduce the memory access penalties.

In Chapter 6, we review the contributions of this dissertation and summarize some of the key findings of this research. Finally, we present some future directions for research in these areas.

CHAPTER 2

PRELIMINARIES

“”The beginning is the most important part of the work.”-
Plato

In this chapter, we present the necessary background for the remaining chapters of this dissertation. We describe the current trends in video-on-demand systems, highlighting the work on video-on-demand servers and network transportation protocols. In addition, we describe the relevant details of image and video compression necessary for the understanding of this dissertation.

2.1 Video Retrieval Systems

Video retrieval systems involve two separate layers: the timely retrieval of information from video-based file systems and the real-time transfer of the data to end users. In this section, a description of academic and industrial efforts aimed at providing real-time access to video file systems is given followed by a description of high level protocols that attempt to reserve bandwidth specifically for video data.

2.1.1 Video-On-Demand Architectures

Video-on-demand (VOD) systems that have been proposed in the literature typically consist of three main components: a large archive server, an intermediate cache server, and clients (Figure 2.1). Recent developments in file systems and disk systems such as RAIDs (Redundant Array of Inexpensive Disks) allow file systems to achieve greater throughputs, alleviating some potential bottlenecks in the file sys-

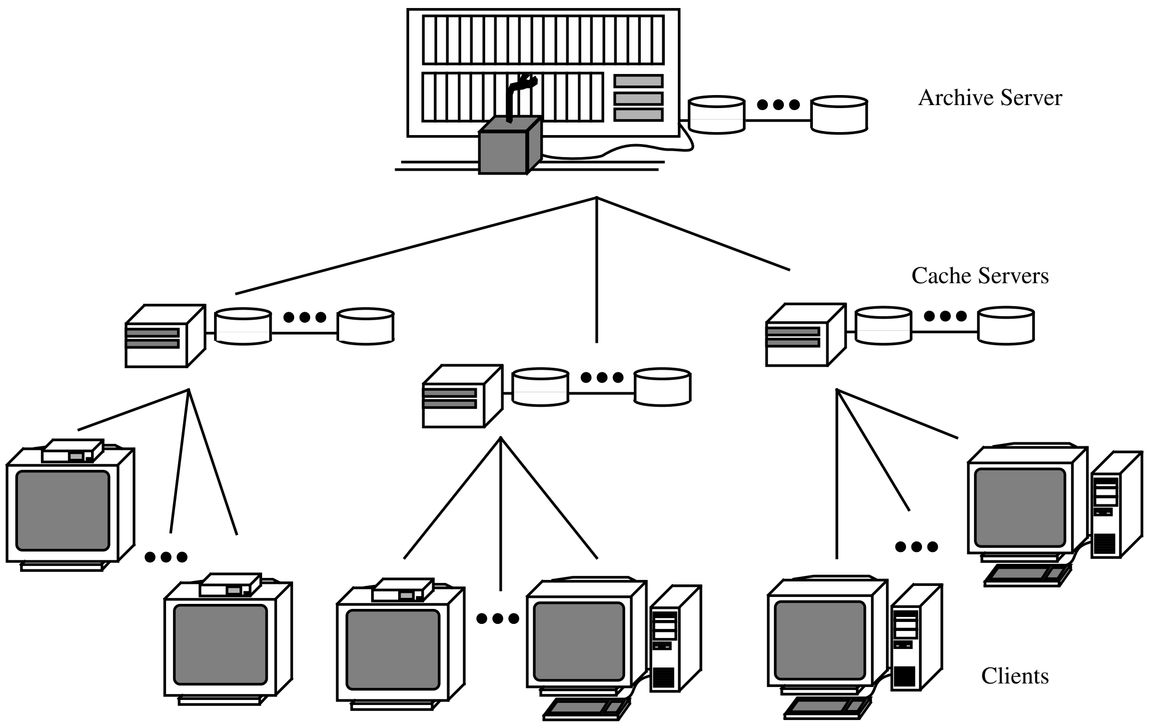


Figure 2.1: A Sample Video-On-Demand Architecture. This figure shows a sample hierarchical VOD architecture. The archive server typically consists of a large tertiary storage such as magnetic tape and large disks to hold entire movies. The cache servers act as load balancers for the more popular movies. Clients may consist of either workstations or set top boxes.

tem. As a result, it is now possible for a single VOD server to handle many clients simultaneously.

Typically, one or more archive servers reside at the top of the VOD hierarchy. These servers typically have large tertiary storage devices, such as magnetic tape jukeboxes for mass storage of video at a relatively low cost[21]. To allow for efficient transport of video to the caching servers, the archive servers download the requested data on to large disks on the archive server. The requested data is then transferred in bulk to the cache servers.

A cache servers consists of several large disks (on the order of 50-100 GBytes) so that the more popular videos can be cached in their entirety [1,32,47,56], distributing the load of access requests away from the archive servers. Video requests that are not present on the cache servers are requested from an archive server. Because the cache servers have enough buffering to hold on the order of 50-100 movies, the delivery of the video data from the archive server can be transferred in a large burst of bandwidth into the caching server's disk. We expect that a large portion of the video requests would be handled by the caching servers. One recent study suggests that the access frequencies for various movies can be modeled by a Zipf distribution model, and that only 38 movies cover more than 70% of all requests in a single week[12]. Note that the set of clients that a caching server may service is not fixed. A client may choose one of a number of nearby servers, just as a customer at a video rental store may go to the next convenient store in the vicinity, if the client cannot find the title of the movie that they are looking for.

The clients within the VOD architecture consist of either desktop computers with support for digital video or a set-top-box devoted solely for viewing compressed video. We assume that clients have some buffering available (either disk or RAM) for smoothing of network bandwidth requirements, but in order to keep consumer costs down, we expect that these client boxes do not have large reserves of buffering available. We also assume that the clients contain enough intelligence to create bandwidth allocation plans and to interact with the network and servers. Because of these constraints, the transportation of video from the cache server to the clients must be monitored so that the clients are not over-run with data or starved of data. In addition, these interactions must be made such that the underlying link layer can be used as effectively as possible.

The network provides the pathway between the video servers and their clients. The only assumption we make about the network is that it can provide network resource guarantees based either on some rate-based or real-time channel approach[2,84] and that the network provides some mechanism for in-advance reservations of bandwidth [14,31,83]. The network resource guarantees are necessary to ensure a quality of service (QOS) level to the users. We assume that the bandwidth reserved on these channels will be delivered assuming the channel has already been admitted. Furthermore, we assume that the admission control and the reservations for network resources can be done in advance, making network scheduling and load estimation easier for the network [55].

2.1.2 Video retrieval techniques

The delivery of data to a client can be handled in one of several ways. The simplest approach is to deliver the video in a best-effort fashion, resulting in no guarantees in QOS. As a result, packets may be arbitrarily delayed or dropped. Delivering high quality video in this environment has been studied in the literature [44]. At the other end of the spectrum, bandwidth can be reserved at the peak bandwidth requirement for the video. Allocating at the peak bandwidth requirements without bandwidth smoothing, however, results in very conservative estimates of the actual resources required. In between these extremes lies two other techniques, statistical multiplexing and bandwidth smoothing techniques.

Statistical multiplexing has been introduced to take advantage of the law of large numbers. This technique reserves bandwidth for a video channel very near to the mean of the video frame sizes expected[68]. Then, during the delivery of data, each video stream transmits a frame of video every 1/30th of a second, where the various multiplexed streams approach the sum of all the averages. This method typically can deliver 95% or more of all packets provided that the channel can deliver a significant number of streams, but no guarantees on the delivery of a packet are given. Statistical multiplexing techniques are suitable for live-video applications, where frames are being digitized and transmitted in real-time. For stored video applications, the use of statistical multiplexing implies that the server must transmit one frame every 1/30th of a second. This, however, results in a large amount of burstiness that the server has to account for in its retrieval off of slower storage devices, leading to scalability problems.

Unlike statistical multiplexing schemes, bandwidth smoothing techniques attempt to remove the burstiness of the video by introducing delay so that the bandwidth requirements can be reduced. Bandwidth smoothing techniques require buffering to reduce the variance in necessary bandwidths. For video-on-demand systems, bandwidth smoothing techniques are useful because they do not require that the time between video capture and video playback to be minimized. As a result, resource allocation plans can be made before playback begins, resulting in channels that can have bandwidth guarantees. In addition, bandwidth smoothing techniques allow the load on the video servers to be smoothed, resulting in a more scalable design.

2.2 Compression Technologies

In order to reduce the sheer amount of data that is required to represent an image, compression technologies have been developed that have a compression ratio of roughly 25:1 for still images and 50-100:1 for video. The additional compression derives from dependencies between frames. In this section, we present a high-level description of the Joint Photographic Expert's Group (JPEG) image compression standard and the Motion Picture Expert's Group (MPEG) video compression standard. The JPEG standard, while originally aimed at still-images, forms the basis for the MPEG video compression standard and is therefore necessary for the understanding of the MPEG video standard. Following our discussion of JPEG, we then give a high-level description of the MPEG motion video compression standard in as much detail as necessary for the discussions in this dissertation.

2.2.1 JPEG Image Compression

The JPEG image compression standard is a "lossy" image compression technique that uses knowledge of visual perception of the human eye to allow for a relatively high compression ratio. We refer readers interested in the lower level details to the JPEG standard and its companion introductory paper [81].

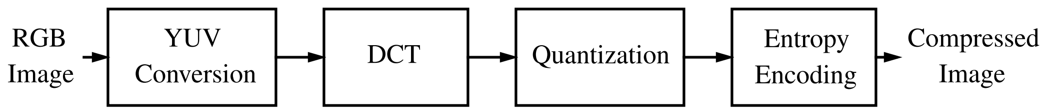


Figure 2.2: JPEG Overview. This figure shows the four main steps involved in compressing an image into the JPEG format. 1) Conversion of RGB color space to YUV color space, 2) Transformation into frequency domain via discrete cosine transform (DCT), 3) Quantization of DCT values, and 4) Entropy encoding (using either Huffman or arithmetic encoding).

The JPEG compression standard comprises of four main steps (Figure 2.2). The compression algorithm operates on 16 pixel by 16 pixel squares called *minimum coding units* or *macroblocks*. For each macroblock, a conversion from the red, green, blue (RGB) color space into the *YUV* color space is performed. This transformation allows the more important luminance component (*Y*) to be separated from the two chrominance channels *U* and *V*. The luminance component is essentially the brightness of each pixel. The human eye is most sensitive to small changes in the luminance component, while changes in the chrominance channels must be larger to be easily perceived by the human eye. Once the *YUV* transformation is complete, the luminance component is subdivided into four 8x8 pixel blocks, while the chrominance components are subsampled from two 16x16 pixels blocks into two 8x8 pixel blocks, one for the *U* channel and one for the *V* channel. As a result this step is somewhat “lossy” in the chrominance channels. Next, these six 8x8 blocks are compressed.

Compression in JPEG takes place in three steps: discrete cosine transformation (DCT), quantization, and entropy encoding. First, the DCT transforms each of the 8x8 pixel blocks into the frequency domain. This transformation moves the lower frequency components into the upper left corner of the block while moving the higher frequency components into the lower left corner. Thus, the average or DC level of each block is in the upper left corner. The other 63 coefficients are called the AC values. In the second step, the coefficients are quantized into discrete levels giving coarser distinctions for higher frequency components. This is considered the “lossy” part of the compression standard. The JPEG standard allows for varying qualities by allowing the coarseness of the quantization matrix to be specified by the user. The more bits that are used in the quantization (and hence more levels), the closer the reconstructed picture is to the original. Finally, the run-length encoded coefficients for each block are compressed with either Huffman or arithmetic encoding. To retrieve the original image the above process is reversed.

2.2.2 Motion JPEG and MPEG Video Compression

In order to compress video streams, a natural extension of the JPEG still image standard is to apply it to a stream of successive images for video, resulting in a stream of JPEG compressed images (Motion-JPEG or MJPEG). While relatively simple, this compression technique does not take advantage of similarities between frames. Because of high frame to frame correlation, the similarities between frames can be used to achieve considerably higher compression rates. The MPEG video compression standard is a layered video compression standard that results in VHS quality compressed video stream that has a bit rate of approximately 1.5 Mbit/second. To support future broadcast quality video, the MPEG-2 video standard results in the compression of 720x480 video at 60 frames per second into a stream of 4 to 10 Mbits[71]. The standard itself specifies a syntax that all MPEG encoded streams must follow, but within the standard there are many different encoding schemes that can be used.

At a high level, MPEG video sequences consist of several different layers that provide the ability to randomly access a video sequence as well as provide a barrier against corrupted information. The six layers within MPEG are shown in Table 2.1 along with their main functions within the standard.

All MPEG frames are encoded in one of three different ways: Intra-encoded (I-Frames), Predictive-coded (P-Frames), or Bidirectionally-predictive-coded (B-Frames). As shown in Figure 2.3, each frame type in an MPEG stream can depend on

MPEG Layer	Function
Sequence Layer	Random Access Unit: context
Group of Pictures Layer	Random Access Unit: video
Picture Layer	Primary Coding Unit
Slice Layer	Resynchronization Unit (within picture)
Macroblock Layer	Motion Compensation Unit Within Slice
Block Layer	DCT Unit Within Macroblock

Table 2.1: The MPEG Video Compression Layers

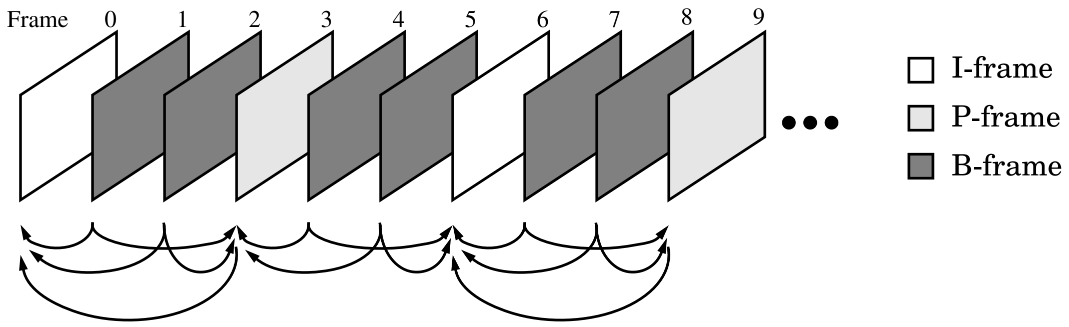


Figure 2.3: MPEG Frame Dependencies. This figure shows the frame dependencies for the macroblocks within an MPEG encoded video stream. The actual pattern of these frame types may vary depending on the amount of random access required.

up to two other frames, trading off degree of compression for random access and computation. I-frames are encoded as discrete frames, independent of adjacent frames. Thus they provide randomly accessible points within the video stream. Because of this, I-frames have the worst compression ratio of the three frame types. P-frames are coded with respect to a past I-frame or P-frame. The B-frames require a preceding and a following frame, which may be either I-frames or P-frames, in order to be decoded, but they offer the highest degree of compression. To allow for maximal compression and quality of picture, the individual macroblocks within B and P-frames may also be coded in several ways depending on the correlation of the macroblock to the frames on which they depend (seeTable 2.2). A macroblock in a B-frame, for example, can be encoded in one of five ways. All macroblocks can be intra-encoded

Frame	Types of macroblock encodings				
Type	intra	forward	backward	bidirectional	skipped
I	✓				
P	✓	✓			✓
B	✓	✓	✓	✓	✓

Table 2.2: MPEG Macroblock Encodings. This table shows the possible macroblock encodings for different frame types. I frames have no dependencies and each macroblock must be intra-coded. P frames can have up to 4 macroblock dependencies with non-zero motion vectors, while B frames can have up to 8 macroblock dependencies if bidirectionally encoded.

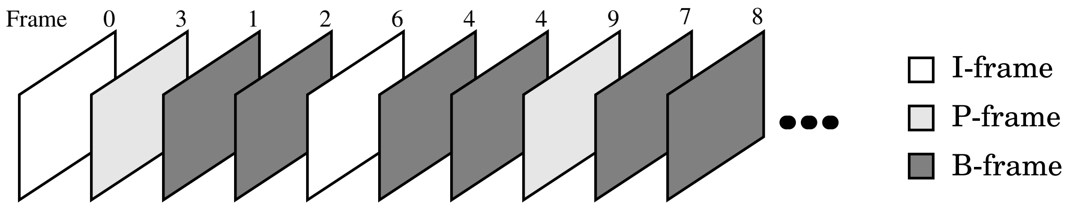


Figure 2.4: MPEG Encoding Order. This figure shows the actual ordering of the compressed frames within the MPEG encoded video stream for the sample clip shown in Figure 2.3.

(stand-alone) if there is not a high enough correlation with the frames on which they can be dependent on. Because the B-frames can depend on frames in the future, when encoding, the MPEG stream is rearranged so that the dependent frames are placed after the frames that they depend on. Figure 2.4 shows the reordering that occurs for the sample clip from Figure 2.3. For clarity of discussion, we refer to the frames upon which the P and B frames depend as *key* frames.

The Group of Pictures Layer (GOP) allows the user to group together any arbitrary number of frames starting with an I-frame. The GOP is a self-contained unit and therefore can be viewed by itself without looking at the GOPs surrounding it in the sequence. For MPEG encoded videos with multiple frame types, each GOP is generally set to start on every I frame. Thus, the example clip shown in Figure 2.3 is generally encoded with 6 frames per GOP. The actual number of frames required by each GOP, however, can vary and is not fixed by the MPEG standard.

Our research is mainly focused at the picture level for the delivery of compressed video across networks, and at the macroblock level for the reconstruction of MPEG video in software. To fully understand the lower layers of MPEG encoding, we refer readers to the MPEG compression standard which provides information down to bit level of how MPEG is encoded[8]. A higher level description of MPEG can be found in [54].

2.3 Video Compression and Burstiness

An understanding of how burstiness is introduced into a video stream can provide insight into the effective handling of compressed video. The MJPEG compression technique applies the JPEG compression standard to each individual frame within a video stream. As a result, the burstiness in a MJPEG video stream is purely a result of differences between frames. Because each macroblock must have an average DC value, the differences in compression are mainly a result of the difference in the high frequency components. For video streams that have roughly the same scene content, such as a typical video conference or lecture, the bit rate generated by MJPEG is fairly stable. As an example, consider the video sequence shown in Figure 2.5. The *Seminar* video consists of a speaker standing in front of an overhead projector presenting a talk. The inverted spikes that occur fairly regularly are a result of the speaker removing a transparency from the overhead projector, thus eliminating the need for a lot of high frequency components to represent the words and figures from the overhead projector. As a result, the difference in the various levels in the *Seminar* video are due to the transparencies and not the movement of the speaker. On the other hand, videos that have many different scenes generally have more burstiness due to the larger varying amounts of high frequency data. As an example, the movie *Speed* (see Figure 2.6) has frame averages that covered a wide range of bit-rates.

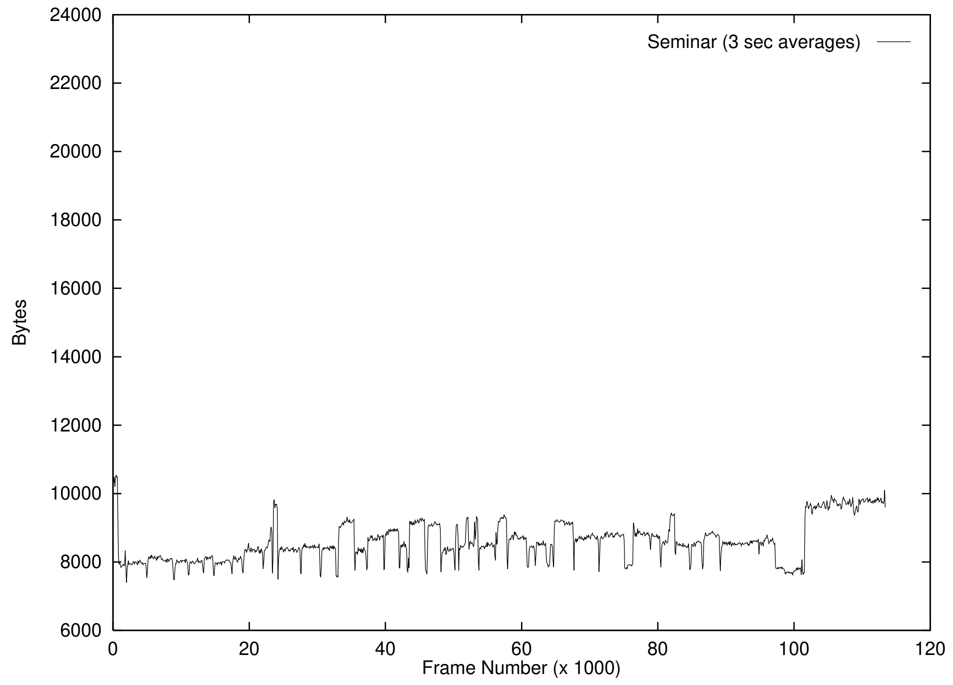


Figure 2.5: Seminar Example. This figure shows the 3 second frame averages (in Bytes) for a video recording of a M-JPEG compressed (320x240 @ 30 fps, 90 quality) seminar. This seminar shows a speaker presenting the seminar on an overhead projector with transparencies.

Because MPEG video compression techniques uses the same basic DCT compression algorithm as JPEG, the bit-rates for an all I-encoded video are similar to those of a MJPEG compressed video. To take advantage of temporal correlation between frames, MPEG compressed videos usually consist of a fixed pattern of I, P and B frames. While the use of P and B frames typically cause burstiness within the pattern, they do not really affect the long-term burstiness of the video data for two reasons. First, the P and B frames generally require fewer bits to represent, which results in a smaller variation within each frame type (P and B). Second, the frame sizes of P and B frames are not correlated with the sizes of their key I frame. This stems from the fact that good motion compensation techniques remove much of the high frequency data. As a result most of the bits in the frame are devoted to encoding vectors that represent the motion of the macroblocks between frames. Figure 2.7

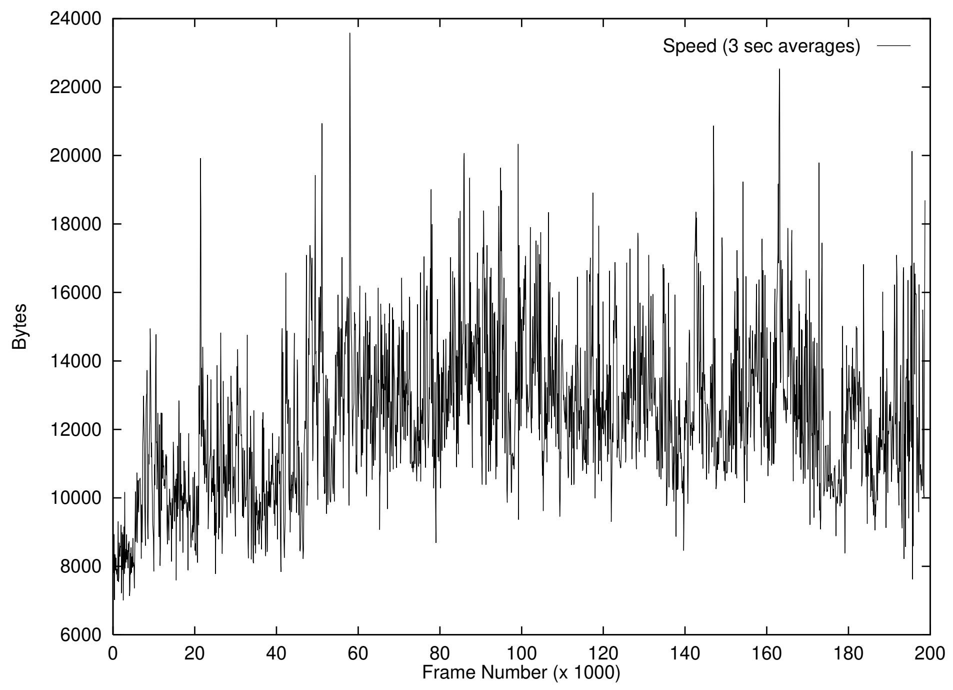


Figure 2.6: Movie Example.This figure shows the 3 second frame averages (in Bytes) for the movie *Speed* M-JPEG compressed (320x240 @ 30 fps, 90 quality).

shows how the pattern burstiness introduced by MPEG P and B frames differs from the long-term burstiness. In particular, note that the B-frames within the MPEG-encoded clip have a very small variance.

2.4 A Video Capture Testbed

In performing our experiments for this dissertation, we were interested in how the smoothing of bandwidth requirements for a single video affects the underlying services as well as how the interactions between videos may affect the utilization of the underlying network. In order to effectively test these, we required a video capture testbed capable of capturing a large amount of video data. In addition, capturing large amounts of video allowed us to study the differences that varying subjects of video had on compression and the burstiness that they introduced into the video

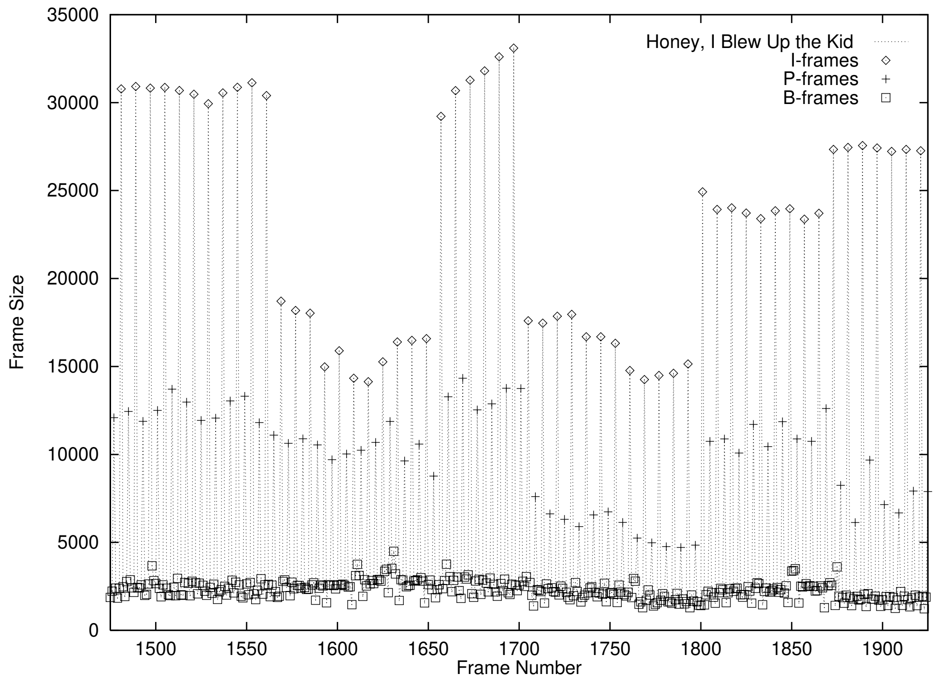


Figure 2.7: Burstiness Example. This figure shows how burstiness is introduced in a sample MPEG-encoded video clip from the Walt Disney movie *Honey, I Blew Up the Kid*. Pattern burstiness is introduced as a result of the encoding pattern (i.e. I, P, and B frames). Scene burstiness (or long term burstiness) is the burstiness between the different scenes within the movie. The scene burstiness is exhibited by the differences in the I-frames.

stream. Using this PC-based test bed, we were able to capture many videos of varying length and subject material for our video library.

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 containing the C-Cube Microsystems' Motion-JPEG chip, the CL550. Because the smoothing algorithms we introduce are most sensitive to the changes in scene content, we felt the additional (order of magnitude) cost for a real-time MPEG encoder would not significantly change our results. Furthermore, because the basic routine for encoding I-frames within an MPEG video are based on the JPEG compression standard, the frame sizes for our experimental video data are roughly equivalent to all I-frame encoded MPEG video movies. The MiroVideo board digitized the movies at 640x480 and then subsampled them to 320x240 with guaranteed VHS picture quality.

2.5 A Video Library

Using our PC video capture testbed, we digitized 20 video clips, representing 31 hours of video, which totalled 38.5 GBytes of JPEG-compressed video data. In digitizing the video data, we attempted to capture a variety of different movies in order to have a fairly representative set of movies that might be handled. 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 scenes with lots of detail. As a result, the *1993 NCAA Final Four* video had the highest average bit rate. In addition, we captured several seminars and lectures to study the compression of “educational” videos. Because these videos were single scene videos, they resulted in the smallest variation in frame sizes. The rest of the movies are a mix of conventional entertainment containing a wide range of scene content, including digital effects and animations. As can be seen by the statistics for these movies found in Table 2.3, the captured data resulted in a wide range of compression ratios. Table 2.3 also shows that the bit-rates for these Motion-JPEG encoded movies were higher than the bit-rate specified for the MPEG video standard.

The 64 frequency components of the 8x8 block produced by the DCT algorithm are quantized at a level of coarseness determined by a quantization matrix. This matrix can be adjusted by scaling an overall picture quality factor. We captured the movie *E.T. - the Extra Terrestrial* at three different qualities, 75, 90, and 100. These numbers do not express linearly the quality that is seen by the user. For our sample *E.T.* video, picture qualities of 75, 90, and 100 resulted in bits per pixel measurements of 0.66, 0.94, and 1.64 bits per pixel, respectively. According to an introductory

JPEG paper, 0.66 bits per pixel corresponds to “good to very good” quality, 0.94 bits per pixel corresponds to “excellent” picture quality, and 1.64 bits per pixel corresponds to a quality that is “usually indistinguishable from the original”[81].

For comparison of the different movies, we have graphed the 3 second frame averages for all the movies digitized (Figure 2.8 to Figure 2.27). These graphs show the pattern of variation within each movie and how the burstiness was introduced. Note the low variation in the three seminar videos in contrast to the other videos.

Title	Quality	Total Size (GBytes)	Length (min)	Ave. Size (bytes)	Largest Frame	Smallest Frame	Mbps	Std. Dev.
Beauty and Beast	90	1.82	80	12661	30367	2701	3.04	3580
Big	90	2.26	102	12346	23485	1503	2.96	2366
Croc. Dundee	90	1.82	94	10773	19439	1263	2.59	2336
E.T.	100	3.11	110	15749	30553	6827	3.78	3294
E.T.	75	1.24	110	6305	14269	1511	1.51	1840
E.T.	90	1.78	110	9022	19961	2333	2.17	2574
Home Alone 2	90	2.35	115	11383	22009	3583	2.73	2480
Honey, I Blew Up the Kid	90	2.12	85	13836	23291	3789	3.32	3183
Hot Shots 2	90	1.92	84	12766	29933	3379	3.06	3240
Jurassic Park	90	2.50	122	11363	23883	1267	2.73	3252
Junior	90	2.71	107	14013	25119	1197	3.36	3188
Rookie of the Year	90	2.22	99	12435	27877	3531	2.98	2731
Seminar	90	0.98	63	8604	10977	7181	2.07	592
Seminar2	90	1.08	68	8835	12309	1103	2.12	608
Seminar3	90	0.88	52	9426	11167	7152	2.26	690
Sister Act	90	2.06	96	11902	24907	1457	2.86	2608
Sleepless in Seattle	90	1.72	101	9477	16617	3207	2.28	2459
Speed	90	2.46	110	12374	29485	2741	2.97	2707
Total Recall	90	2.34	109	11978	24769	2741	2.88	2692
1993 NCAA Final Four	90	1.21	41	16456	29565	2565	3.95	4138

Table 2.3: Video Movie Library Statistics. This table shows the statistics that were gathered for the video clips in our video movie library.

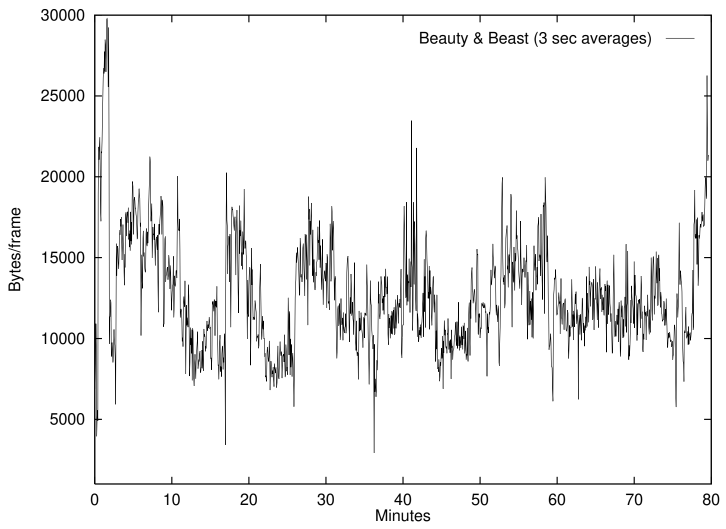


Figure 2.8: ***Beauty and the Beast*** **- 3 second frame averages**

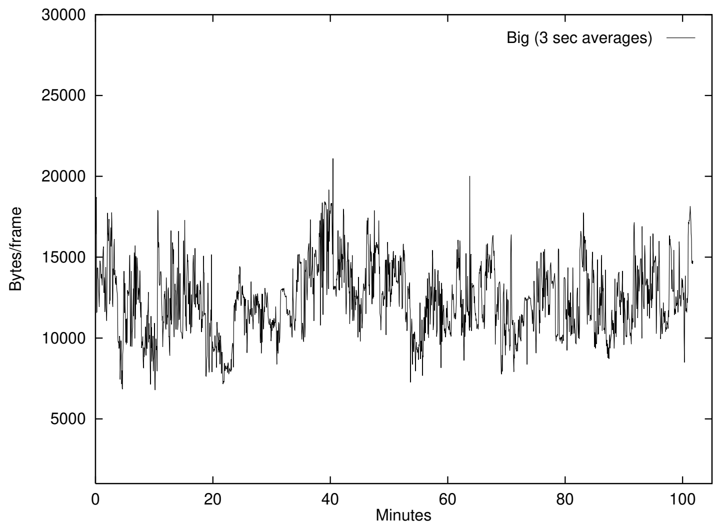


Figure 2.9: ***Big*** **- 3 second frame averages**

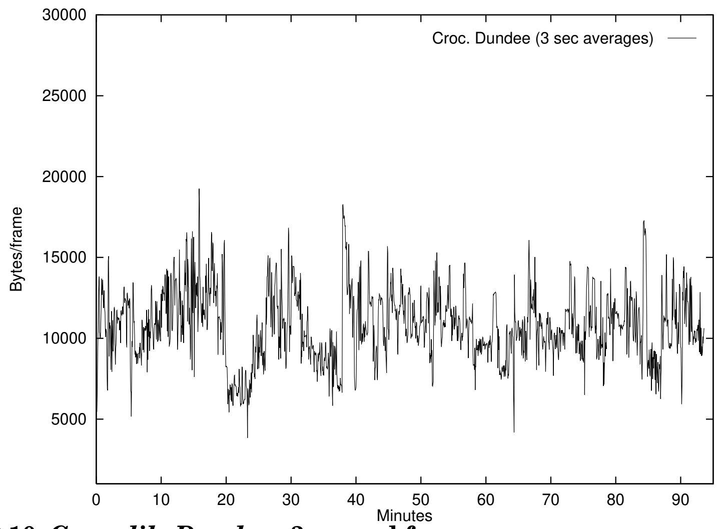


Figure 2.10: ***Crocodile Dundee*** **- 3 second frame averages**

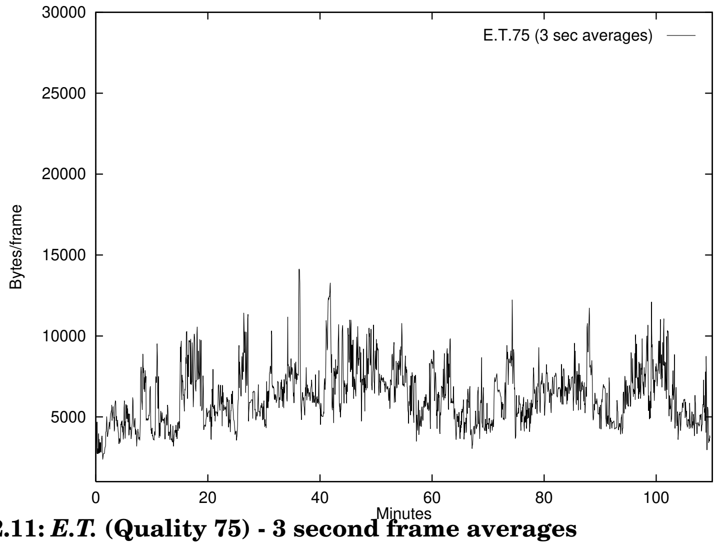


Figure 2.11: ***E.T.*** **(Quality 75) - 3 second frame averages**

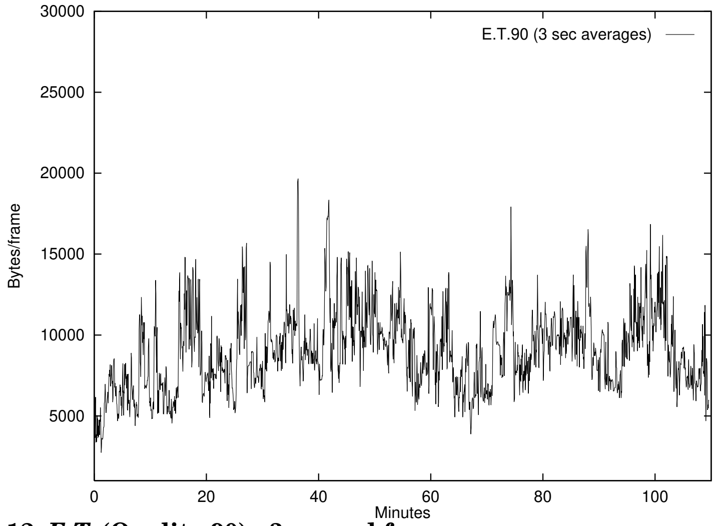


Figure 2.12: ***E.T.*** **(Quality 90) - 3 second frame averages**

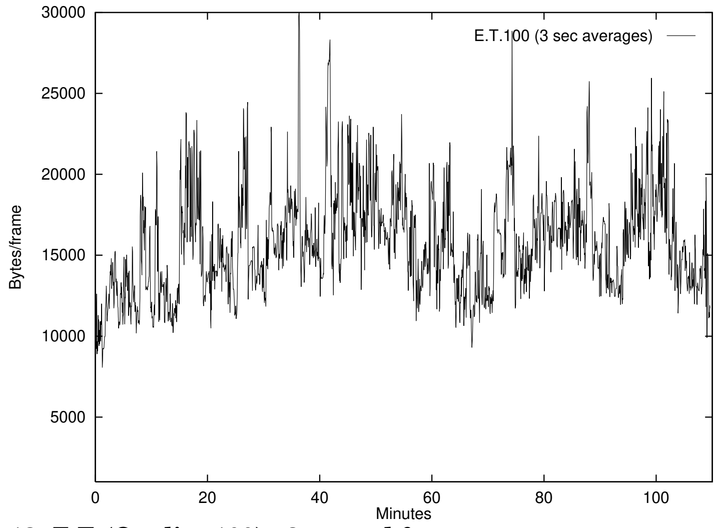


Figure 2.13: ***E.T.*** **(Quality 100) - 3 second frame averages**

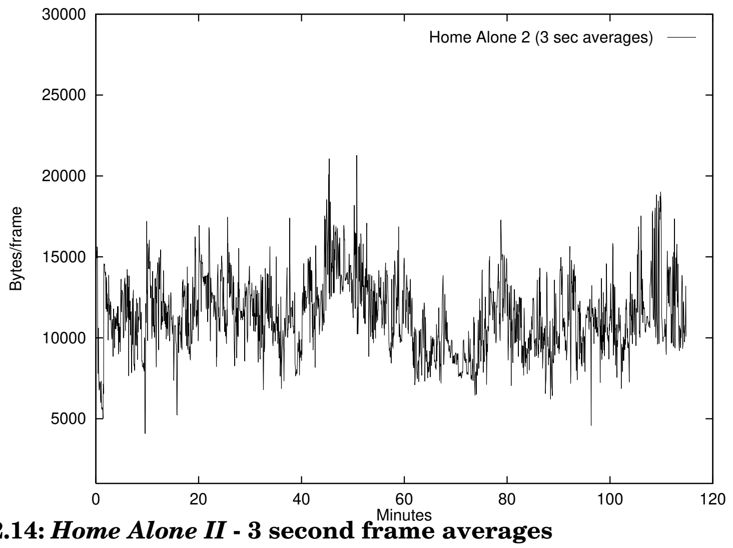


Figure 2.14: ***Home Alone II*** **- 3 second frame averages**

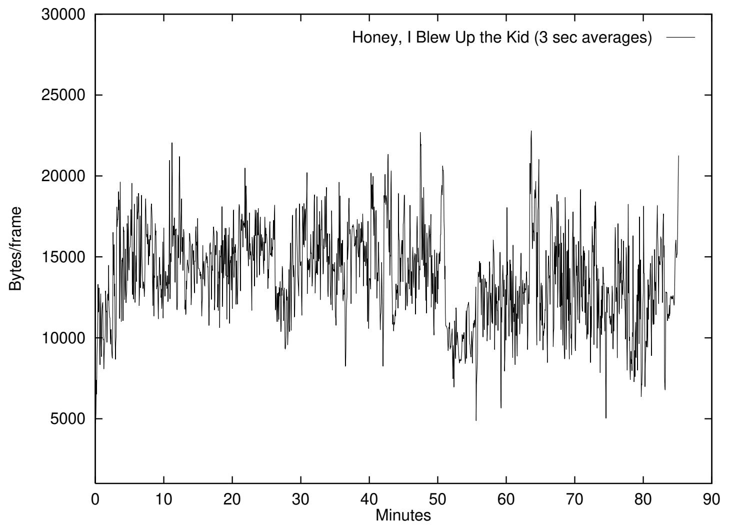


Figure 2.15: ***Honey, I Blew Up the Kid*** **- 3 second frame averages**

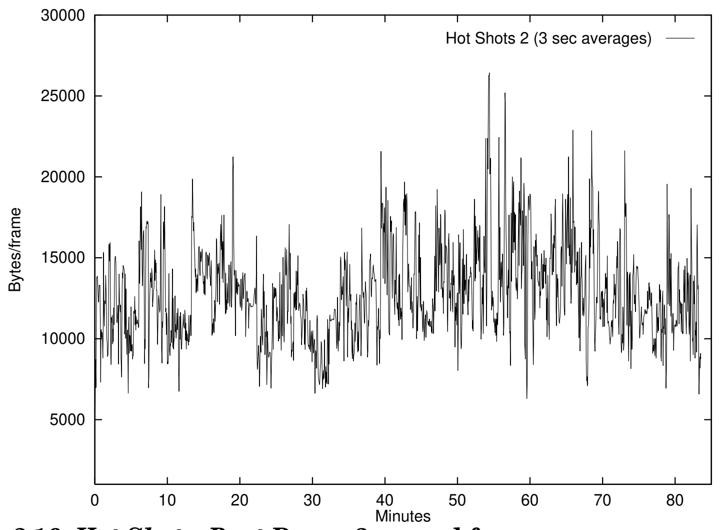


Figure 2.16: ***Hot Shots, Part Deux*** **- 3 second frame averages**

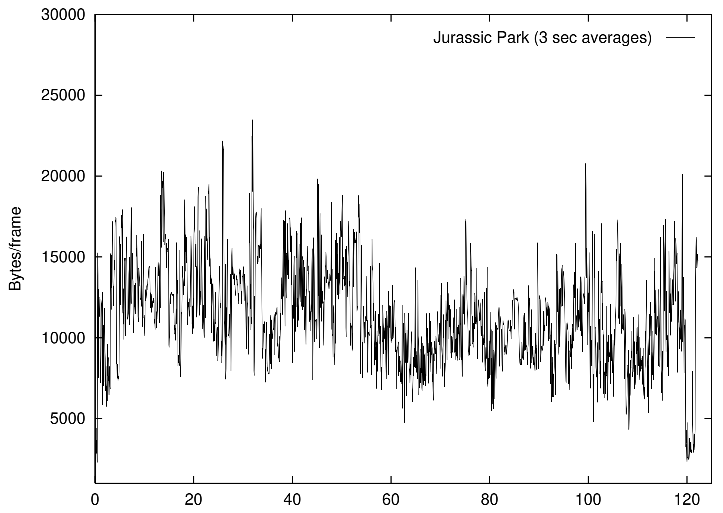


Figure 2.17: ***Jurassic Park*** **- 3 second frame averages**

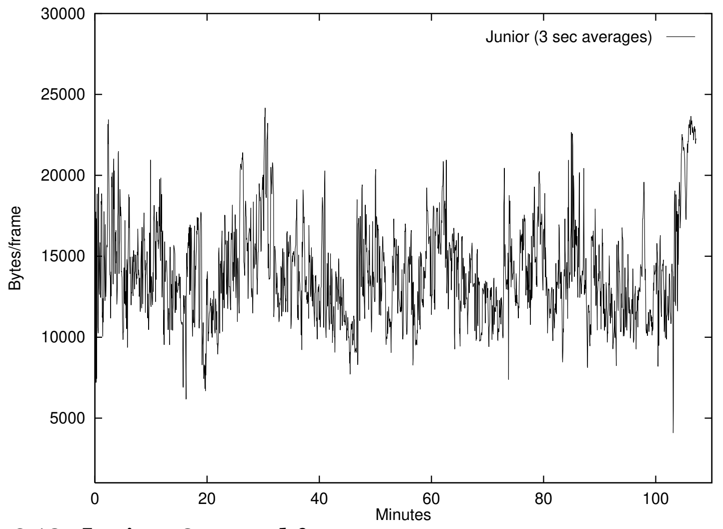


Figure 2.18: ***Junior*** **- 3 second frame averages**

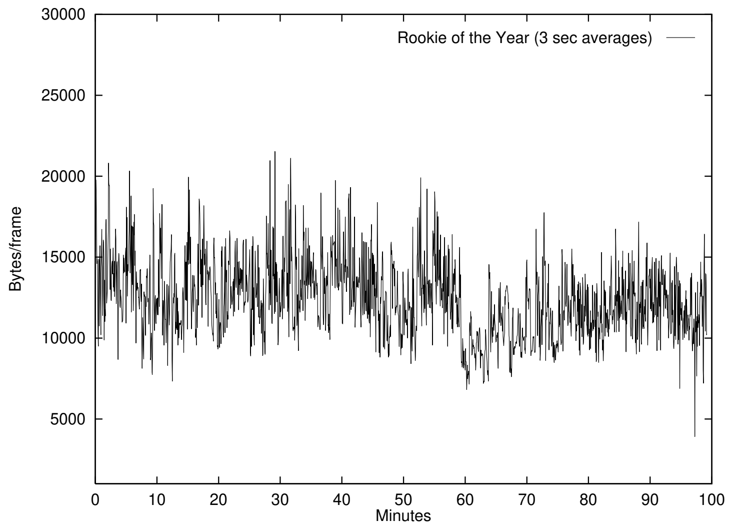


Figure 2.19: ***Rookie of the Year*** **- 3 second frame averages**

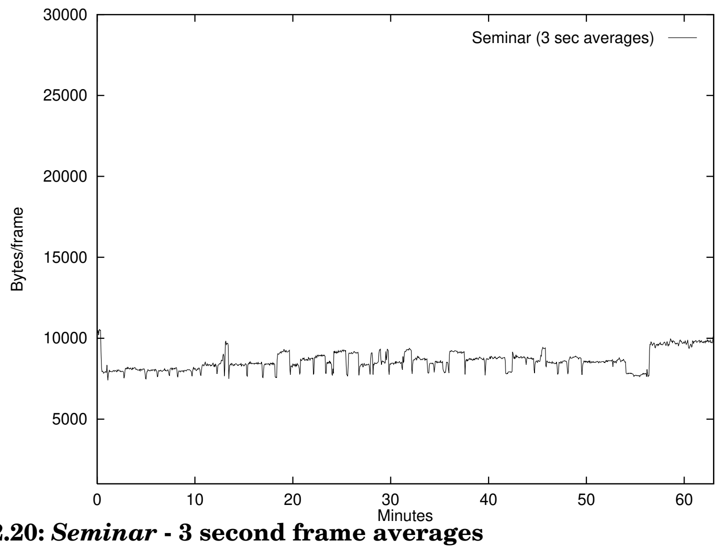


Figure 2.20: *Seminar* - 3 second frame averages

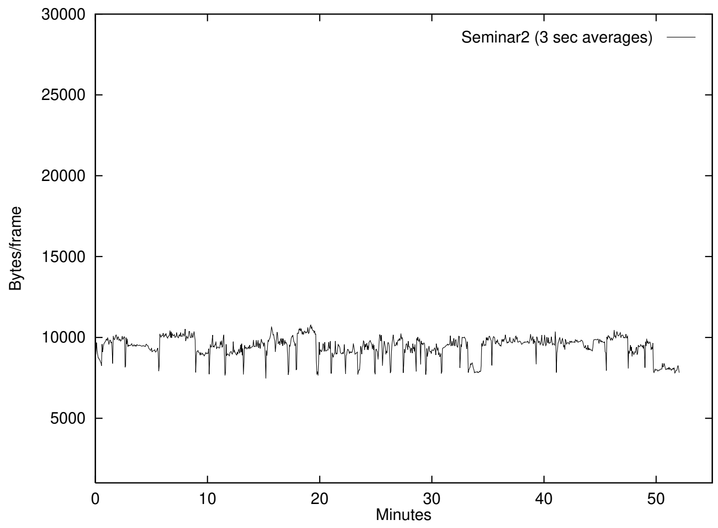


Figure 2.21: *Seminar2* - 3 second frame averages

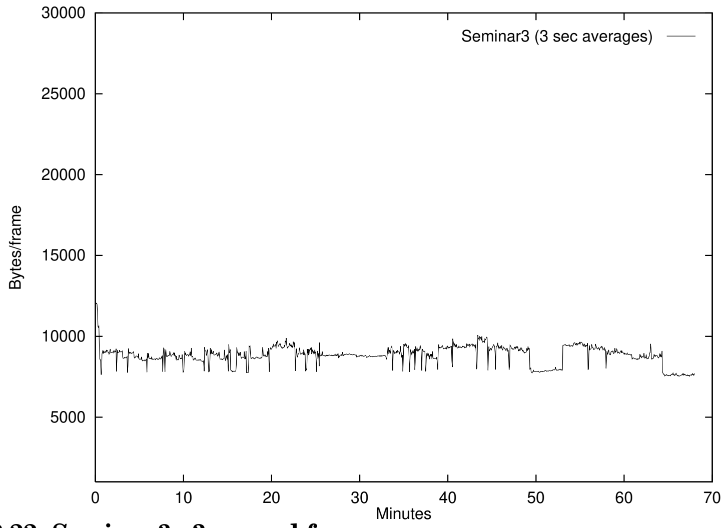


Figure 2.22: *Seminar3* - 3 second frame averages

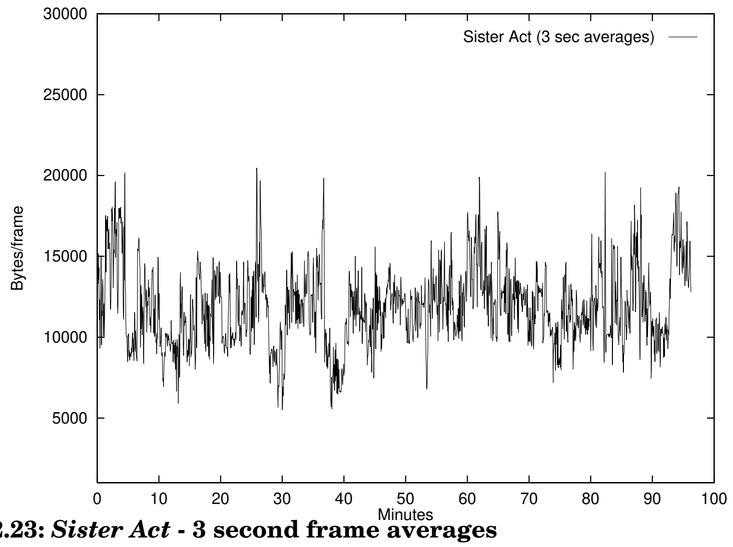


Figure 2.23: ***Sister Act*** **- 3 second frame averages**

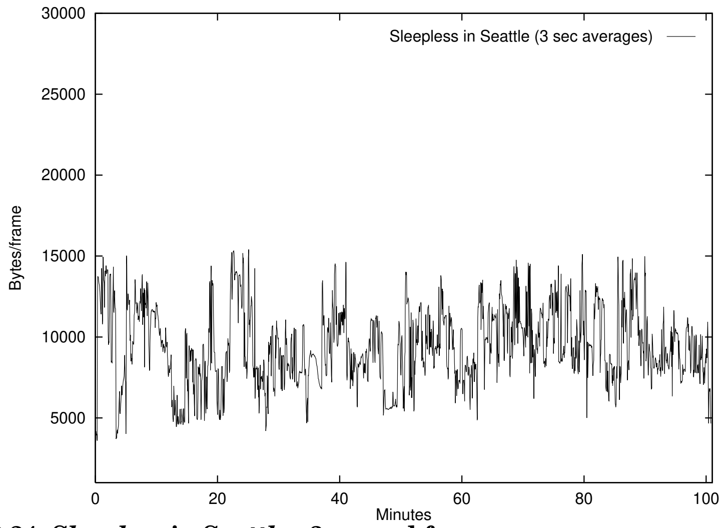


Figure 2.24: ***Sleepless in Seattle*** **- 3 second frame averages**

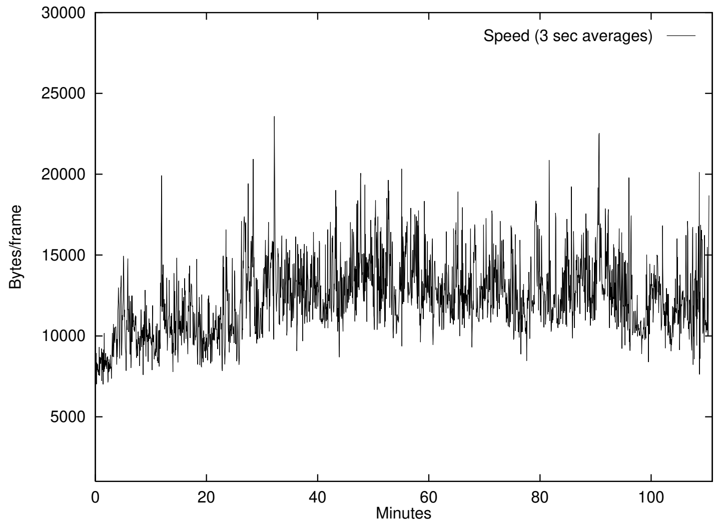


Figure 2.25: ***Speed*** **- 3 second frame averages**

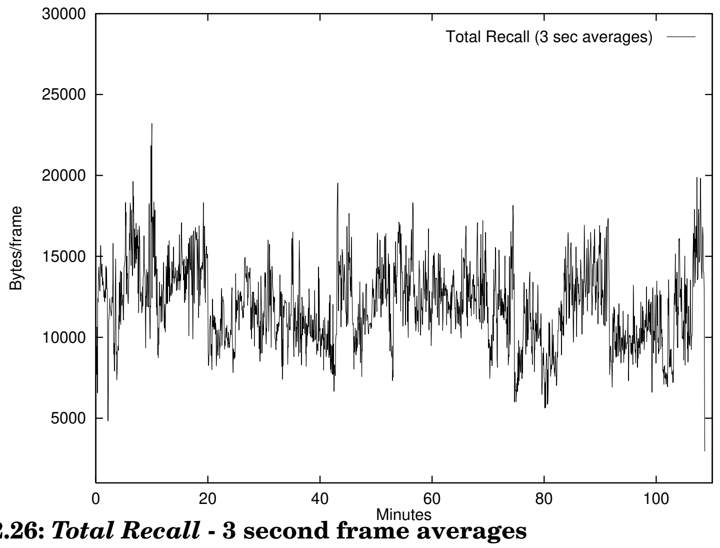


Figure 2.26: ***Total Recall*** **- 3 second frame averages**

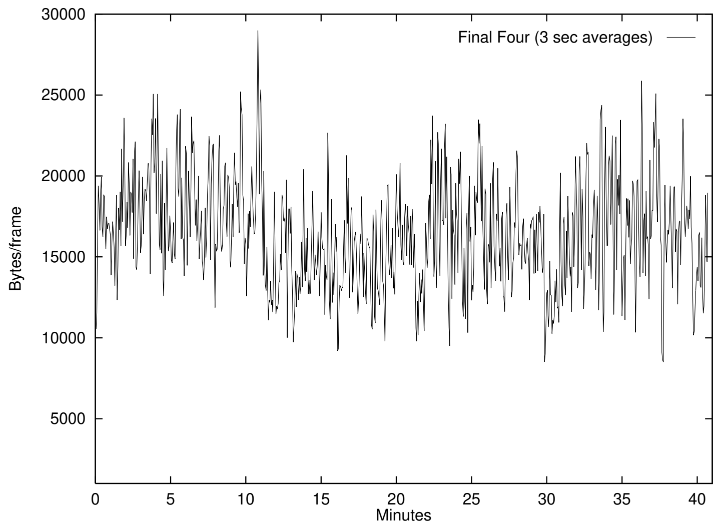


Figure 2.27: ***1993 NCAA Final Four*** **- 3 second frame averages**

CHAPTER 3

BANDWIDTH SMOOTHING ALGORITHMS

"Never underestimate the bandwidth of a station wagon full of tapes hurtling down the highway." - Andrew S. Tanenbaum, Computer Networks, Second Edition, p. 57

3.1 Introduction

In this chapter, we address the issues involved with smoothing the bandwidth requirements for a single stream of video data. 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 complicates the problem of network 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 limited. As a result, live-video applications may have to settle for weakened 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[11,69], smoothing in exchange for delay[52], jitter control[63,78], and adjusting the quality of service to fit the resources available[59,60]. For stored video applications, on the other hand, the system can take a flexible approach to the latency of data delivery. In particular, it 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 wasting of network bandwidth through overstatement of bandwidth requirements.

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 compensated for by filling the prefetch buffer in advance of each burst, by delivering more bytes than needed across the network, and draining it in the course of the burst. The size of the prefetch buffer 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 may remain relatively bursty. With a larger buffer, 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 that place a high demand upon the buffer.

In this chapter, we examine how the addition of a smoothing buffer can smooth the necessary bandwidth requirements from the underlying network for a single video stream. We present smoothing techniques that fall into two broad categories: window-based and non-window-based smoothing techniques. We refer to algorithms that smooth bandwidth based on some maximum number of frames as *window based* smoothing algorithms. On the other hand, we refer to algorithms which make smoothing decisions based on the size of the smoothing buffer alone as *non-window* based smoothing techniques. We introduce the notion of *critical bandwidth allocation* for the delivery of compressed prerecorded video. This algorithm minimizes the number of bandwidth increases as well as the peak bandwidth requirement for the continuous playback of stored video. To the best of our knowledge, the critical bandwidth allocation approach is the first reported work that uses non-window based buffering for the delivery of stored video [22]. In addition to the critical bandwidth algorithm, we present an *optimal bandwidth allocation* algorithm that also minimizes the total number of changes required to play back stored video.

We present window-based and non-window-based smoothing techniques in Section 3.2 and Section 3.4, respectively. In these sections we state the theorems that apply to the use of the critical bandwidth techniques. For clarity of presentation, we have put all the proofs of these theorems at the end of the chapter. In Section 3.4, we compare the various smoothing algorithms and demonstrate the key differences between the algorithms. Finally, we summarize our findings in Section 3.5

3.2 Window-Based Smoothing Algorithms

In this section, we describe two window-based smoothing algorithms for the delivery of video data. Because these algorithms smooth bandwidth based on some maximum number of frames (or window size), the maximum amount of prefetch is determined by both the window size and the size of the buffer used for smoothing. The window size can be picked to be a multiple of the video encoding pattern for MPEG compressed video or can be a multiple of the bandwidth allocation unit of the underlying link layer. Window-based smoothing is particularly suitable for use in live video conferencing applications because they result in a maximum delay equal to the window size.

3.2.1 Average Bandwidth Allocation

A simple window-based method for easing bandwidth fluctuations is to group some number of frames together into a *chunk* and send the frames across the network at the average bandwidth requirement for the chunk. The use of averaging algorithms has been proposed for both live and stored video applications [74,52]. By using this method, clients can guarantee that the bandwidth needed is constant throughout the chunk. The amount of smoothing, however, is directly related to the chunk size. Using this constant bandwidth for the entire chunk implies that the end

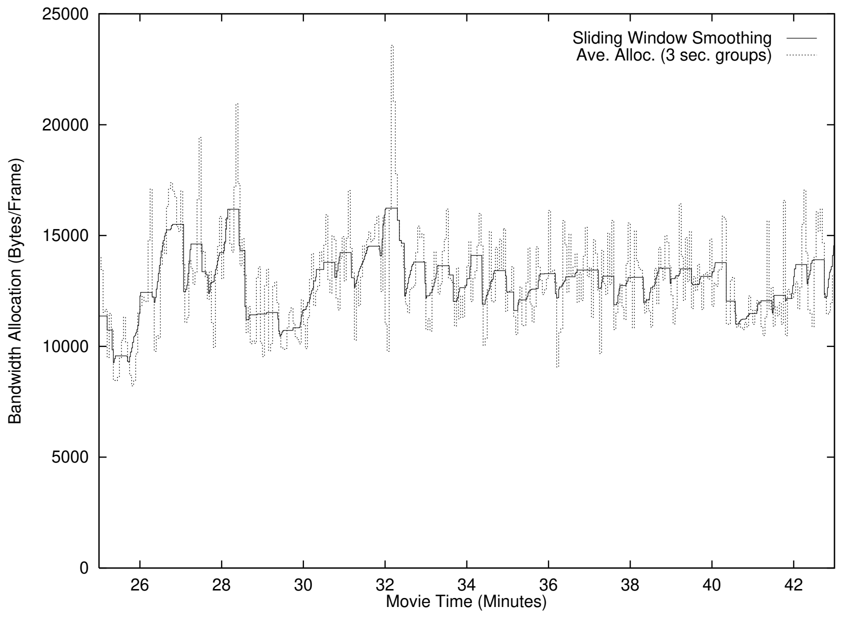


Figure 3.1: Average Bandwidth Allocation vs. Sliding Window Smoothing. Each line in the graph represents a sample of bandwidth requests from an 18 minute sample from the movie *Speed*. The video has been M-JPEG encoded at 30 frames per second. The burstiness is therefore due entirely to differences in scene content. Each algorithm was run assuming a maximum buffer size of 2 MBytes. The dotted line shows the average allocation algorithm for 90 frame chunks. The solid line represents the sliding window smoothing applied to 45 frame chunks with a window size of 11 chunks.

user is not guaranteed that local buffer starvation does not occur during the display of the chunk unless it has buffered at least one chunk ahead. The end user is, however, guaranteed that the maximum delay between the transmission and playback of the video is proportional to the chunk size used. For stored video applications, this maximum delay is twice the size of the chunk. Using bandwidth smoothing with very large chunk sizes becomes impractical because of this buffering requirement. In addition, because this algorithm merely groups a fixed number of frames together, no smoothing occurs across chunks. A sample bandwidth allocation graph using the average allocation algorithm is shown in Figure 3.1. The pseudo-code for the average allocation algorithm along with the pseudo-code for all the other algorithms can be found in the appendix.

3.2.2 Sliding Window Smoothing

Rather than simply averaging within fixed chunks, a sliding window can be used to smooth within a larger region. A moving window of *n* chunks is smoothed by shifting large bandwidth requirements in a chunk to earlier chunks (within the window) that stand below the average for the window as a whole. Thus, for a window of size *n* chunks, the bandwidth allocation for a chunk *i* is set to the average of all the frames in chunk *i* to chunk *i+n-1*. For chunk *i+1*, the bandwidth allocation is set to the average of all the frames in chunk *i+1* to chunk *i+n*. This averaging has the effect of causing the data to be prefetched in advance of bursts of large frames. However, redistribution is limited by the size of the window, so peaks and valleys will still occur.

Figure 3.2 shows the affect that changing the window size has for the sliding window smoothing algorithm.The amount of smoothing is determined by the window size. A window size of 1 yields no smoothing at all and the graphs are equivalent to the average bandwidth allocation algorithm. With sufficiently large windows considerable smoothing is possible. This smoothing is accomplished through prefetching and requires buffer space in the receiver, but far less than would be required to achieve equivalent smoothing with the average allocation algorithm. Figure 3.1 shows a sample sliding window bandwidth allocation plan for the movie *Speed* compared with the average allocation algorithm.

Sliding window smoothing, however, has some drawbacks for planning bandwidth allocation. While the number of adjustments in bandwidth is less than required for the average allocation algorithm, frequent adjustments are still necessary. Furthermore, rises in bandwidth are generally accomplished in several steps, each possibly requiring negotiation with the network manager, while drops in bandwidth are sharper.

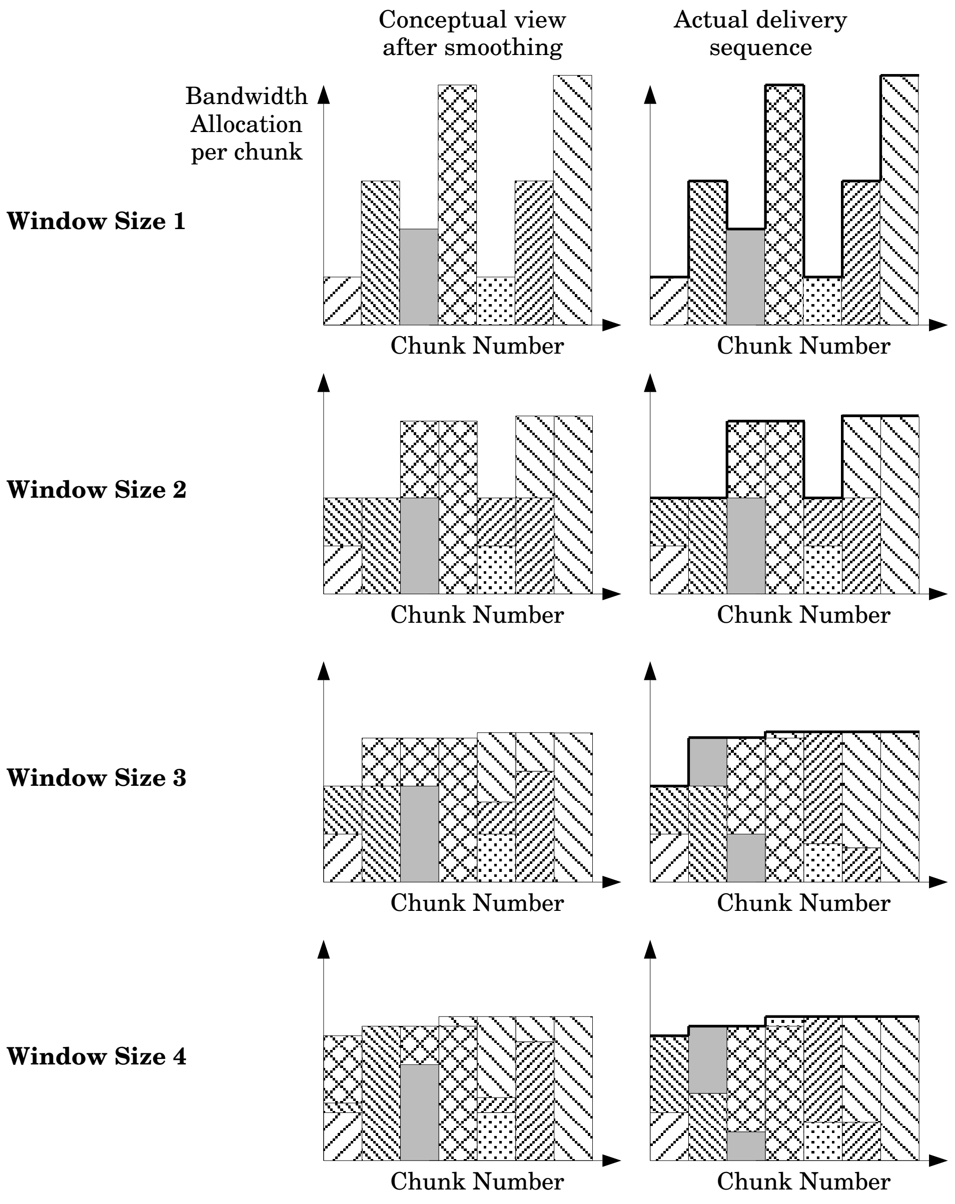


Figure 3.2: Sliding Window Smoothing Example. This figure shows the affect that the window size has on the smoothing for the sliding window smoothing algorithm. The heavy solid lines on the right represent the bandwidth allocation plans that are made. Note how the window size (particularly for the window size of 1 and 2) limits the amount of smoothing to the window.

3.3 Non-Window-Based Smoothing Algorithms

While window-based smoothing algorithms are suitable for live-video applications because they have a maximum delay between transmission and playback, they may not take full advantage of the buffer available for smoothing. Because the constraints between transmission and playback are relaxed for stored-video, creating *a priori* bandwidth plans that maximize the usage of the buffer allows the network management to be made simpler. In this section, we introduce the notion of *critical bandwidth allocation* for stored video playback. These algorithms base the bandwidth allocation decisions based on the *a priori* knowledge of the video frames that are available. By taking advantage of this knowledge, a bandwidth plan for retrieval that minimizes the range of bandwidth values for playback is possible.

3.3.1 Critical Bandwidth Allocation

The critical bandwidth allocation 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 only calculates the minimum bandwidths that are necessary for continuous playback, the buffer size requirement for continuous playback, may be fairly substantial. The buffer size requirement, however, is generally not as large as one required by a single constant bandwidth allocation for the entire video. Finally, the CBA algorithms result in a bandwidth plan that (1) does not require prefetching (and, hence, delay) for playback to begin, and (2) results in a monotonically decreasing sequence of bandwidth allocations. For clarity, we say that a bandwidth allocation plan consists of *runs* of constant bandwidth allocations.

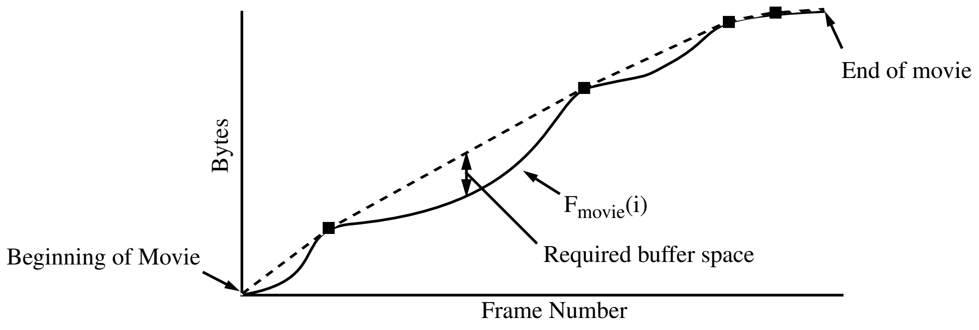


Figure 3.3: Critical Bandwidth Allocation Example. The solid line in the graph shows a possible graph for $F_{movie}(i)$, while the dotted line shows a plan determined by the CBA algorithm. The dotted set of lines show the critical bandwidth allocation algorithm's bandwidth plan that requires 5 decreases 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 $F_{movie}(i)$.

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 Figure 3.3). 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::

$$F_{movie}(i) \le TBR(i)$$

The critical bandwidth allocation 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 $F_{movie}(i)$, where the slope of each line (run) determines the bandwidth allocation that is required for that run. This is shown in Figure 3.3 as a convex arc around $F_{movie}(i)$. 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 monotically 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 any constant bandwidth allocation plan for the entire movie must have this minimum buffer size but is typically much larger in size. This leads to the following theorem which we prove in the appendix.

Theorem 1 : *The critical bandwidth allocation algorithm with no buffer limitation results in a strictly decreasing sequence of bandwidth allocations*

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

$$CB_0 = \max_{1 \le i \le N} \left(\frac{\Sigma_{j=1}^{i} \; frame_j}{j} \right)$$

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

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

The use of the critical bandwidth allocation 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.

3.3.2 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 critical bandwidth allocation 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 Figure 3.4). Thus.

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

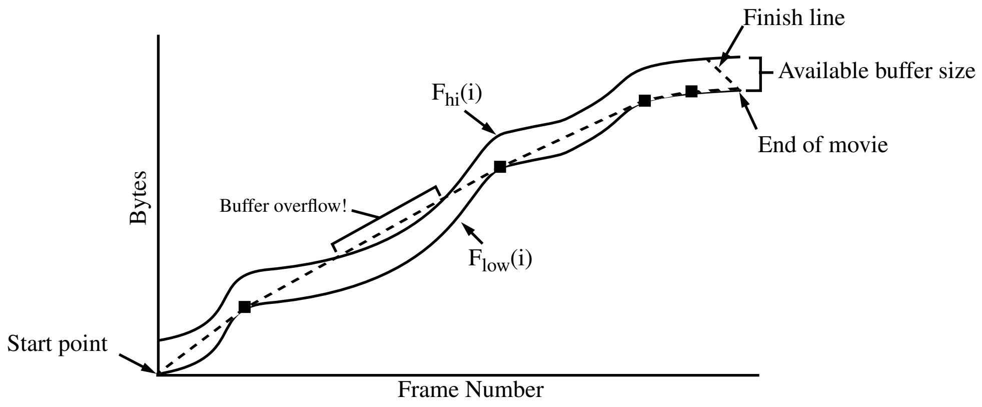


Figure 3.4: Critical Bandwidth Allocation with a Maximum Buffer Constraint. This figure shows a possible graph for $F_{hi}(i)$ and $F_{low}(i)$. The total delivered bandwidth (up to some frame i), must lie between $F_{hi}(i)$ and $F_{low}(i)$ or buffer overflow or underflow will occur. The dotted lines represent 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.

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 critical bandwidth allocation 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 bandwidth such

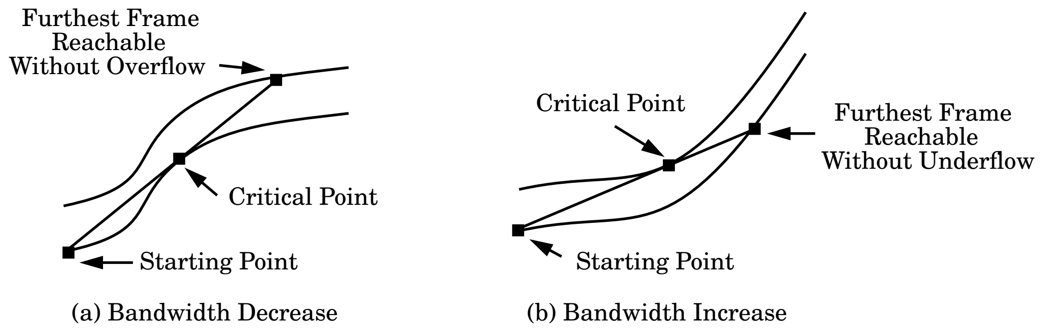


Figure 3.5: Critical Bandwidth and Critical Point Example. On the left is an example calculation of a critical point using the maximum buffer constraint in which a bandwidth decrease is required in the next run, while on the right is a similar calculation for the case where a bandwidth increase is required in the following run.

that the buffer limitations are not violated for the largest number of frames. Figure 3.5 shows two representative examples of critical points for runs that require a decrease and increase in bandwidth in the following run. As a result, 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 Figure 3.5, we use the terms *hub* to refer to the part of the run that precedes the critical point and *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. This search requires the calculation of an initial buffer occupancy between the maximum buffer

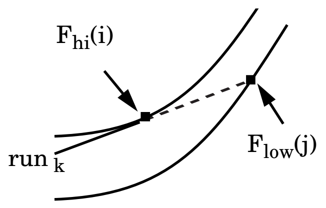


Figure 3.6: Bandwidth Increase Search Example. For the calculation of run k+1 a search is performed on the line connecting $F_{hi}(i)$ and $F_{low}(j)$ to find a starting point such that the critical point for the next run is as far out in time as possible.

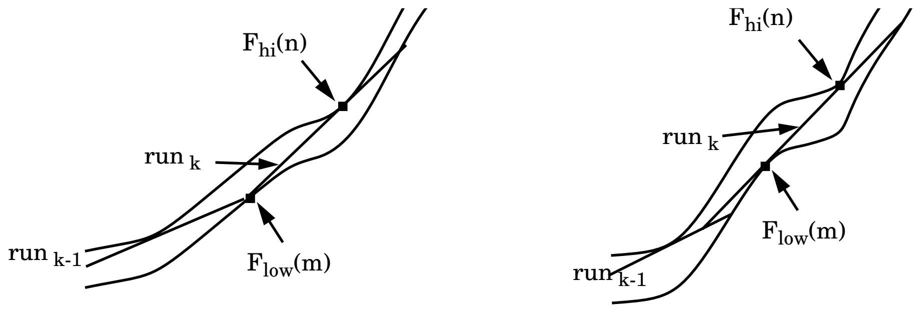


Figure 3.7: Bandwidth Search Example. This figure shows the two representative cases that arise when the search for run *k* results in another bandwidth increase required in run *k+1*. In (a), $F_{low}(m)$ lies on the line created by run_{k-1}. In (b), $F_{low}(m)$ does not lie on run *k-1*.

size and 0 because the frontier of the run connects $F_{hi}(i)$ and $F_{low}(i)$. Hence, bandwidth decreases result in a convex arc around point 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 at the end of run *k* along the frontier of run *k* as shown in Figure 3.6. 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 Figure 3.7 and

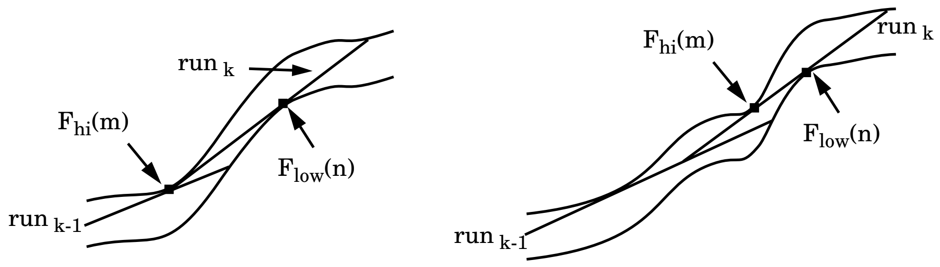


Figure 3.8: Bandwidth Search Example. This figure shows the two representative cases that arise when the search for run *k* will result in a bandwidth decrease required in run *k+1*. In (a), F_{low}(m) lies on the line created by run$_{k-1}$. In (b), F_{low}(m) does not lie on run *k-1*.

Figure 3.8, respectively. This results in the following two properties concerning the bandwidth increase search for a run *k*.

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 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 Figure 3.7, 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 3.9 shows a sample construction using the critical bandwidth algorithm with maximum buffer constraint.

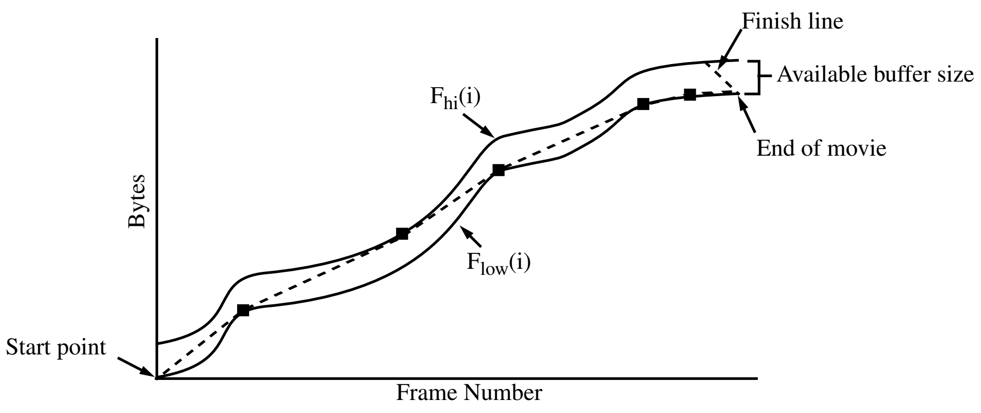


Figure 3.9: Critical Bandwidth Allocation with Maximum Buffer Example. This figure shows a sample construction of the CBA algorithm with a maximum buffer constraint. The dotted lines represent the runs within the bandwidth plan.

To recap, the CBA algorithm then 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 2 : *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 with (1) the smallest number of bandwidth increases possible, (2) the minimum peak bandwidth requirement, and (3) the largest minimum bandwidth required.*

To algebraically calculate the CBA plan requires the allocation of individual 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 *i*th frame within the run. This can be defined as:

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

- *$MaxBW_i$* be the maximum average bandwidth sustainable from the beginning of the run to the *i*th 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

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

The critical bandwidth for the run is then

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

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 will be 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 will be necessary in the next run. As described earlier, a search is then performed at the end of the run for a starting point that maximizes the point reached by the next run. Note that this search involves a fairly trivial calculation to find the appropriate initial buffer for the next run.

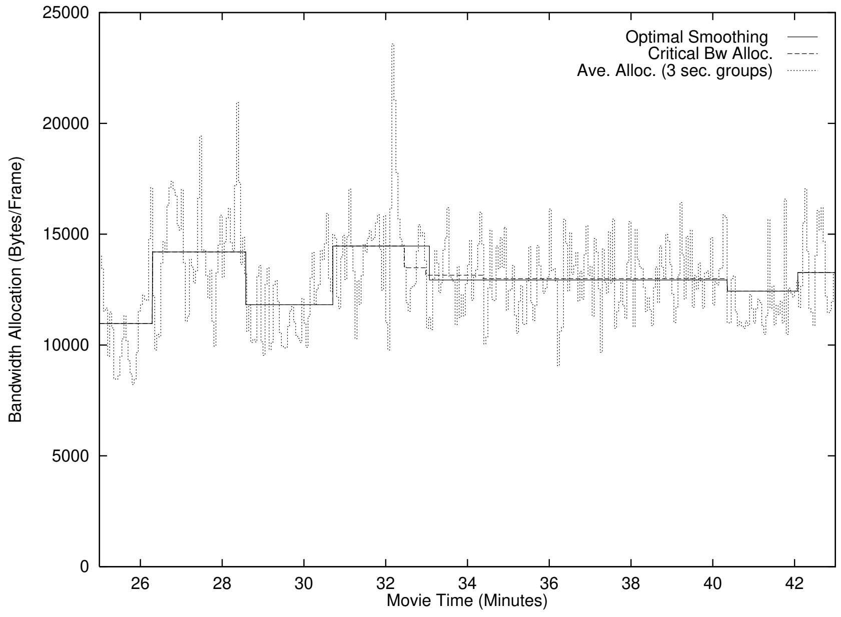


Figure 3.10: Critical Bandwidth Allocation vs. Optimal Bandwidth Allocation. This figure shows the same sample clip from the movie *Speed* as in Figure 3.1. The solid line shows the bandwidth allocation plan using the OBA algorithm and a 2 MB buffer, while the heavier dotted line shows the CBA algorithm. The main difference between the algorithms is that the OBA algorithm combines all the bandwidth decreases into a few request.

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 also minimizes the peak bandwidth requirements and has the largest minimum bandwidth requirement. An example of critical bandwidth allocation smoothing can be found in Figure 3.10.

3.3.3 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 2 while also minimizing the total number of bandwidth changes required. The *optimal band-*

width 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 with a maximum buffer constraint. 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 in order to reduce the number of decreases in bandwidth required from the network. 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 motivate and describe an optimal bandwidth allocation strategy. 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 fall 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 Figure 3.7 and Figure 3.8). For a run which requires a bandwidth decrease in the next run, a search on the frontier of the current run results in four other possible outcomes. These representative outcomes, which are essentially mirror images of the 4 bandwidth increase cases, are shown in Figure 3.11 and Figure 3.12. Thus, 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 per-

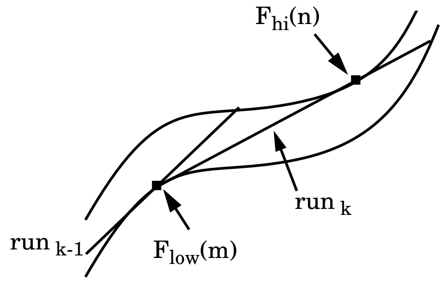


Figure 3.11: Bandwidth Search Example. This figure shows the two representative cases that may result for run *k* in which a bandwidth increase will be required in run *k+1*.

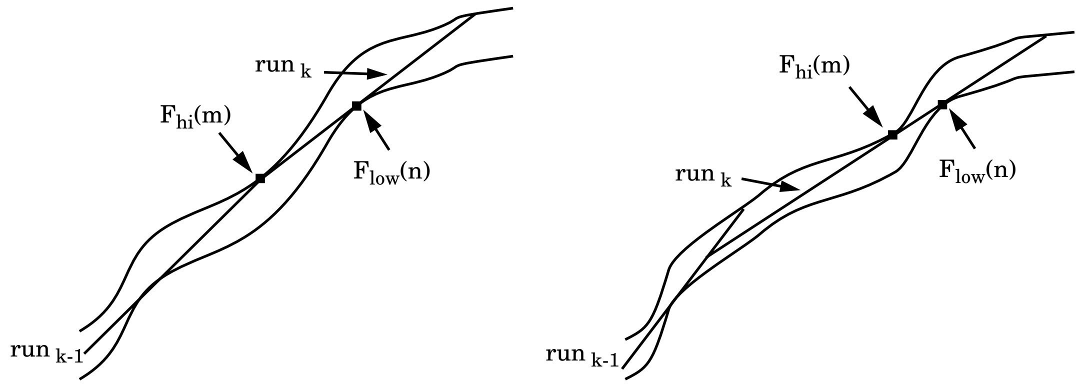


Figure 3.12: Bandwidth Search Example. This figure shows the two representative cases that arise when the search for run *k* results in another bandwidth decrease required in run *k+1*. In (a), $F_{hi}(m)$ lies on the line created by run_{k-1}. In (b), $F_{hi}(m)$ does not lie on run *k-1*.

formed 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 more properties that are parallel to Property 1 and Property 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 $m < n$.*

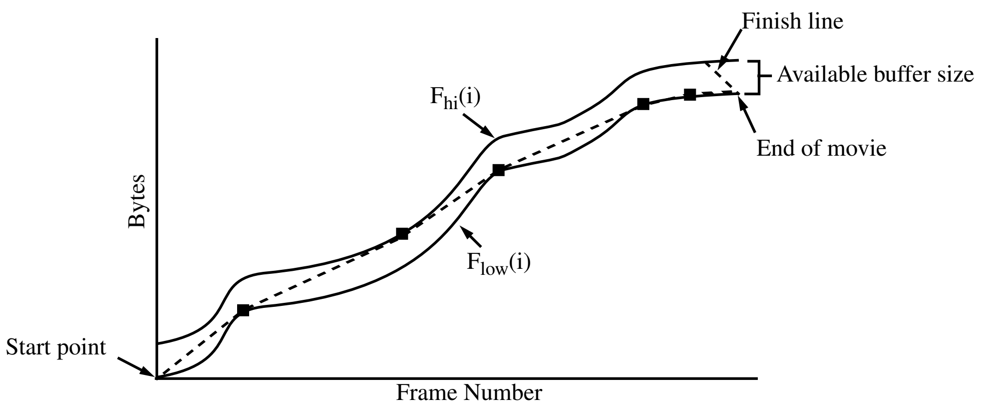


Figure 3.13: Critical Bandwidth Allocation with Maximum Buffer Example. This figure shows a sample construction of the CBA algorithm with a maximum buffer constraint. The dotted lines represent the runs within the bandwidth plan.

This property essentially says that the search for run *k* as shown in Figure 3.11, 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 Figure 3.12. A sample construction is shown in Figure 3.14. Using this “greedy” approach in the allocation of each run within the OBA plan results in the following theorem, which we prove in the appendix:

Theorem 3 : *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 optimal critical*

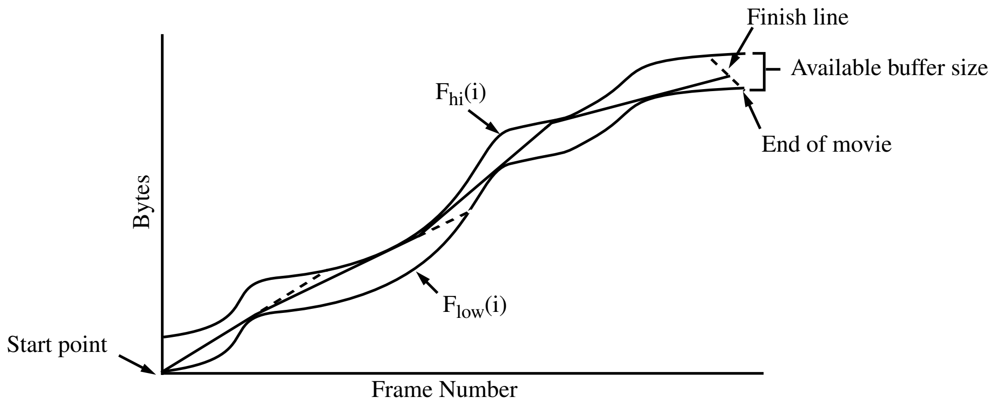


Figure 3.14: Optimal Bandwidth Allocation Construction Example. This figure shows a sample construction of the optimal bandwidth allocation algorithm. Note, this plan includes only 4 runs while the plan in Figure 3.9 required 6 runs. The heavy solid lines show $F_{hi}(i)$ and $F_{low}(i)$, while the 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.

bandwidth algorithm results in (1) smallest peak bandwidth, (2) the largest minimum bandwidth, and (3) the fewest possible bandwidth changes.

To construct the optimal bandwidth allocation plan, we use the same algorithm for finding the critical bandwidth and critical point for a run as defined in Section 3.3.2. 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 far the bandwidth should be held past the end of the previous run's critical point (See Figure 3.11 and Figure 3.12). 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 extends as far into time 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* with its critical point determining the end of the run.

For each subsequent run, we 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 Figure 3.10.

3.4 Evaluation of Algorithms

From the point of view of network management, load estimation, the necessary bandwidth resources, 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, a network manager can handle the partitioning of its bandwidth in several ways. The network manager can reserve the bandwidth at the expected peak bandwidth requirement for the channel, in which case, minimizing the peak bandwidth is important. The network manager can also let the clients allocate bandwidth as they go along, if it does not have provisions for in-advance bandwidth reservations. In this case, both the number and magnitude of bandwidth increases can be important because the bandwidth increase requests may be denied. As a result, three measures that influence this performance

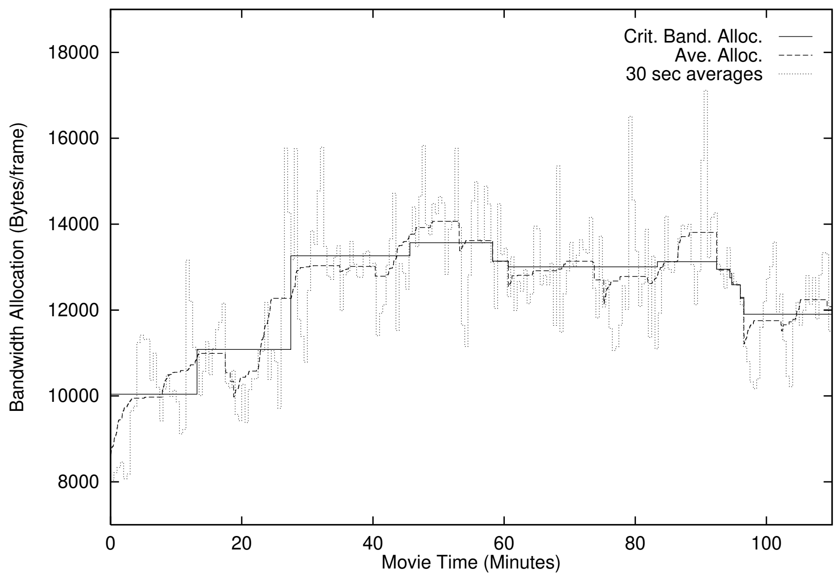


Figure 3.15: Comparison of Bandwidth Allocation Algorithms. This graph shows the results of smoothing using the average allocation based and CBA algorithm with a 10 MByte buffer for smoothing. The data above was obtained by using the movie *Speed*. Because the sliding window smoothing algorithm has two possible parameters, chunk size and window size, we fixed the chunk size at 90 frames and expanded the window to take advantage of the buffer (in this case 81 chunk window size).

are the frequency of requests for increased bandwidth, the size of these increases, 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. We break the evaluation of the algorithms presented in this chapter into two parts. We first compare and contrast the window-based smoothing algorithms with the CBA algorithms. We then move into a more in-depth comparison of the various critical bandwidth allocation based techniques.

3.4.1 Averaging Techniques Versus CBA

To test the effectiveness of the window-based algorithms and the CBA algorithm, each algorithm was run on the digitized movie *Speed*. Figure 3.15 shows the results of smoothing using a 10 MByte buffer. This graph shows that the critical

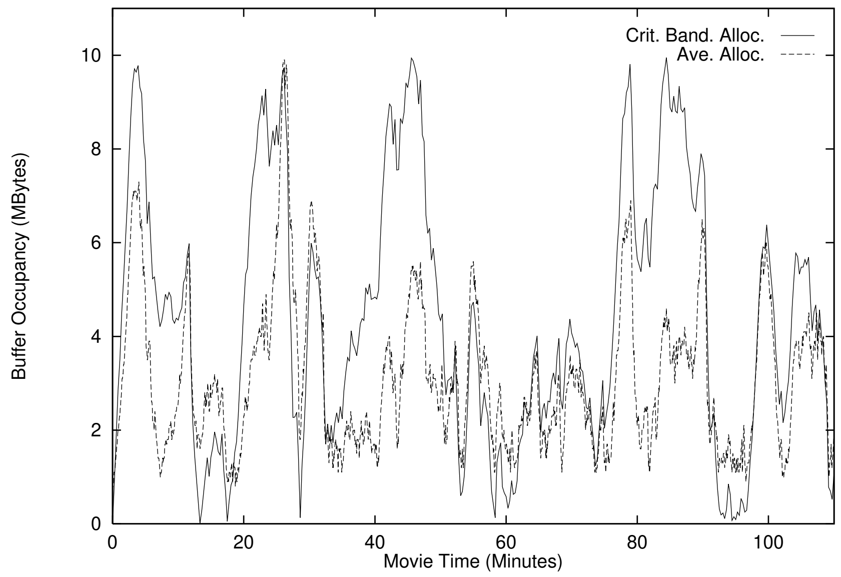


Figure 3.16: Comparison of Bandwidth Allocation Algorithms. This graph shows the utilization of a 10 Mbyte buffer for the sliding window smoothing and the critical bandwidth algorithms. The size of the sliding window is a constant, set in such a way as to make use of the full buffer in the worst case. However, the algorithm is only able to use the buffer in the first part of the video clip. A larger window in the second half of the segment would have taken better advantage of the buffer. The critical bandwidth algorithm adjusts its bandwidth to take full advantage of the buffer whenever necessary.

bandwidth approach produces smooth allocation with infrequent adjustments in bandwidth, requiring only 4 negotiations for increased bandwidth. It is interesting to note that the critical bandwidth algorithm require increases only after large valleys of small frame sizes. While the sliding window smoothing technique reduced the peak bandwidth requirements, it still requires the network to incrementally give bandwidth to it as a burst of large frame sizes approaches. Figure 3.16 shows the buffer utilization for the sliding window smoothing algorithm and the critical bandwidth allocation algorithm. As shown by the figure, the critical bandwidth algorithm results in the buffer reaching near capacity several times, while the smoothing window sliding algorithm is set in such a way as to make use of the full buffer in the worst case. In fact, the critical bandwidth allocation algorithm results in a full buffer at least once for each increase in bandwidth that occurs within the video. The sliding window

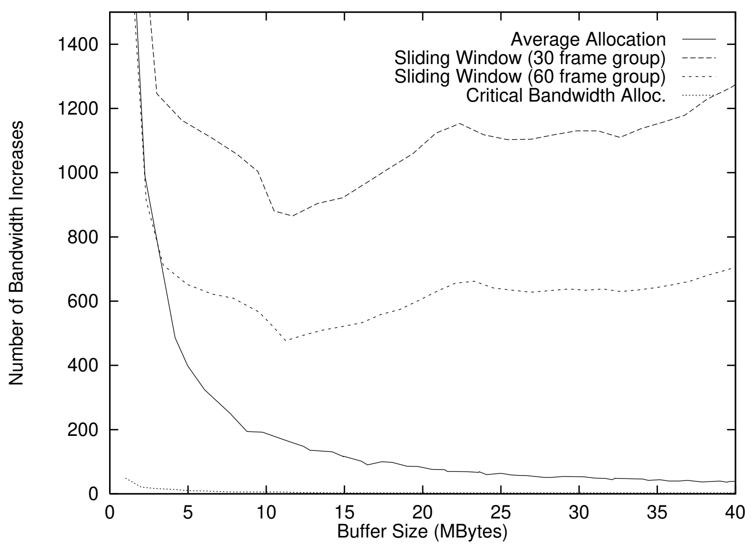


Figure 3.17: Bandwidth Increase Requests. The total number of bandwidth increases represent the number (not magnitude) of bandwidth increase requests for the 110-minute length movie *Speed*. Each algorithm was applied to the movie segment with different buffer capacities. The average allocation algorithm varied the chunk size in order to make use of the extra buffer space. The sliding window algorithms chunk sizes were fixed at 30 and 60 frame chunks and then the window size was varied to obtain the amount of smoothing given some buffer space. The critical bandwidth algorithms used the buffer size to determine the bandwidth requests.

smoothing algorithm, on the other hand, would benefit from a larger window in the second half of the movie.

To avoid overtaxing network resources, bandwidth allocations have to be tracked and approved by some network manager. Increases in bandwidth are particularly important because they may be denied and some adjustment will have to be made, such as a change in quality of service, the establishment of an alternative route, or perhaps the allocation of additional prefetch buffer space.

Increases in Bandwidth

In Figure 3.17, the total number of bandwidth increases is shown as a function of buffer size. The sliding window algorithms makes many small bandwidth increase requests. This stems from the fact that when a large burst of frames first moves into the sliding window, the window size limits the amount of smoothing that can be done. Consequently, as the burst continues to move through the window it is smoothed across many groups, requiring bandwidth increases every 1 to 2 seconds. The average allocation algorithm uses larger chunks for an equivalent buffer size and therefore calls for fewer increases in bandwidth than does the sliding window algorithm. Because the smoothing is still limited by the size of the period used, the average allocation algorithm requires substantially more increases in bandwidth. The critical bandwidth algorithm makes a low number of increase requests as expected. As an example, with a 12 MByte buffer, the critical bandwidth algorithm requires only 3 (including the initial bandwidth request) increases in the bandwidth allocation. As a result, increases occur on average of every 36 minutes. With a 14 Mbyte buffer only 2 increases are needed (one in the beginning of the movie) and one in the middle.

The size of a requested increase can be important, since the larger the increase, the less likely the network manager will have sufficient bandwidth available. Critical bandwidth allocation makes adjustments that are larger on average than does sliding window smoothing. However, this is often because sliding window smoothing will break a large change in bandwidth into a number of requests. It is, therefore, useful to examine the total size of increases requested from the different algorithms.

As Figure 3.18 shows, the average allocation algorithm makes the worst use of buffer space, requiring large changes in the actual bandwidth allocation requirements. The primary limitation of this algorithm lies in its inability to effectively prefetch data. The only way for the algorithm to obtain a lower increase cost was to increase the size of the chunks. By using the sliding window smoother, the size of the chunk can remain relatively small, reducing the amount of buffer space used to store the next chunk. By using this extra buffer space to prefetch bursts of large frames, the sliding-window algorithms are more effective in smoothing bandwidth curves

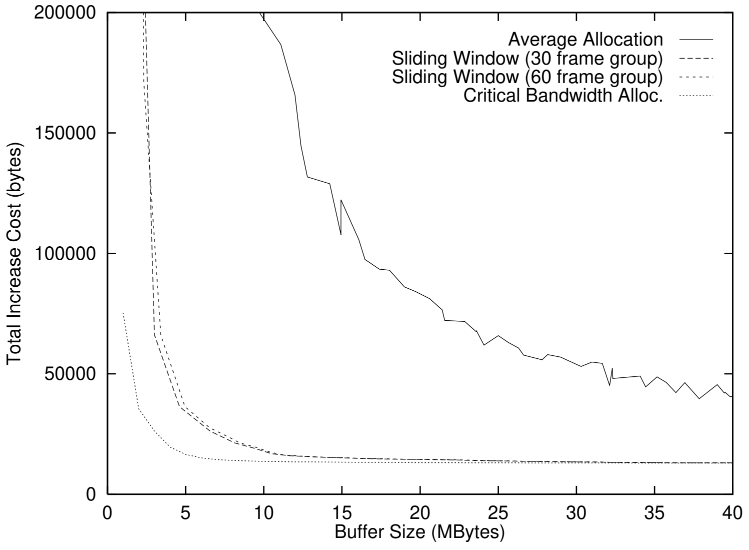


Figure 3.18: Bandwidth Increase Costs. The total increase cost of each algorithm is the summation of all bandwidth *increase* requests for the movie *Speed*. While both sliding window smoothing and critical bandwidth allocation lead to a small cost for large buffers, sliding window allocation spreads its increases over a much greater number of requests.

than the average allocation algorithm. The critical bandwidth algorithm has the smallest cost because it minimizes the total range of bandwidth requirements as well as the number of bandwidth increases required within that range.

Decreases in Bandwidth

A decrease in required bandwidth may require an interaction with the network manager, but will not be disallowed due to contention for resources. Some approaches to access control assume that bandwidths will be constant, but smoothing shows that within lengthy segments it may be monotonically decreasing. Thus, there may be benefits in developing admission control policies that allow reserved bandwidth to be released without terminating a channel. The critical bandwidth algorithm does a good job of identifying blocks of bandwidth that can be released in a low number of requests (see Figure 3.19).

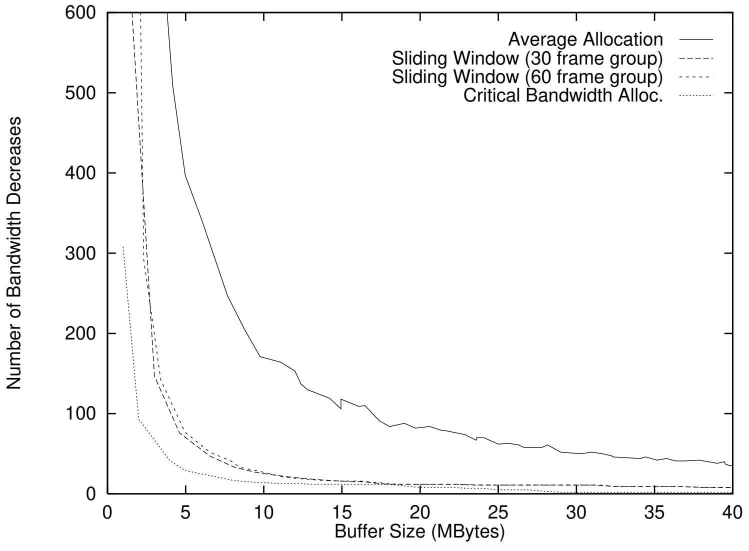


Figure 3.19: Bandwidth Decrease Requests. The total number of bandwidth decreases represent the number (not magnitude) of bandwidth decrease requests for the movie *Speed*. The critical bandwidth algorithm makes more decrease requests than increase requests. The sliding window smoothing algorithm on the other hand, makes many fewer requests to decrease bandwidth than to increase, since increases are frequently spread across several requests.

Peak Bandwidth Requirement

For systems that allocate bandwidth based on the peak bandwidth requirements, minimizing the peak bandwidth used as well as the range of bandwidths used can be helpful with admission control. As shown in Figure 3.20, the average allocation algorithm does not always lower the peak bandwidth requirement given more buffer for smoothing. The primary limitation is that the averaging algorithm can only use larger and larger period sizes to accomplish more smoothing, hence, the boundaries for the calculation of averages continue to change as more buffer is added. Fixing the period size and then increasing the window size has the effect of smoothing the bandwidth requirements as more buffer is added. As more buffer is added, the additional buffer is used strictly for prefetching bursts of data in the future. The critical bandwidth algorithm finds the minimum peak and maximum valley bandwidths. For buffers greater than 17 MBytes, the first run determined the maximum band-

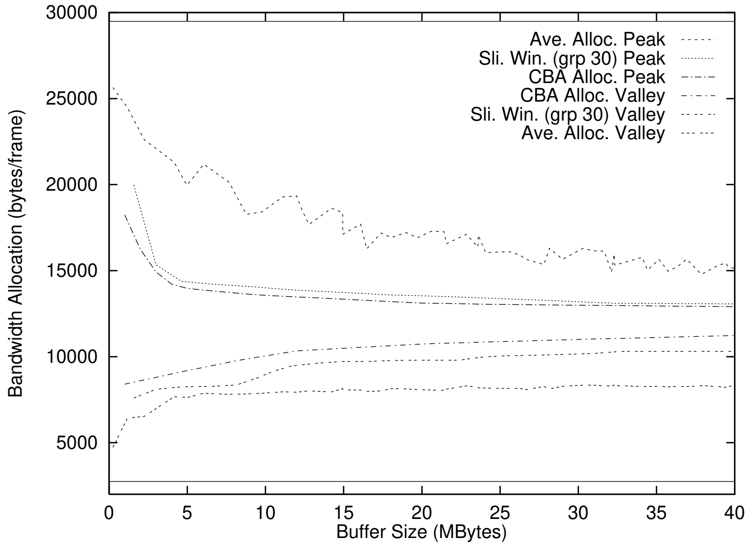


Figure 3.20: Bandwidth Requirement Ranges. This graph shows the bandwidth requirements ranges for the movie *Speed*. The solid horizontal lines are the minimum and maximum frame sizes for the movie.

width requirement for the movie. If prefetching is allowed (and the delay is tolerable), the peak bandwidth requirement for buffers greater than 17 MBytes can be reduced further.

3.4.2 Non-Window Based Smoothing Algorithms

In this section, we compare and contrast the various critical bandwidth based algorithms. We have graphed the bandwidth allocation plans for the OBA algorithm using both a 10 and 30 MByte buffer for all the movies. These graphs, located at the end of this chapter in Figure 3.33 to Figure 3.52, show how the OBA algorithm smooths the bandwidth requirements for the delivery of video.

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

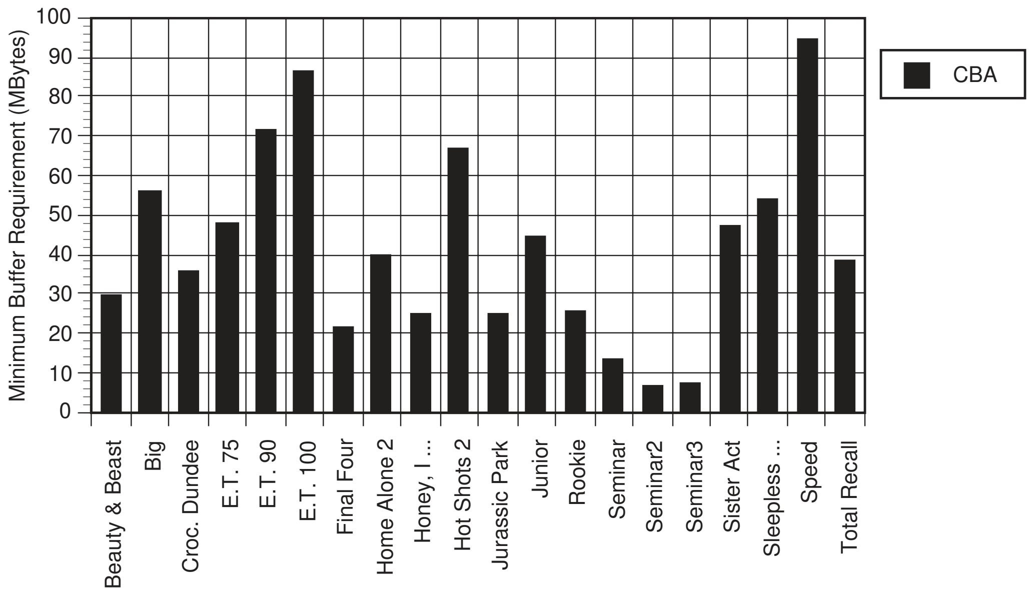


Figure 3.21: Critical Bandwidth Allocation Minimum Buffer Requirement. This figure shows the maximum buffer requirements for the movies encoded using the CBA algorithm with no buffer limitation.

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 Figure 3.21, the amount of buffer required to play back the sample videos varied quite a bit. The total amount of buffering depends primarily on two factors, the average size of the frames and the long-term burstiness of the video. If the movie had a sustained area of smaller frames followed by a sustained area of larger frames, the amount of buffering tended to be much higher. As examples, the seminar videos required much smaller buffer sizes because both the size and variation of frames sizes were small in these videos. The *E.T.* videos required proportionately larger buffers as the quality was increased. Because these videos exhibited 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* required a larger buffer

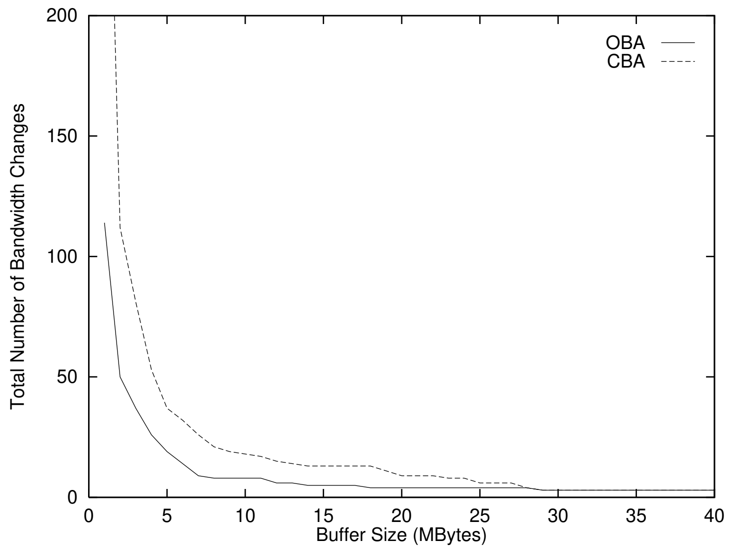


Figure 3.22: Bandwidth Change Requests. The graphs above show the total number of required bandwidth allocation change requests for the movie *Speed*. The OBA and CBA algorithms were run on the entire video clip for varying buffer sizes in one MByte increments.

than the *E.T.* (Quality 100) video even though both the variance in frame sizes and the average frame sizes were smaller in the *Speed* video. Thus, the buffer size is primarily due to the long term burstiness, and to a lesser degree, the average frame sizes within the videos.

Bandwidth Changes

As Figure 3.22 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 MByte smoothing buffer, and the OBA algorithm results in 21 bandwidth changes over the 110 minute 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 minutes, using only 5 MBytes of buffer. After the initial start of the movie, the *Speed* movie using the OBA algorithm had a minimum run length of approximately 1 minute and 45 seconds and a maximum run length of approximately 14 minutes. By

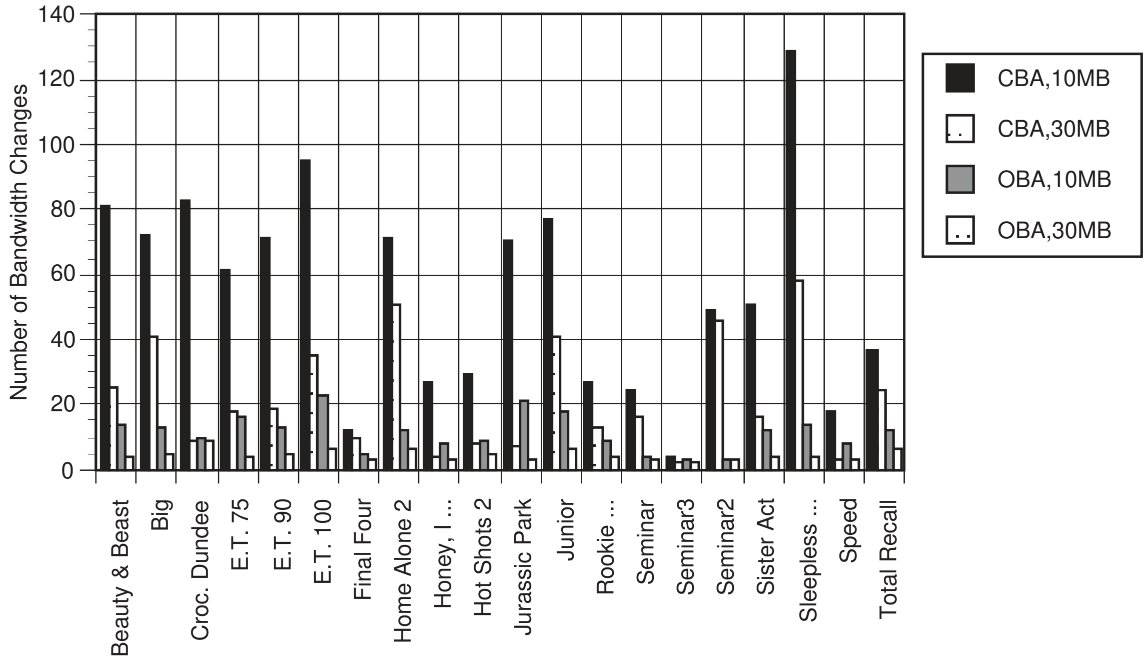


Figure 3.23: Bandwidth Changes for All Movies. This graph show the number of bandwidth changes required for all of the sample movies using a 10 and 30 MB buffer and the OBA and CBA Algorithms.

using a 10 MByte buffer for buffering, the shortest and longest run lengths grow to 13 minutes and 25.5 minutes, respectively. As a result, a moderate amount of buffering can reduce the number of interactions required from the network to the order of tens of minutes.

As shown in Figure 3.23, the total number of bandwidth changes required is relatively small even for a 10 MByte buffer with loosely encoded video. The OBA algorithm results in a large difference in required bandwidth changes. For a 10 MByte 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 MByte smoothing buffer and 63% fewer bandwidth changes at 30 MBytes. The distinction between increases and decreases in the bandwidth allocation plan can be useful because the requests for

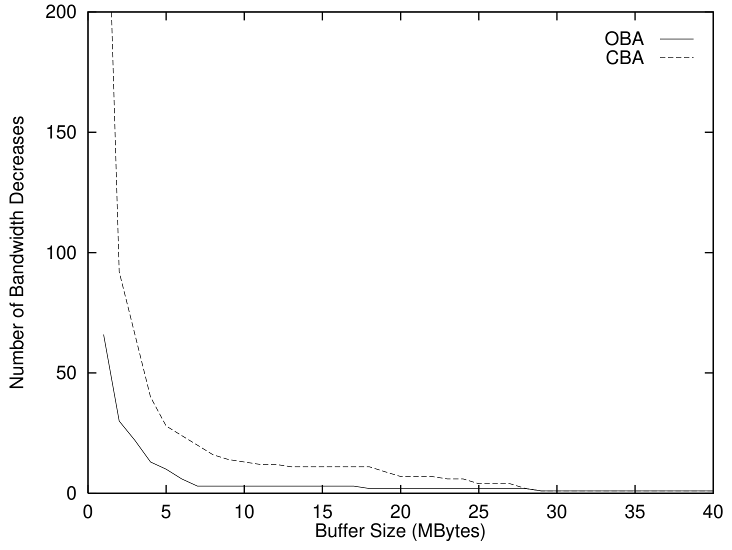


Figure 3.24: Bandwidth Decrease Requests. The graphs above show the total number of bandwidth allocation decrease requests for the movie *Speed*. The CBA and OBA algorithms were applied to the movie segment with different buffer capacities.

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.

Bandwidth Decrease Requests

As shown in Figure 3.24, the total number of bandwidth decreases for the *Speed* video are similar to the total number of bandwidth change graph (Figure 3.22). 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 critical bandwidth allocation algorithm allocates each run at the minimum bandwidth requirement to avoid underflow, while the optimal bandwidth

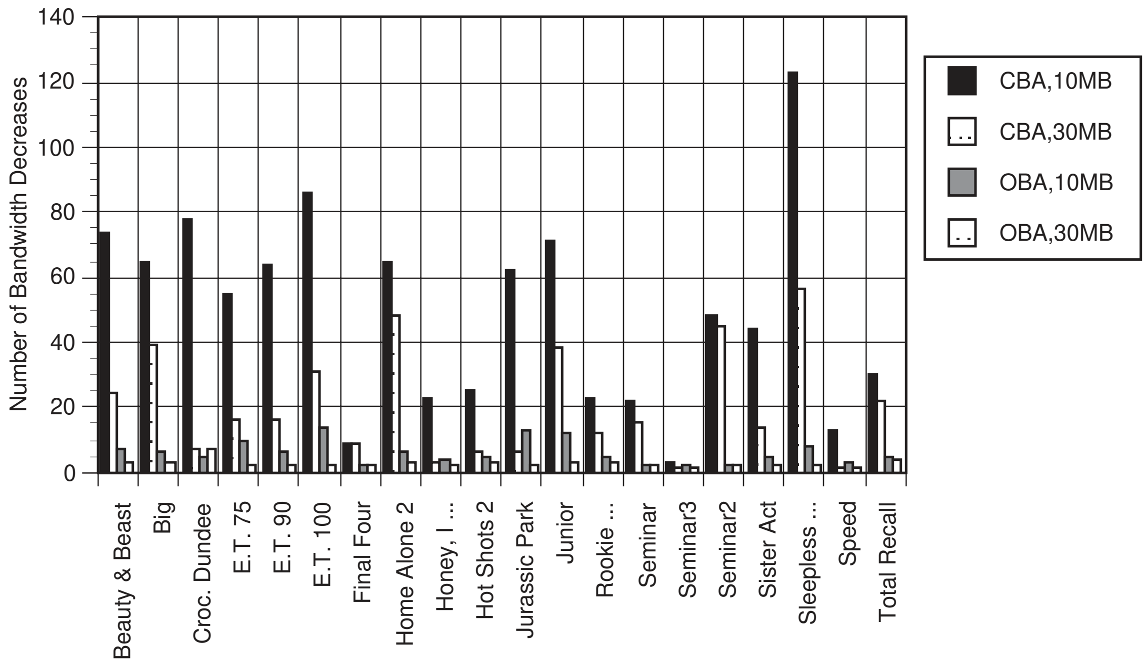


Figure 3.25: Bandwidth Decreases for All Movies. This graph show the number of bandwidth decreases required by the OBA and CBA algorithms for 10 and 30 MB buffers for all the sample movies.

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.

Bandwidth Increase Requests

For bandwidth increases, using the CBA and OBA algorithms result in the same number of increases across all buffer size constraints for each movie, therefore, verifying Theorem 2. As shown in Figure 3.26, the number of increases required for the movie *Speed* drops to 5 increases for buffers greater than 8 MBytes and drops to only 2 increases for buffers greater than 14 MBytes. As a result, interactions with the network will only be required, on average, every 21 and 55 minutes for an 8 and 14 Mbyte smoothing buffer, respectively. As shown in Figure 3.27, all the other movies exhibited similar behavior to the *Speed* video. It is interesting to note that all but 4 of the videos require only one increase in the bandwidth requirement after the initial bandwidth allocation.

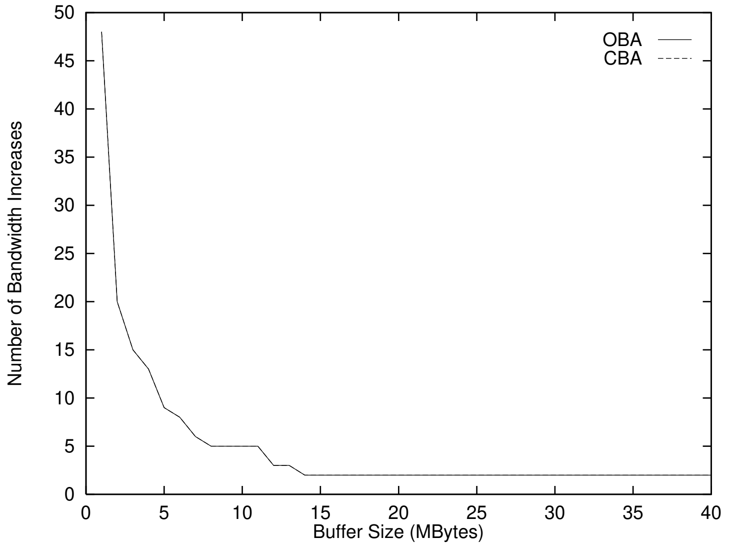


Figure 3.26: Bandwidth Increase Requests. The graph above shows the total number of bandwidth allocation increase requests for the movie *Speed*. The CBA and OBA algorithms were applied to the video with different buffer capacities, all resulting in the same number of bandwidth increases as shown above.

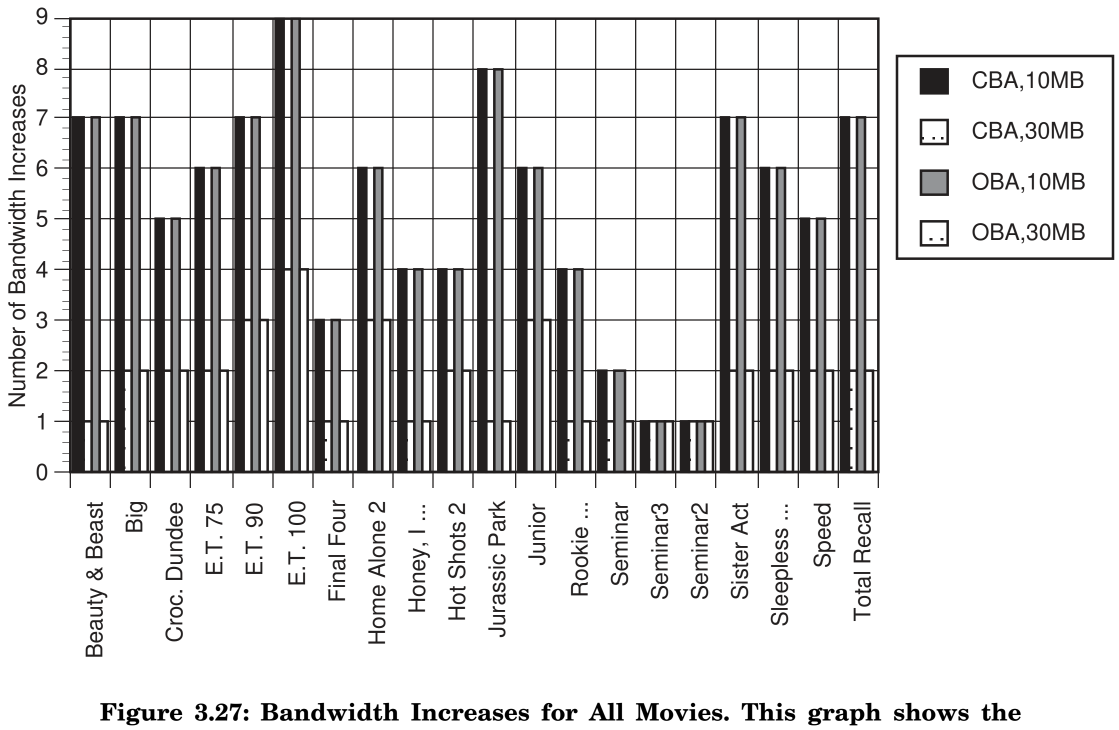


Figure 3.27: Bandwidth Increases for All Movies. This graph shows the number of bandwidth increases required using the OBA and CBA algorithms with a 10MB and 30 MB buffer.

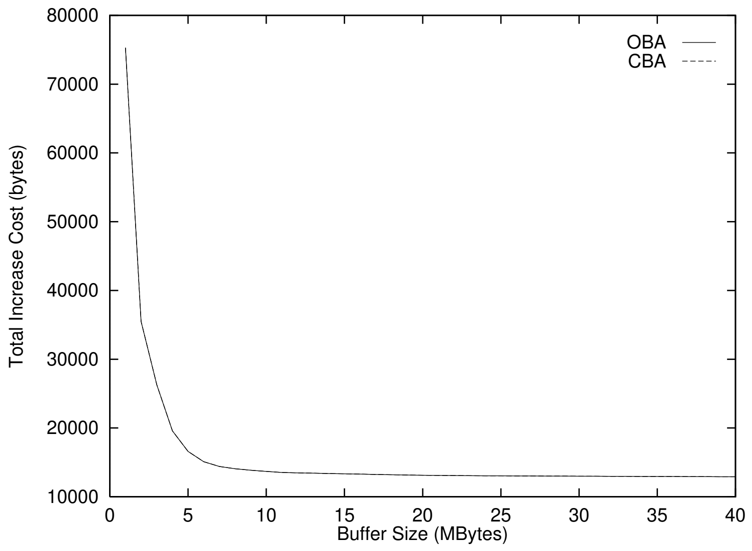


Figure 3.28: Bandwidth Increase Costs. The graph above shows the total sum (magnitude) of all the bandwidth allocation increase requests for the movie *Speed*. The CBA and OBA algorithms were applied with different buffer capacities. The optimal bandwidth allocation algorithm and the critical bandwidth allocation algorithm result in the same totals.

Bandwidth Increase Magnitude

Requests for increases in bandwidth allocation require interaction with the network manager for more resources. The frequency and magnitude of these increases will determine the network's ability to adapt to changing network load. Comparison of increases in bandwidth magnitudes are only relevant when the total number of increase requests is the same. As an example, if one stream requests 1000 bytes/frame more bandwidth and another asks for 100 bytes/frame on ten separate occasions, no comparison can be made. Because the critical bandwidth based algorithms have the same number of increases for a given movie and both have the minimum-peak and highest-minimum bandwidths, their results are the same. As shown in Figure 3.28, the critical bandwidth algorithm with prefetching and the optimal bandwidth allocation algorithm have the same total magnitude of bandwidth increases.

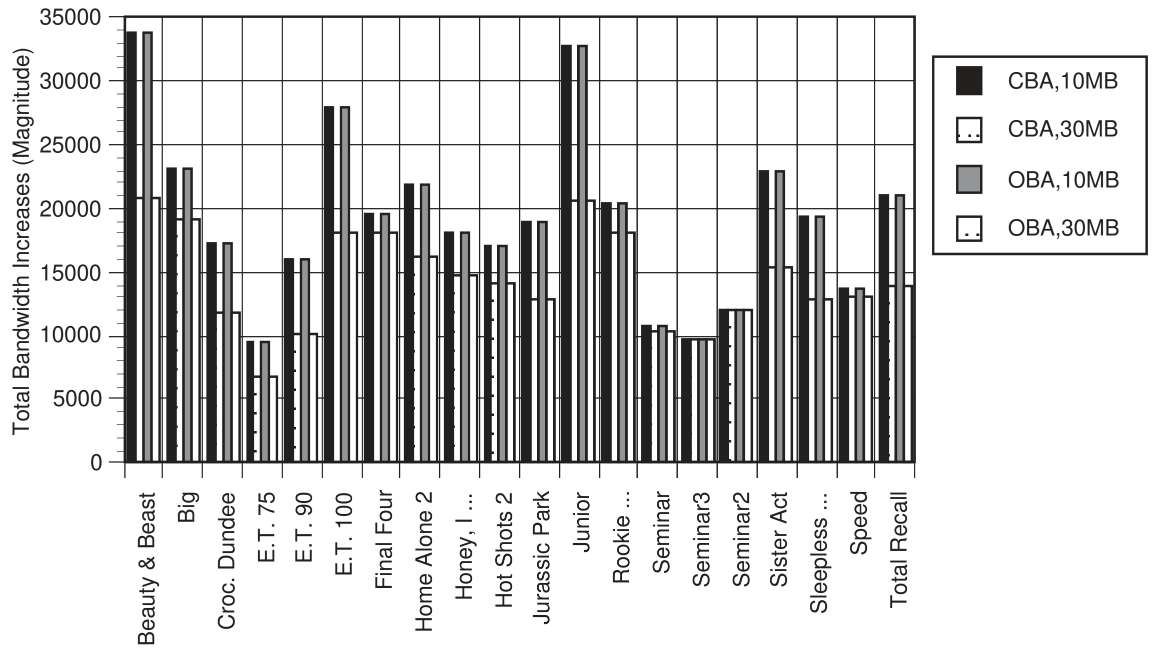


Figure 3.29: Bandwidth Increase Requests for All Movies. This figure shows the total magnitude of all the bandwidth increase requests for the sample movies using the OBA and CBA algorithms.

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 are used for the entire movie and then calculate the utilization of the bandwidth reserved. The peak bandwidth utilization measurements are shown in Figure 3.30. From this graph, we see that some movies (particularly the ones with low utilization) do not improve their utilization of bandwidth between 10MBytes and 30 MBytes 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

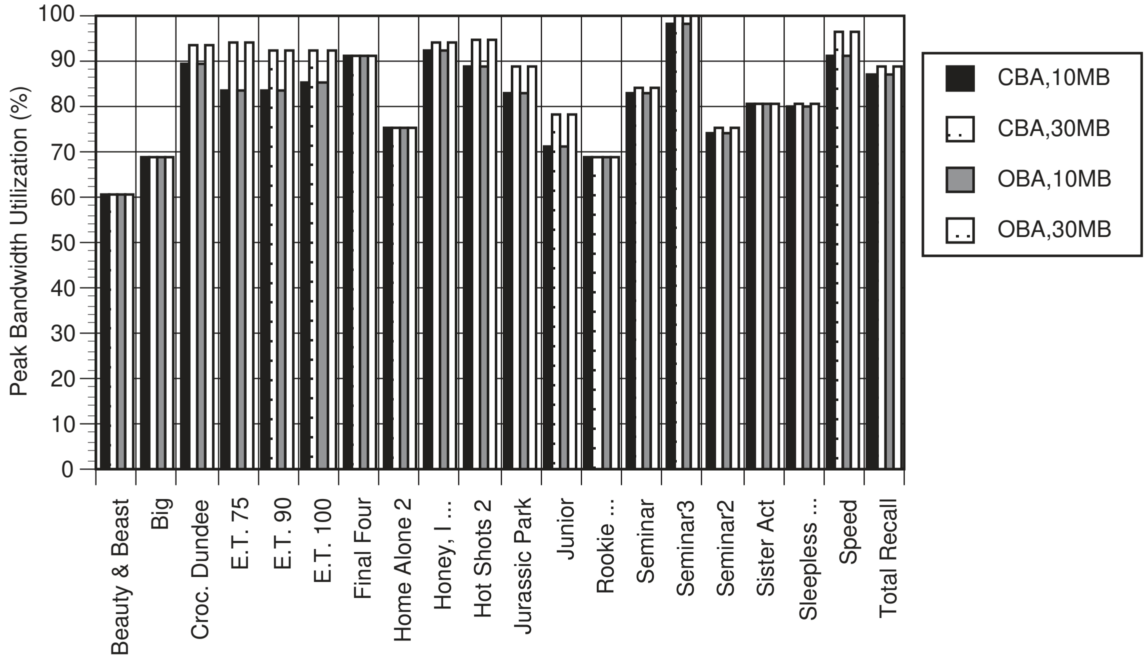


Figure 3.30: Peak Bandwidth Utilization for All Movies. This figure shows the utilization of resources allocated using the peak bandwidth requirement and the OBA and CBA algorithms (with no initial prefetching at the start of the movie).

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 Figure 3.31. As Figure 3.32 shows, the tumbling bandwidth utilizations are much higher than the peak bandwidth utilization measurements, mostly due to the initially high bandwidth requirement of the optimal bandwidth allocation 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*, it is interesting

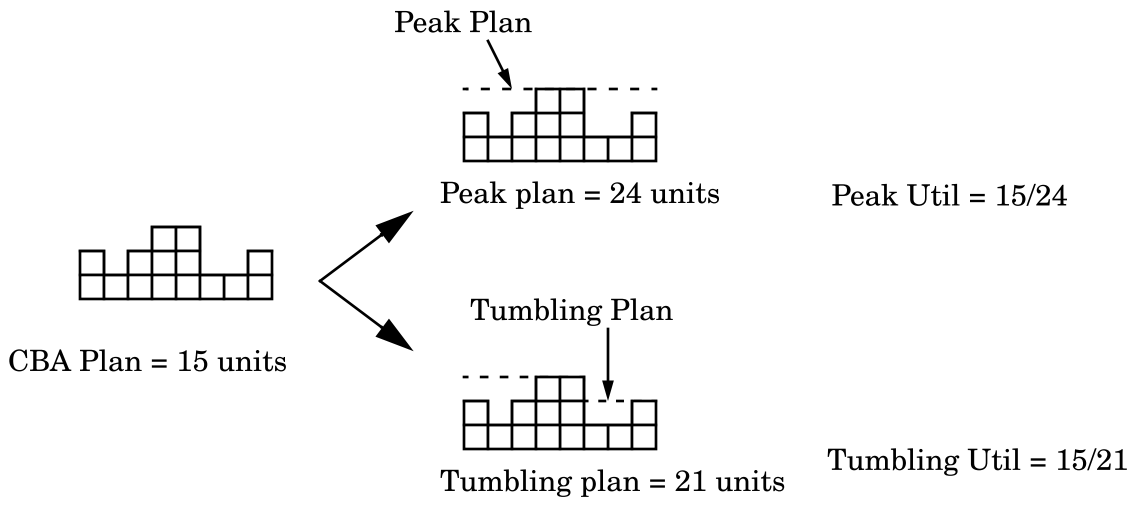


Figure 3.31: Peak Utilization vs. Tumbling Utilization. This figure shows the difference between the peak and tumbling utilization calculations. The CBA plan (heavy solid line) requires 15 units of bandwidth (represented by squares). The tumbling utilization plan always results in a utilization greater than or equal to the peak utilization.

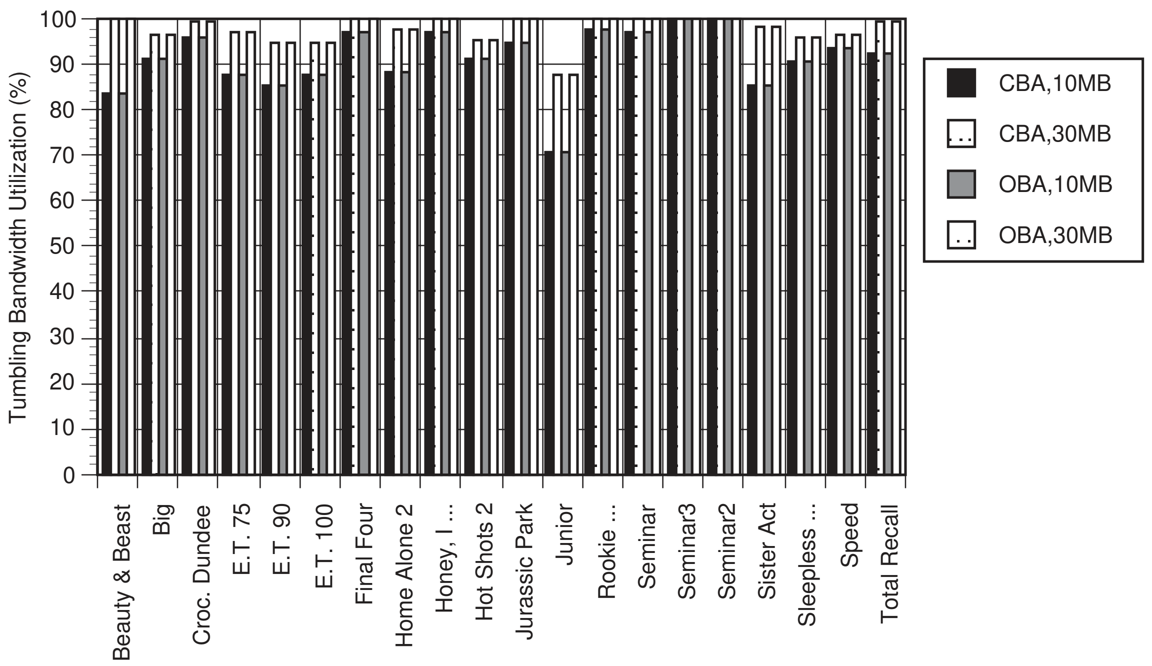


Figure 3.32: Tumbling Bandwidth Utilization for All Movies. This figure shows the tumbling utilization for the OBA and CBA algorithms on the sample movies. The tumbling utilization is the same as the peak bandwidth requirement utilization except once the peak bandwidth requirement is passed, a channel may reduce its peak bandwidth requirement once the peak has passed.

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

3.5 Summary of Bandwidth Smoothing Algorithms

In this chapter, we have examined several bandwidth smoothing algorithms that can be used for the delivery of compressed video streams. Window-based smoothing algorithms are useful in smoothing bandwidth requirements where the delay between the receipt and playback of a frame must adhere to some maximum delay. Thus, window-based smoothing algorithms are particularly suited for live-video applications.

The non-window-based *critical bandwidth allocation* algorithms have been shown to be useful for the delivery of prerecorded compressed video, where the *a priori* knowledge of the video is obtainable before the bandwidth reservations are made. The CBA-based algorithms have been shown to minimize both the peak bandwidth requirements as well as the number of bandwidth increases required for the delivery of compressed prerecorded video. In addition, we have shown that the CBA algorithm can be extended to minimize the total number of changes in bandwidth as well.

For stored video applications, the use of critical bandwidth allocation plans in admission control can help in the efficient allocation of network resources. For a system that has smaller clips such as those in an educational interactive setting, using the critical bandwidth allocation approach without a maximum buffer constraint makes admission control easier by having to perform only a single check for admission control. In addition, with smaller clips which last less than an hour, the amount of buffering needed is not as great. For video-on-demand movie systems, the use of the optimal bandwidth allocation algorithm with a maximum buffer constraint can also simplify admission control because the number of bandwidth changes as well as the peak bandwidth requirements are minimized. We will examine the use of the optimal bandwidth allocation algorithm in a general setting in the next chapter.

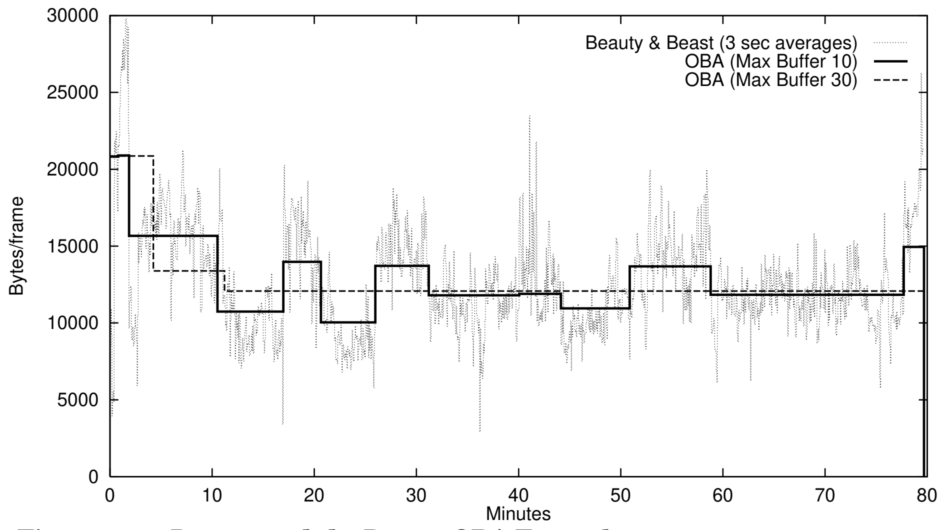


Figure 3.33: ***Beauty and the Beast*** **- OBA Example**

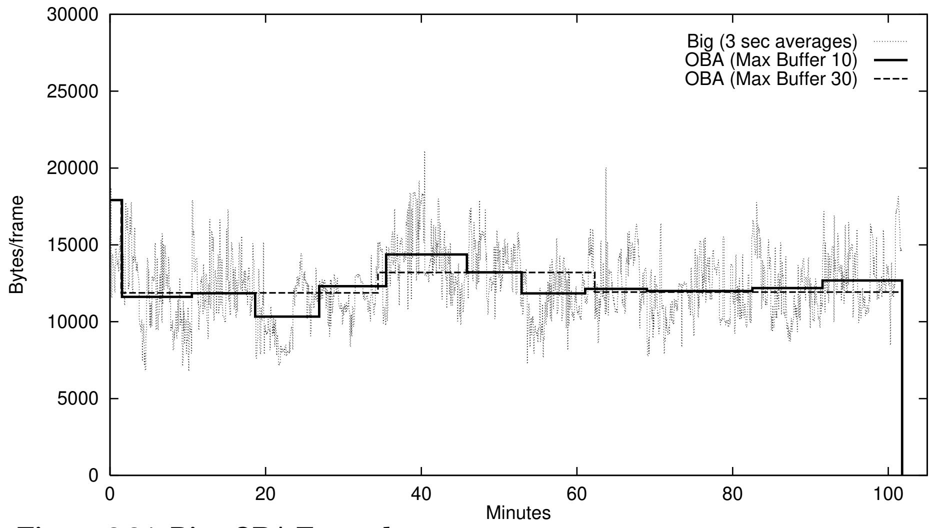


Figure 3.34: ***Big*** **- OBA Example**

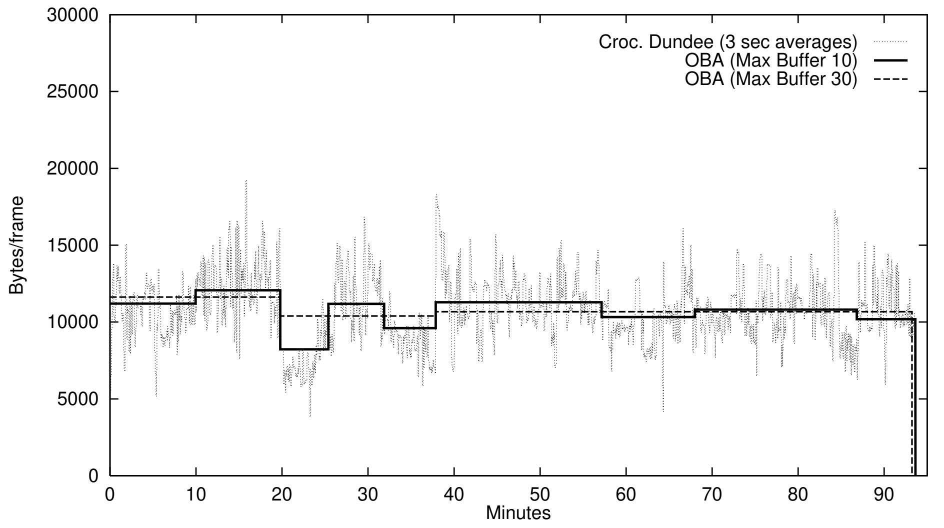


Figure 3.35: ***Crocodile Dundee*** **- OBA Example**

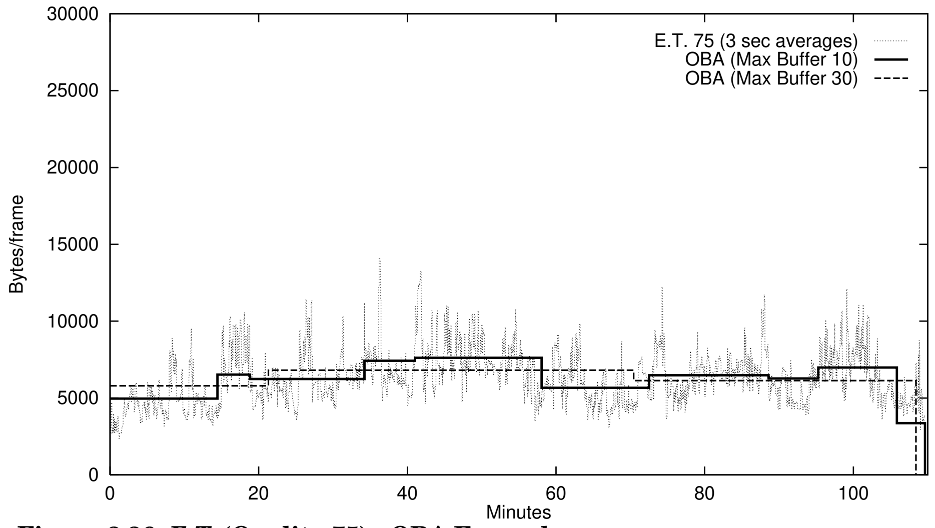


Figure 3.36: *E.T.* (Quality 75) - OBA Example

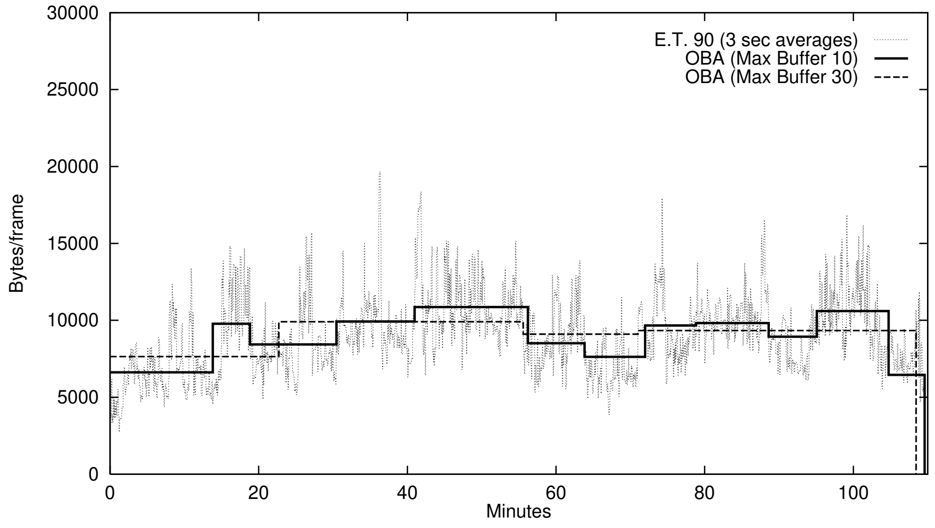


Figure 3.37: *E.T.* (Quality 90) - OBA Example

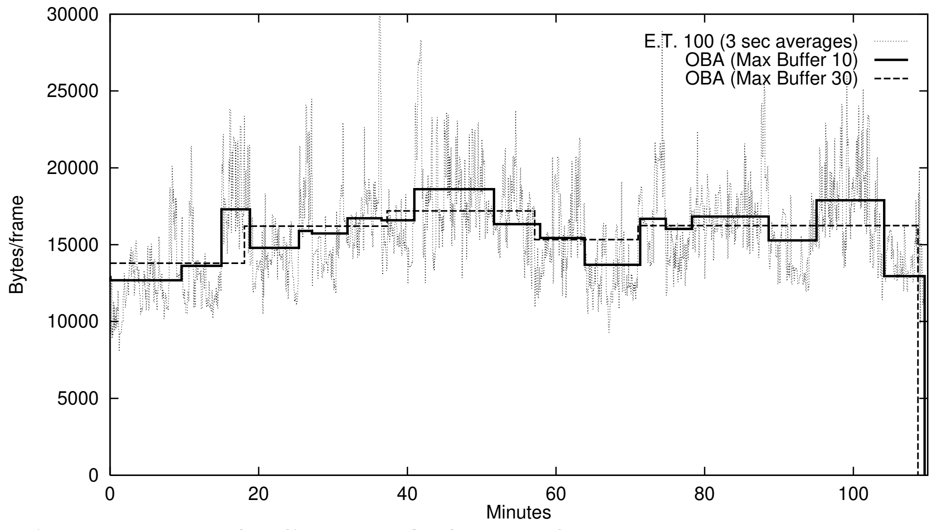


Figure 3.38: *E.T.* (Quality 100) - OBA Example

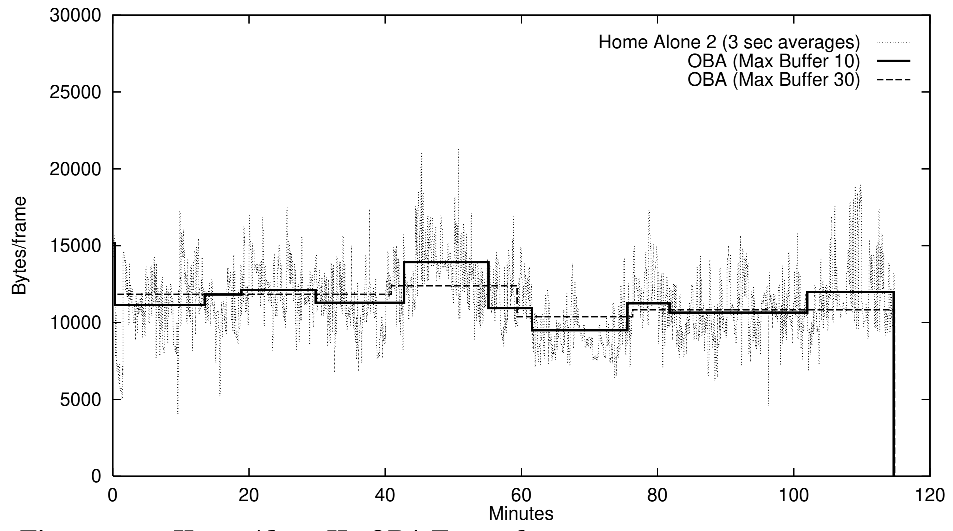


Figure 3.39: ***Home Alone II*** **- OBA Example**

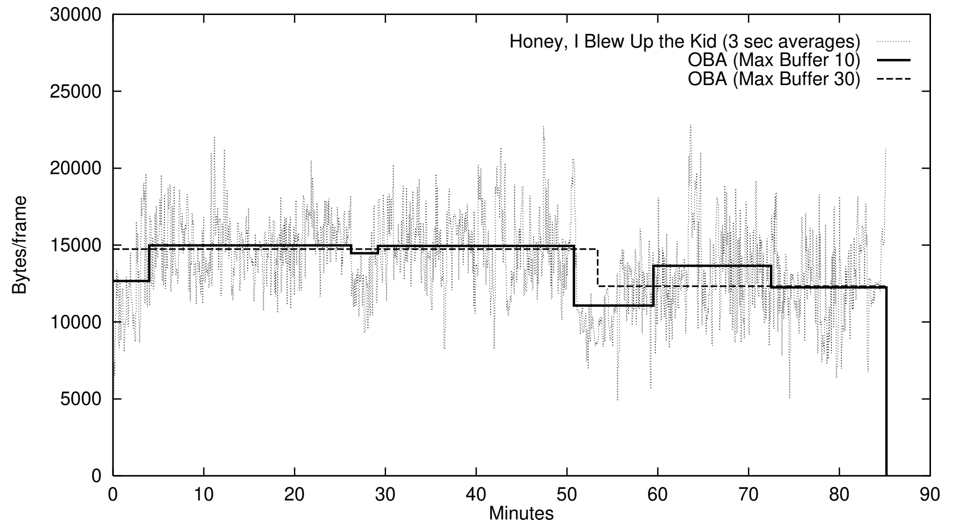


Figure 3.40: ***Honey, I Blew Up the Kid*** **- OBA Example**

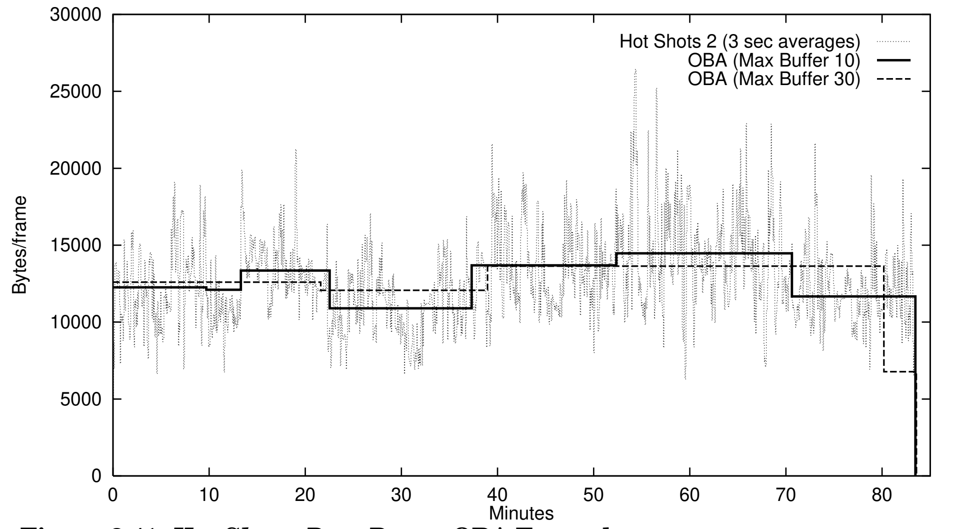


Figure 3.41: ***Hot Shots, Part Deux*** **- OBA Example**

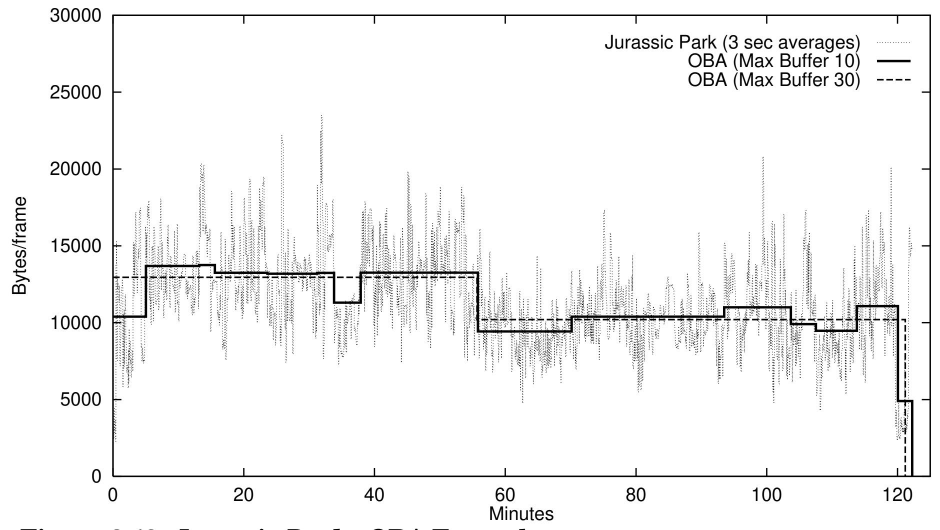


Figure 3.42: ***Jurassic Park*** **- OBA Example**

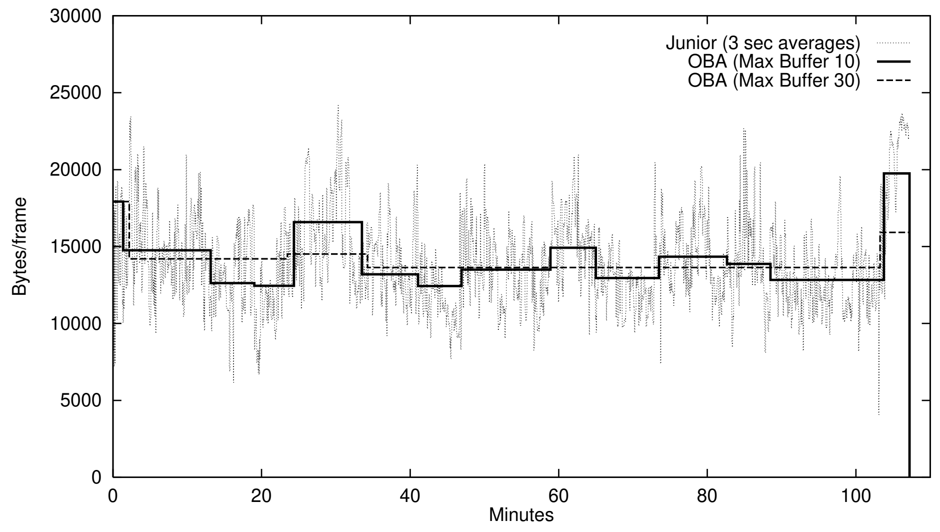


Figure 3.43: ***Junior*** **- OBA Example**

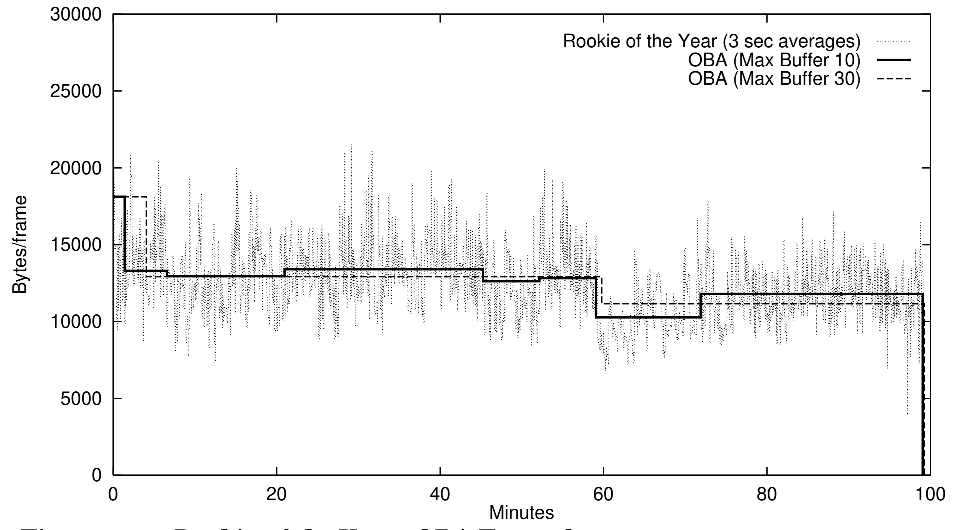


Figure 3.44: ***Rookie of the Year*** **- OBA Example**

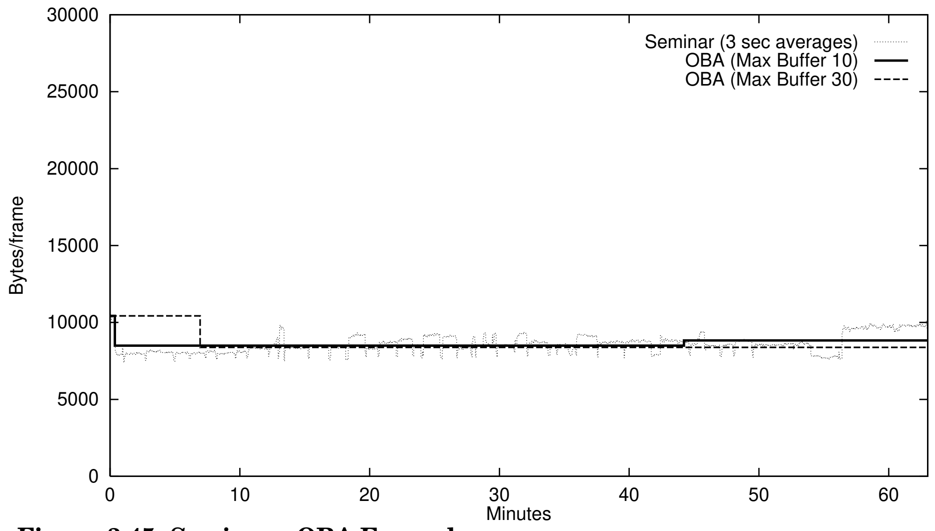


Figure 3.45: ***Seminar*** **- OBA Example**

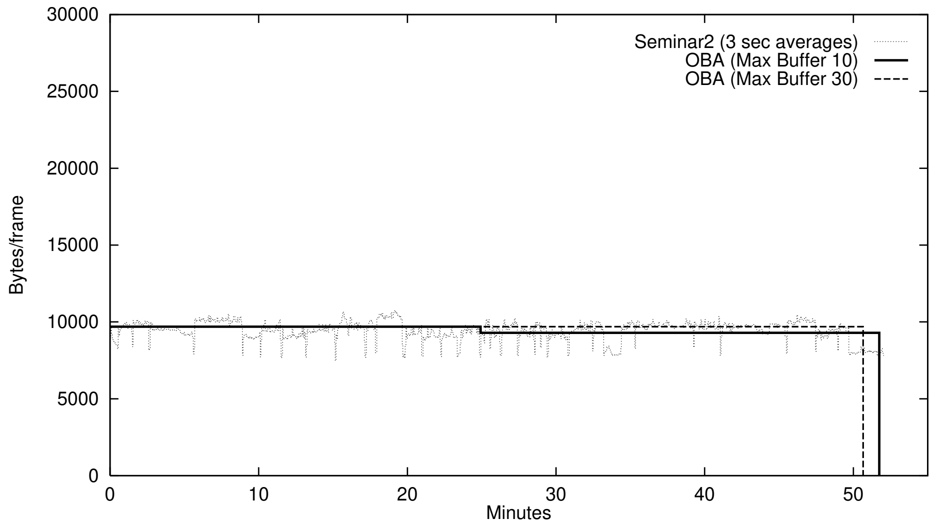


Figure 3.46: ***Seminar2*** **- OBA Example**

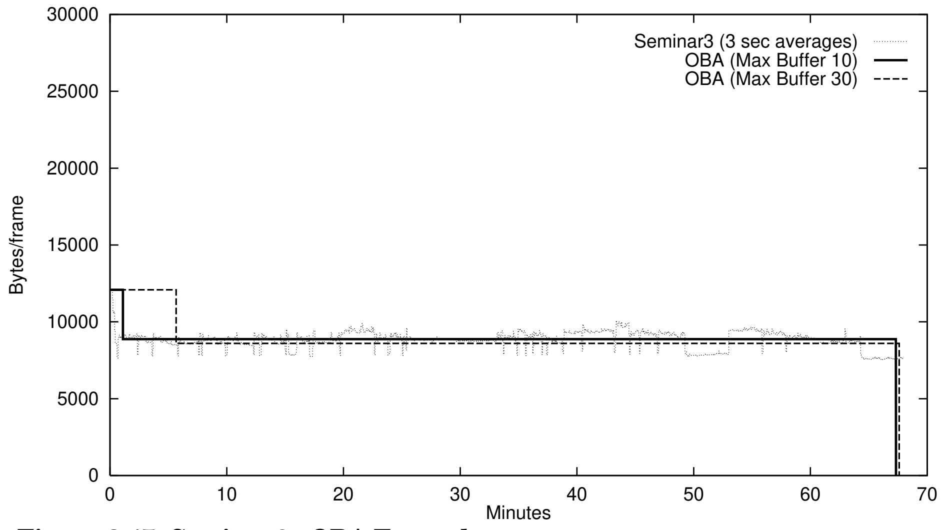


Figure 3.47: ***Seminar3*** **- OBA Example**

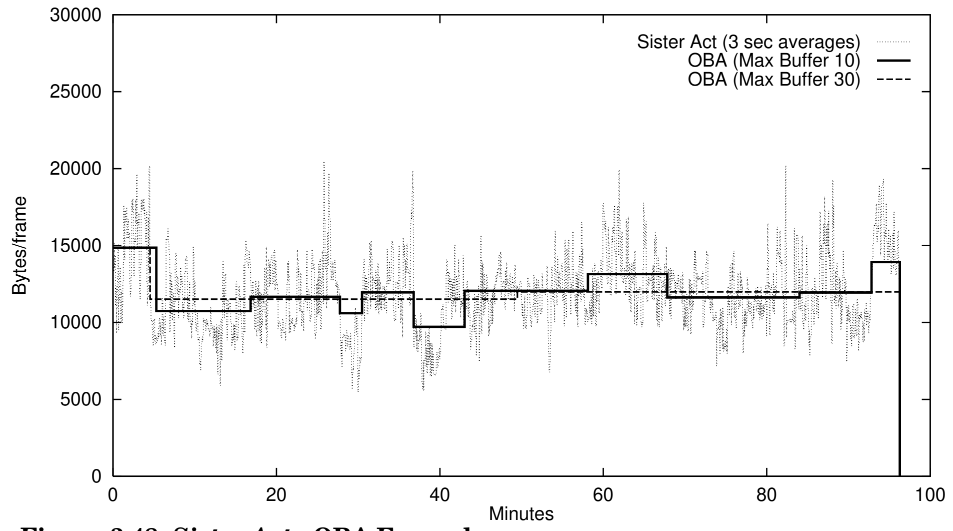


Figure 3.48: ***Sister Act*** **- OBA Example**

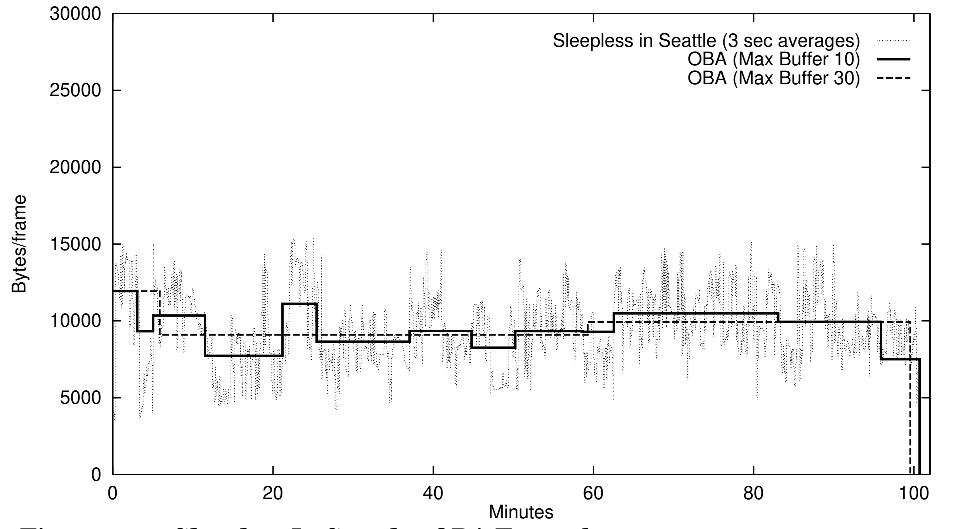


Figure 3.49: ***Sleepless In Seattle*** **- OBA Example**

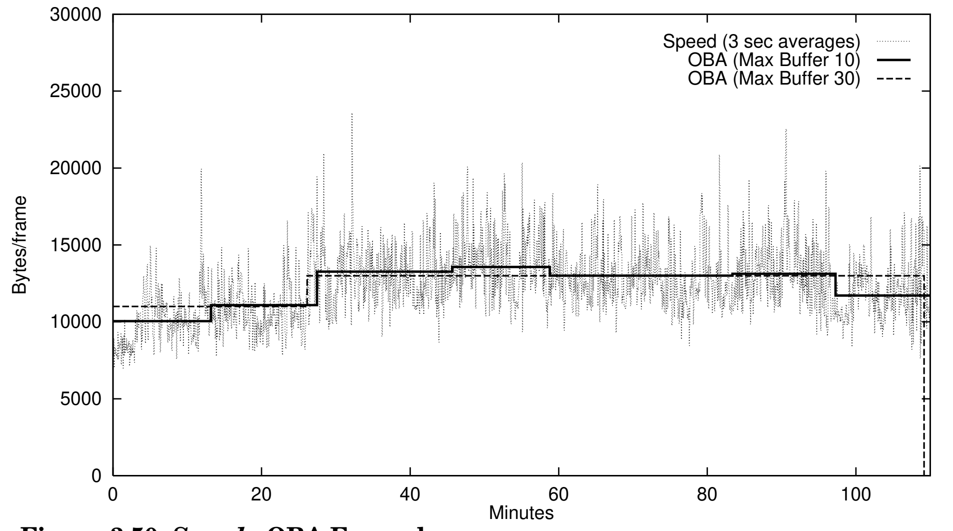


Figure 3.50: ***Speed*** **- OBA Example**

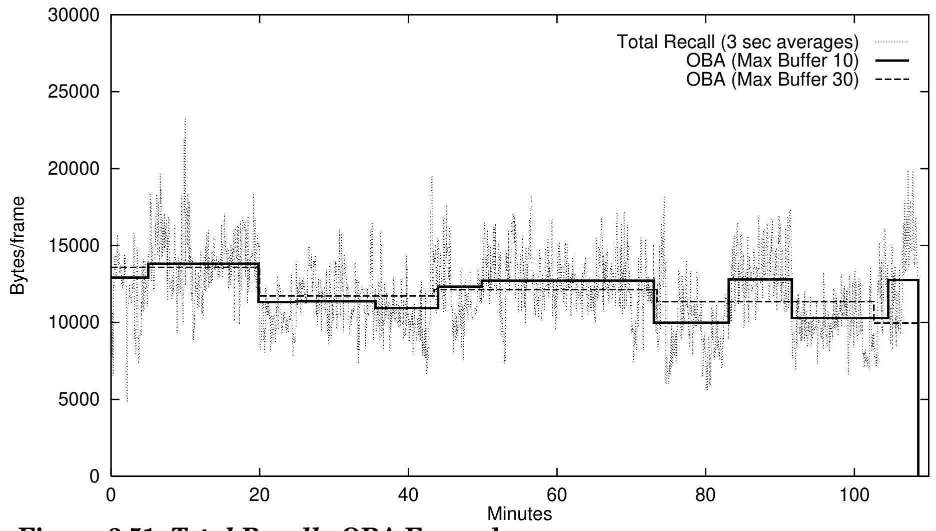


Figure 3.51: ***Total Recall*** **- OBA Example**

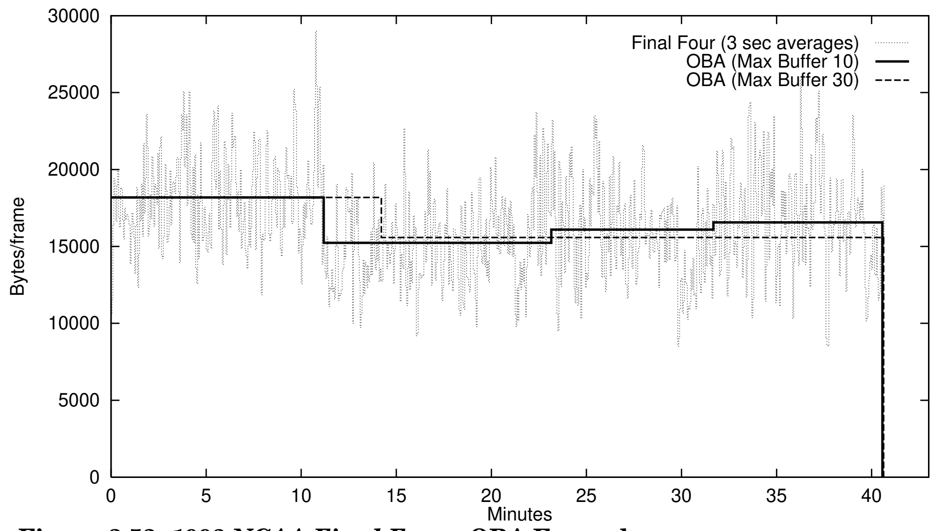


Figure 3.52: ***1993 NCAA Final Four*** **- OBA Example**

CHAPTER 4

DELIVERING VIDEO ACROSS NETWORKS

"If you build it, they will interact." - J. Tierney, New York Times (June 30, 1993)

4.1 Introduction

The delivery of constant quality compressed video requires that the network adapt to large fluctuations in bandwidth. We have shown that bandwidth smoothing techniques are effective in removing burstiness, making network resource scheduling simpler. In this chapter, we describe how critical bandwidth based smoothing techniques affect the underlying services in an interactive video-on-demand service.

As previously mentioned, the benefits of smoothing for live video applications are constrained by the requirement that latency remain low between video capture and video playback. Stored video applications, on the other hand, can schedule the network bandwidth resources well in advance of the playback of the video. The use of in-advance reservations in such bandwidth smoothing schemes, however, has implications on the ability to provide users with familiar video cassette recorder (VCR) functionality such as stop, pause, rewind, and fast forward.

For constant quality video delivery, the video bandwidth requirements can be smoothed by prefetching data into a buffer, shifting bursts of large frames forward in time. Depending on the amount of buffering available, a frame may sit in the client's smoothing buffer for a shorter or longer time before it is played back. Long buffer residency times are often required to reduce large peak bandwidth requirements. Despite these long buffer residency times, the rate of transmission and the rate of consumption remain coupled. Alterations in the consumption rate that occur with VCR functions will require alteration in the video delivery plan, lest the buffer overflow or underflow. In addition to the goal of high network utilization, a video-on-demand system must effectively handle the contradictory goals of smoothing versus responsiveness.

For video-on-demand systems with little or no buffering, the clients and servers must be tightly coupled. Any change in consumption of video data from the client must be immediately and continuously handled by the server. With buffering, changes in consumption still require an adjustment by the server. These adjustments, however, need not be made instantaneously. Many operations can be performed without requiring the delivery of any new data from the server. Larger buffers allow greater latitude in handling these stops, starts, and rewinds. With excess buffering specifically used for handling variations in consumption rate, it is possible to further decrease the required disruptions of the server by combining the changes in consumption into a few requests. The number of disruptions that the servers must handle is proportional to the buffer size used. If a majority of the rate changes can be handled by the client machine, then the network and servers can devote their resources to handling the special cases that may arise instead of handling cases which can be taken care of with appropriate buffering.

In this chapter, we discuss a framework for providing VCR functionality and in-advance bandwidth reservations within a bandwidth-smoothing, stored-video environment. We introduce the notion of *VCR-window,* the set of buffered frames within which full-function VCR capabilities are available without requiring changes to the bandwidth reservations. The size of the VCR-window is determined by the size of the client buffer. We expect that for reasonable buffer sizes a large proportion of VCR operations can be handled from the VCR-window, with the remainder requiring more involved client and server interactions. For accesses outside the VCR-window, we describe a strategy for using a *contingency channel* for users to renegotiate temporary bandwidth so that their plans can return to the bandwidth originally reserved. The use of the VCR-window along with the contingency channel affects the way in-advance reservations can be handled. We discuss the impact of providing VCR functionality on *a priori* bandwidth reservations and present a reservation system for interactive video-on-demand systems. Our results show that the VCR-window can be implemented with a small amount of additional buffering with little modifications necessary to the bandwidth reservations.

In this chapter, we first describe classes of video-on-demand applications that require access to stored video. In Section 4.3, we describe the problems associated with providing VCR function capabilities to users and introduce a solution for providing limited VCR function capabilities. In Section 4.4, we present an admission control algorithm for video-on-demand resources. Using this framework, we then describe a resource reservation mechanism for critical bandwidth allocation based video delivery systems. Finally, in Section 4.5, we present the experimental simulations used to evaluate VCR-functionality in bandwidth smoothing environments.

4.2 Motivation

4.2.1 Video Playback Applications

Interactive video-on-demand encompasses a large range of applications. Because the social and economic impacts of video-on-demand systems are not yet understood, we can only offer a list of expected applications. As an example, consider broadcast television programming that is in use today. Television stations solicit companies to pay for commercial slots to generate revenue. If television becomes true video-on-demand, however, only a few commercials would ever be viewed in their entirety, thus undermining their purpose. In such a situation, we can probably expect video-on-demand suppliers to provide “catalog” type services for users who wish to

.

Application	Description
Movies-on-demand	Customers can select and play movies
Video-on-demand	Customers can access archived television and sporting events.
Interactive News T.V	An interactive newscast in which customers have the ability to define which stories are selected.
Catalog Browsing	Customers view commercials for goods and services.
Distance Learning	Customers can view lectures on selected topics of their choice.

Table 4.1: Video Interactivity. This table lists some common applications of video on demand services that may be used in the future.

browse commercial offerings. Several services that have been talked about in the literature are shown in Table 4.1. The actual requirements for interactivity and the load placed on the underlying network depends heavily on the type of application. Because interactive news consists of many smaller pieces of video, the video server can expect more variation in access patterns, especially for news items that the customer does not find interesting. Movies, on the other hand, are longer running and are focused on a single theme. As a result, the amount of interaction may not be as large.

For bandwidth smoothing environments, the size, length, and burstiness of the video affects the way that VCR functions can be provided. In a server that typically handles small clips, all the videos may be downloaded to the client and all interactions are then serviced from the buffer. For servers that handle medium length clips (on the order of 15 to 45 minute length clips), the application of the CBA algorithm results in a single monotonically decreasing sequence of bandwidths for all clips with small amounts of buffering. In this case, the server can take advantage of the monotonically decreasing sequences to schedule the network effectively. For longer running videos, the scheduling of network resources becomes more complex because scheduling decisions last on the order of hours (and not minutes). The key to providing VCR functionality in these videos is the efficient use of the smoothing buffer.

4.2.2 Buffering Versus Delay

Using the CBA or OBA bandwidth smoothing techniques results in a trade-off between buffering and delay. To smooth large frame-size peaks such as those found at the end of the *Seminar* video (see Figure 4.1), the data in the burst must be prefetched before the peak is played back. Because VCR functionality is coupled with the buffering of frames, the burstiness exhibited by the videos impacts the ability to service VCR interactivity with buffering.

For our discussions, we use the digitized movie *Speed* and the *Seminar* video. As shown in Figure 4.1, the bandwidth allocation plans generated by the OBA algorithm result in very few required bandwidth changes for the playback of the videos. Note, the bandwidth plans generated by the CBA and OBA algorithms do not require any prefetching before the playback of the videos begin, hence, a high initial bandwidth may be required (as in the *Seminar* video). If the video request is made in advance, this initially high bandwidth requirement can be removed by prefetching data before the start of playback of the video. It is important to recall that the *Seminar* and *Speed* videos are Motion-JPEG encoded, thus, they do not take advantage of temporal similarities between frames. This results in buffering and bandwidth estimates that are conservative for the results shown. In general, using MPEG encoded video streams instead of Motion-JPEG encoded video streams results in one of two situations: 1) The actual buffer requirements are smaller than presented (assuming the same buffer residency times) or 2) The buffer residency times will be much higher than the numbers presented (assuming the same buffer sizes) because more frames can be prefetched with the same amount of buffer.With the use of MPEG's B and P frame types, the amount of buffering required to achieve the same amount of smoothing can be expected to be 4 to 10 times smaller.

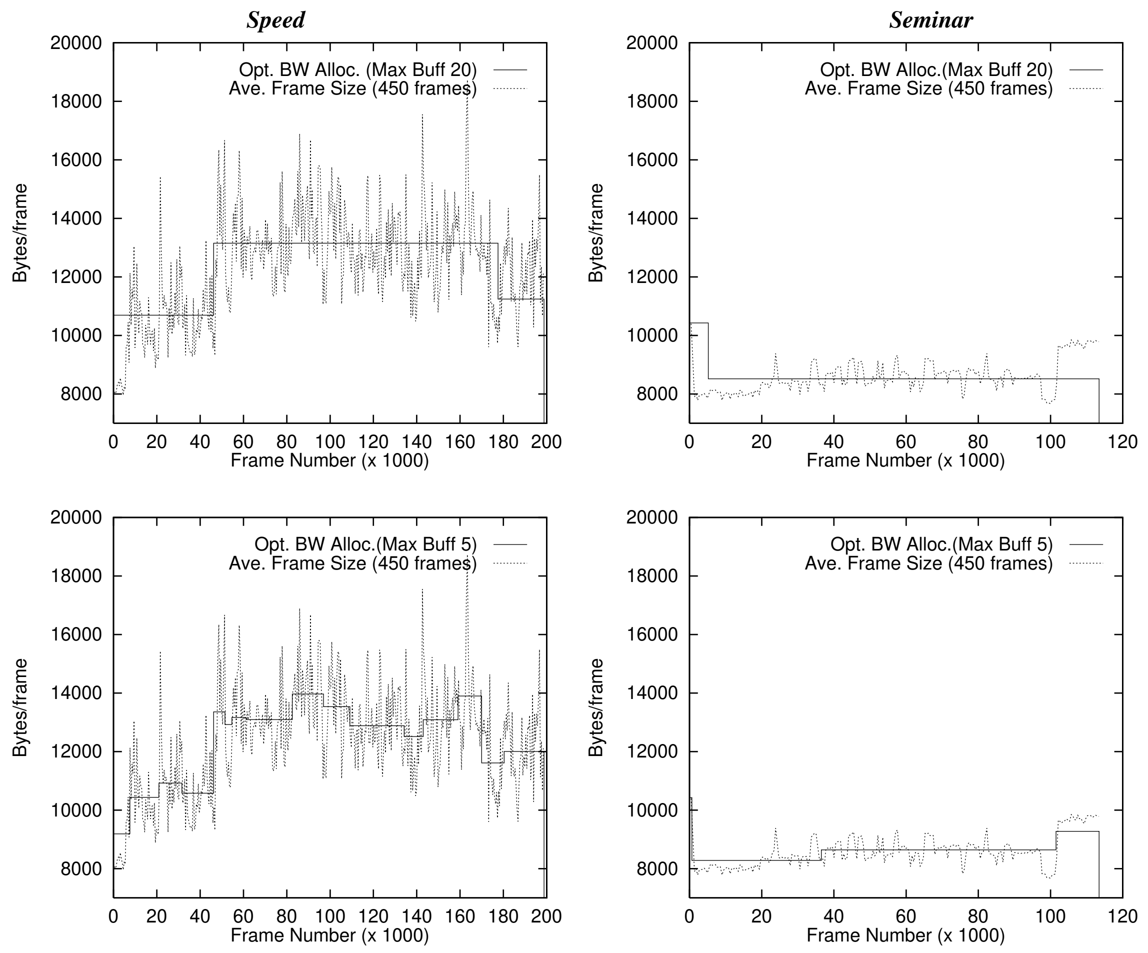


Figure 4.1: Bandwidth Smoothing Examples. This figure shows the plan created using the optimal bandwidth allocation algorithm with a 20 MByte and 5 MByte buffer for the Motion-JPEG encoded videos *Speed* and *Seminar.* The dashed lines represent the average frame sizes for 15 second (450 frame) groups within the videos.

Because the CBA and OBA algorithms smooth bursts as much as possible through prefetching, the buffer residency times (the time that a frame sits in the buffer) can be fairly substantial. The buffer residency times for the videos *Speed* and *Seminar* using the optimal bandwidth allocation plans from Figure 4.1 are shown in Figure 4.2. As shown by Figure 4.2, the buffer residency times are correlated to the amount of buffering used for smoothing. That is, the larger the buffer used, the higher the buffer residency times tend to be. In addition, the variance of buffer residency times also depends on the long term burstiness of the video. As described in Figure 4.2, the amount of time a frame spends in the buffer can be on the order of

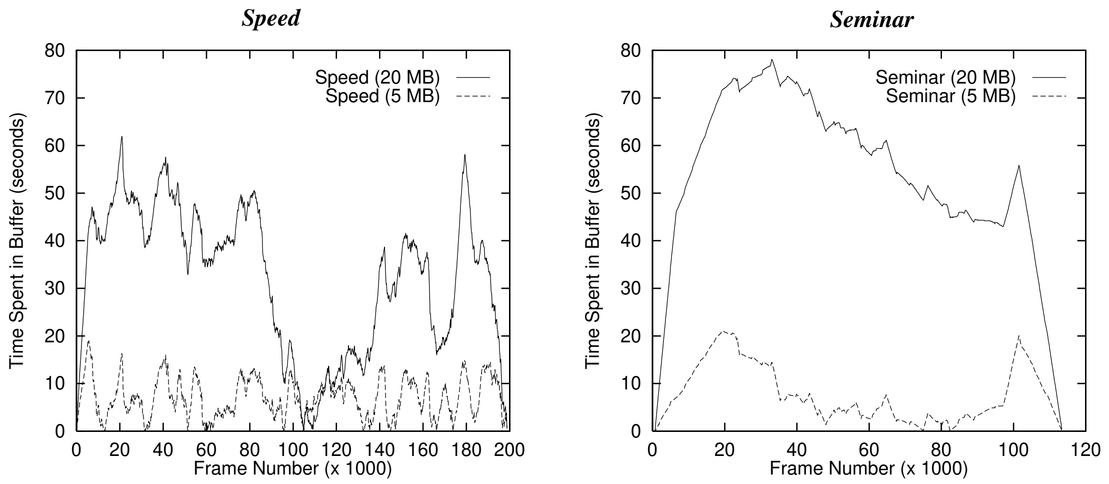


Figure 4.2: Buffer Residency Times. This figure shows the buffer residency times for the frames in the Motion-JPEG videos *Speed* and *Seminar* using a 5 and 20 MB buffer. For these videos and a 20 MB buffer, the average buffer residency time was 32.1 and 54.0 seconds for the *Speed* and *Seminar* videos, respectively. Note that for equivalent size buffers the residency times for MPEG encoded videos would be considerably larger.

half a minute to a minute for a 20 MB buffer. For the *Speed* video, there is a larger variation in frame sizes throughout the movie, resulting in buffer residency times that also vary more. For the *Seminar* video, the stream consists of roughly the same size frames except at the end where a larger bursts of frames occur. As a result, large buffer residency times are required to smooth out the large frame sizes at the end of the video. Incidentally, the end of the *Seminar* video consists of the lights turning on, a panning of the speaker toward the center of the room, and a short question and answer session that includes the first row of listeners.

The large buffer residency times introduced by non-window based smoothing techniques have a direct impact on the ability to provide users with VCR capabilities. If random access to any point is to be allowed (while keeping the video quality constant), the network may have to contend with a potentially large required burst in bandwidth above that originally allocated. This large burst of extra bandwidth may be required to make up for the absence of buffering (and delay) to help reduce the bandwidth requirements. Thus, providing VCR capabilities in a bandwidth smoothing environment can be a difficult task.

4.3 VCR Functionality

Any interactive video-on-demand system must bring together several interrelated issues such as scheduling of disk resources at the server, reserving underlying network bandwidth, and handling rate consumption changes. The support for VCR-functionality can be handled at several different layers. Most notably, a fair amount of work is focused on supporting VCR functions such as rewind-scan and fast-forward-scan at the server [9,13,16,74]. These systems assume that the delivery of data during the use of VCR functions is not buffered and is aimed at providing interactivity with frames being delivered from the server. While providing VCR functionality may ultimately require the server to run in this mode, it suffers from scalability problems from the overhead in the number of changing requests. On the other hand, for systems that deliver constant quality video and use bandwidth smoothing to reduce peak bandwidths, the bandwidth allocations (especially if made in advance) are somewhat rigid to change because of the buffer residency times needed to smooth bandwidth requirements. In this section, we describe a video-on-demand service that has several key features: constant quality video delivery and VCR functionality (with the *VCR-window*). Before describing the VCR-window, we first describe the type of interactivity that we expect to see in future video-on-demand systems.

4.3.1 VCR Interactivity

In a bandwidth-smoothing video-on-demand system, providing unconstrained full-function VCR capabilities can cause problems with the ability to deliver the required video data due to lack of network resources. By looking at the expected interactions during the playback of video, the video-on-demand system may be designed to take advantage of common interactions. For video-on-demand services, we believe that video-on-demand users typically change the access pattern during the playback of a video that fall into one of the four categories:

- *Pause/Stop* - The user stops the movie for a short time to answer a phone call, go to the kitchen, etc.
- *Rewind* - The user rewinds the video to play back part of the video that was not understand
- *Examine* - The user stops the VCR to examine more closely a portion of the video. As an example, a user may be watching a football game and wants to see a certain play a couple of times in slow motion to see why it did or didn't work.
- *Fast forward scan* - The user scans past parts of the video such as commercials in the program.

In the future, we believe that users may also require all of these functions from a video-on-demand system, although the actual distribution of access patterns within these categories may change. As an example, consider the operation fast-forward scan, which typically gets used to fast-forward through commercials. As mentioned earlier, it is unclear how commercials will play a role in future video-on-demand systems. Nevertheless, we should not rule out the possibility of fast-forward scans in our discussion. We expect, however, that many of the accesses will be in a localized area within the video, thus, providing a limited window of full function VCR capabilities may suffice for most applications. In addition, by limiting the window size, the network bandwidth reservation levels may not need to be altered, and the required interactions with servers and networks may be minimized.

4.3.2 The VCR-window

To allow for VCR functionality, we propose a different model of video delivery which allows users to have full function VCR controls in a limited window called the *VCR-window.* In our model of video transfer, we allow all VCR functions to occur at

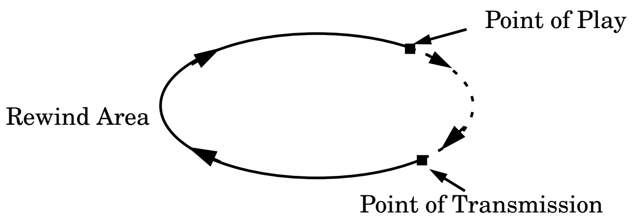


Figure 4.3: Circular Buffer Example. This figure shows the conceptual model of buffering for video data. The point of play (POP) and the point of transmission (POT) move clockwise. The solid line represents the rewind area, while the dashed line represents the amount of buffering that is in use for prefetching (smoothing)

any time within the course of playback but limit the range of accessible data without having to renegotiate the reserved bandwidth. We define the notion of the point of play (POP) to be the furthest frame in the video that has been viewed by the user and the point of transmission (POT) as the furthest frame in the movie that resides in the client buffer. Our model then consists of viewing the smoothing buffer as a circular buffer, in which, the POP and POT traverse the circumference in a clock-wise manner (see Figure 4.3). The distance the POT is ahead of the POP is the amount of buffer space used for prefetching. The remaining part of the circumference, the *rewind area*, is the amount of data that has been played back and is still in the buffer. Thus, when the buffer is nearly full, the POT is just behind the POP, and when the buffer is nearly empty, the POP is just behind the POT. Note, if the POT ever passes the POP or the POP passes the POT, we have buffer overflow and buffer underflow, respectively.

Using these definitions of the POT and POP, we make the observation that we can allow the user to have full function VCR capabilities in the area that the POP leads the POT without changing the bandwidth reservation level. No change in bandwidth is required because the data that is being viewed is sitting in the buffer already. One major drawback of this method is that when the buffer is nearly full the POP does not lead the POT by any significant amount. To ensure that the rewind

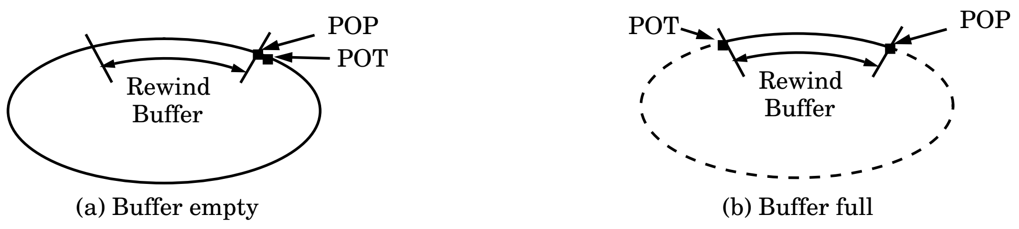


Figure 4.4: Buffer Limit Conditions. Figures (a) and (b) show the cases that occur during playback when the buffer is empty and full, respectively. The solid line represents data that has been played back but not removed from the buffer, while the dashed line represents data that has been transmitted but not played back.

area has some minimum amount of data, we define the *rewind buffer* to be the closest distance that the POT can approach the POP. For clarity we still refer to the total distance that the POP leads the POT as the *rewind area* (or the VCR-window). Figure 4.4 shows the resulting two cases when the buffer is full and when the buffer is empty for the VCR-window.

Formally, we can define the amount of available data in the rewind area on the ith frame as

$$RewBuffSize(i) = MaxBuff - \left(\left(\sum_{j=0}^{i} BwAlloc(j)\right) - \left(\sum_{j=0}^{i} FrameSize(j)\right)\right)$$

where,

- •*MaxBuff* is the maximum buffer size including the *Rewind Buffer.*
- •*BwAlloc(k)* is the bandwidth allocation on frame k. Note, we assume that the bandwidth allocation plan is allocated in bytes/frame.
- •*FrameSize(k)* is the frame size of the kth frame.

This equation takes the difference between the total bandwidth received and the total bandwidth played back (i.e. the amount of data in the buffer) and subtracts it from the amount of buffering available. This equation does not, however, calculate the amount of video that is actually available but calculates its aggregate size. We can calculate the additional amount of rewind buffer needed to have T frames available in the buffer on the ith frame as

$$AddBuffReq(i, T) = \left(\sum_{k = max(0, i-T)}^{i} FrameSize(k) \right) - RewBuffSize(i)$$

This equation is essentially the size of the T frames needed in the rewind area with the amount of data already in the rewind area subtracted. If the user requires that 100% of the time the buffer has T frames in it, then the additional amount of buffering needed is simply the maximum AddBuffReq() over all frames within the movie. That is, to ensure the buffer *always* has T frames in the minimum buffer requirement *MinBuffReq(T)* is:

$$MinBuffReq(T) = \max_{T \le j \le N} AddBuffReq(j, T)$$

To allow for VCR capabilities, when a user starts moving in the rewind area, the data flow from the server is stopped. The flow is then restarted only when the playback point reaches the POP again. Thus, only two interactions to the server and network are required to support the VCR-window, one to stop the data flow, and one to start the data flow again. Because the delivery of bandwidth starts at exactly the same point at which the data was stopped, no changes in the bandwidth reservation level are necessary while allowing for a window of full function VCR capabilities. For long term bandwidth reservations, the only modification necessary is the extension of the bandwidth requirement by the amount of time that was spent in the rewind area. Using this model for VCR functionality, the operations stop/pause, rewind, and examine can be provided to users with only minimal interaction with the network and server. As an example, consider the bandwidth allocation plan shown in Figure 4.5. At time t, the user decides to stop the playback and examine the video that was just

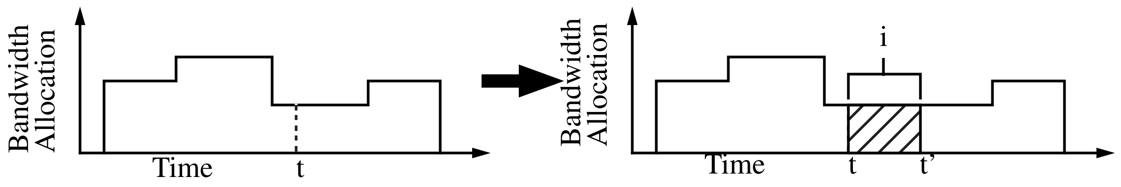


Figure 4.5: VCR-Window Example. This figure shows the adjustment in the bandwidth allocation plan that is made when a user uses the VCR controls for *i* time units (include the time to get pack to the POP that it was at). The remaining portion of the bandwidth allocation plan is then shifted by the amount of time spent in the rewind area. In this case, it is shifted *i* time units.

played. Suppose that the total time it takes for the user to examine the video and get back to time *t* in the video is *i* time units. We then move all bandwidth allocations after the time *t* in the original bandwidth allocation plan to start at time *t'*, the time at which playback started again. Note, by shifting the bandwidth allocation plan after time *t* by *i* time units, the resultant bandwidth reservation has been modified. We discuss how the reservation scheme can be modified to handle this change later in this chapter.

4.3.3 Access Outside the VCR window

Scans to points outside the VCR window require renegotiations with the network and server. For long fast-forwards, the consumption rate originally anticipated will now be compressed in time, resulting in the need for more bandwidth than was originally planned for, assuming all the frames are delivered. We expect that these interactions may not occur very frequently, nonetheless, they should not be disallowed. For the renegotiation of bandwidth reservations in these cases, we expect that the notion of *contingency channels* which reserve part of the network capacity to handle changes in consumption rate are useful [13]. Because the VCR-window filters many of the interactions that are required through the use of buffering, the contingency channels can be more efficiently allocated to handling the special cases that may arise during the playback of video.

Our work on contingency channels is derived from the work at IBM on batching of the delivery of video data for clients watching the same movie with the same approximate playback times [13]. In their work, the authors consider the use of pause/resume functions for the VCR controls on how multiple channels can be batched together, hence, the contingency channels are only used to "unbatch" a client that has paused the video and is not temporally close enough to the channel it was originally batched with. The "unbatched" client is then re-batched with possibly a different group of clients watching the same movie at about the same playback time. The actual bandwidth allocation for the movie, however, is not addressed in this work as well as the use of any buffering that may be used to smooth bandwidth requests. As a result of this, the contingency channel may be used by a client for a short time or for a very long time depending on where the playback time of other batched users is.

For our purposes, we use the contingency channel in a slightly different manner. We do not consider batching of movie channels but use the idea of setting aside bandwidth for the occasional accesses that are made outside of the VCR-window and only use the contingency channel for providing VCR functionality and not batching. Therefore, the contingency channel in our work is used by clients for a short amount of time until their bandwidth requirements reach a level that is within their originally agreed upon bandwidth reservations.

For accesses outside the VCR-window, it is important to get the bandwidth consumption rate for the clients back to their original bandwidth reservations as soon as possible. By returning the system back to its steady state as soon as possible, the network manager needs to only worry about scheduling the contingency channel part of the access outside of the VCR-window (as opposed to re-admitting the entire bandwidth plan). As an example, consider the optimal bandwidth allocation delivery plan shown in Figure 4.6 that has a point to be randomly accessed. Ideally, an impulse of data would be issued at the point of random access and the bandwidth requirements would then fall below the bandwidth requirements for the entire original bandwidth

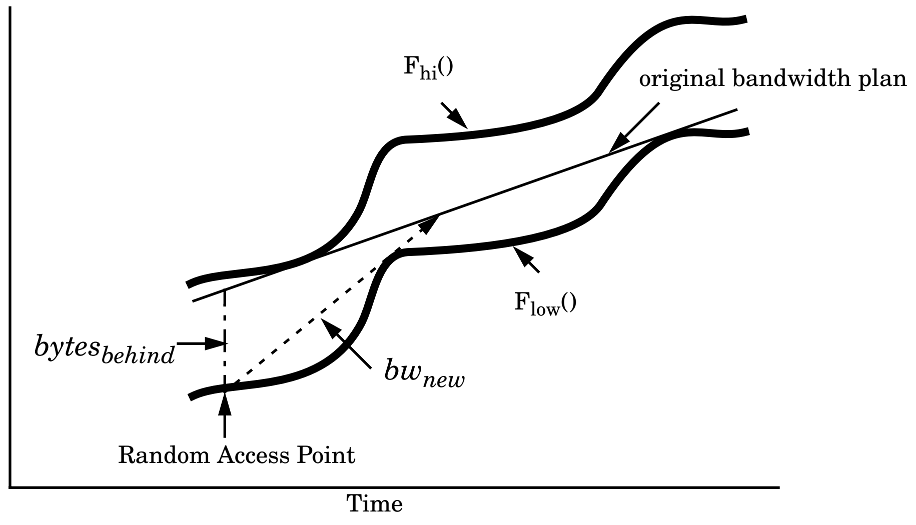
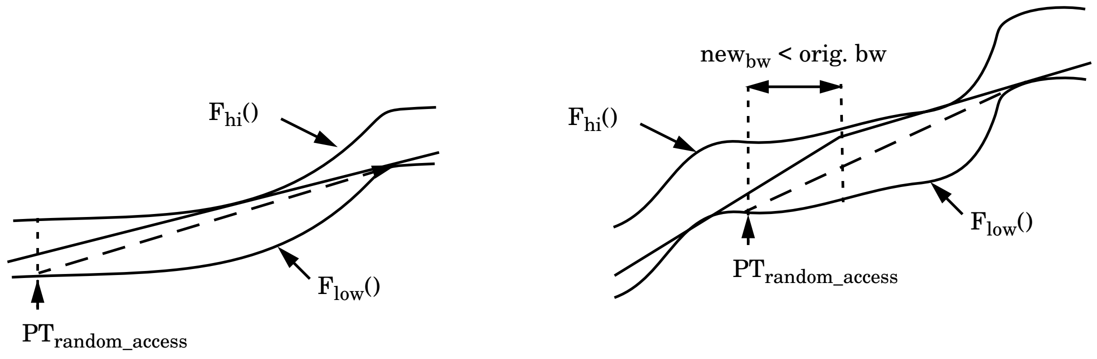


Figure 4.6: Random Access and Smoothing. This figure shows the handling of accesses to a random location within a video delivery plan. The dashed vertical line shows the necessary amount of data required to be sent via the contingency channel. The minimum amount of bandwidth necessary is determined by the critical bandwidth starting from the random access point with 0 bytes in the buffer. Once the plan reaches the original allocation, the contingency channel for that client can be freed.

plan immediately. Because impulses of data are not possible, a plan for the use of the contingency channel must be created. For this random access at point PT_{access}, let BW_{new} be the critical bandwidth from PT_{access} *with no bytes in the buffer*. In addition, let $Bytes_{behind}$ be the difference between $F_{low}(PT_{access})$ and the buffer occupancy of the original bandwidth plan at PT_{access}. Then, we note that any bandwidth greater than BW_{new} does not cause buffer starvation and that once $Bytes_{behind}$ have been delivered the contingency channel can then be freed.

For access to random starting points outside of the VCR-window, the critical bandwidth is generally larger than the original bandwidth allocation at that point. It is, however, entirely possible that the critical bandwidth from the random starting point is nearly the same as (or even below) the original bandwidth allocation at the random point. As an example consider, the two examples shown in Figure 4.7. In Figure 4.7(a), the critical bandwidth is nearly the same as the original plan. Because the runs in the optimal bandwidth allocation algorithm can run in the tens of min-



(a) new_{bw} ~ original bandwidth (b) new_{bw} < original bandwidth

Figure 4.7: Critical Bandwidths and Random Starting Points. In these figures, thin solid lines are the original bandwidth allocations, while the dashed lines are the critical bandwidths starting from PT_{random_access}. Figure (a) shows an example starting point where the bandwidth requirement for the contingency channel is nearly the same as the original bandwidth allocation, while Figure (b) shows an example starting point where the bandwidth requirement for the random starting point is less than the actual original bandwidth allocation plan.

utes, the actual difference in bandwidth requirement may be very small. Figure 4.7(b) shows an example where the critical bandwidth is actually below the bandwidth at the random access point.

In general, the accesses that are made to points outside of the VCR-window are expected to be in the vicinity of the playback point. As a result, the data in the buffer may still be usable, thereby reducing the actual bandwidth requirement from the contingency channel.

Contingency Channel Allocation - An allocation back-off approach

Implementing the contingency channel poses an interesting problem - how should the contingency channel bandwidth be allocated? For the random access that is shown in Figure 4.6 and Figure 4.7, any bandwidth greater than the bw_{new} will work. Thus, a trade-off between using all of the bandwidth of the contingency channel for a shorter period of time, or allocating at the critical bandwidth bw_{new} and using the contingency channel for a longer period of time must be made. The former allows the contingency channel to be freed as soon as possible, but risks starving other outstanding requests. The latter approach may not make the best use of bandwidth available on the contingency channel. To balance between these goals we propose an *allocation back-off strategy* for the allocation of the contingency channel.

The *allocation back-off strategy* allocates the entire available contingency channel bandwidth to a single client when no other clients require servicing. When another request arrives, the network manager picks one (or more) of the users to decrease its bandwidth requirement either in half or to its critical bandwidth to allow the new user to use the contingency channel to re-synchronize its bandwidth allocation plan. When a client is done with its contingency channel allocation, the network manager then notifies one of the clients to increase its use of the contingency channel. By allocating the contingency channel in this manner, the network manager can make the most use of its contingency channel bandwidth, while assuring users a reasonable response time. In the event that the contingency channel is entirely allocated, a user may need to start the retrieval of bandwidth at a smaller rate (thus delaying the actual start-up of the video). With the filtering provided by the VCR-window, we expect that the number of times that the contingency channel reaches capacity and cannot handle other requests will be fairly rare.

4.4 An In-Advance Reservation Scheme

Resource reservations are an important part of network management for both in-advance and on the fly reservations because the network can then accurately estimate the bandwidth requirements of the clients. For stored video-on-demand services, the ability to provide reservations of bandwidth in advance can make the job of resource allocation easier[56]. Resource reservation schemes have two key components that are necessary for resource reservations: the bandwidth requirement and the duration that the bandwidth requirement is needed [14,32,84]. Without providing these, resource reservations in-advance then becomes a difficult task. In addition,

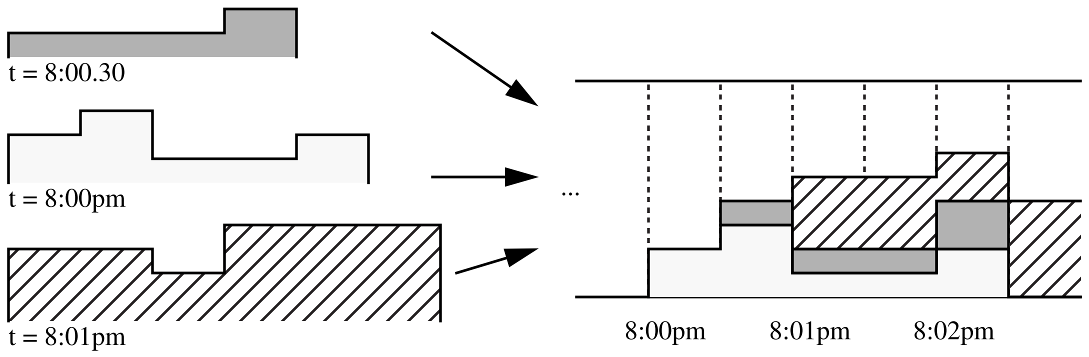


Figure 4.8: Resource Reservation Scheme. This figure shows the reservation of bandwidth for three sample streams. Bandwidth is reserved in 30 second intervals to reduce fragmentation of bandwidth.

Ferrari, Gupta, and Ventre point out that scheduling of bandwidth based on some fixed interval reduces the fragmentation that the reservation scheme has to contend with[32]. Finally, it is commonly agreed upon that advance reservations will consist of two distinct phases, an admission control phase where the reservation is admitted and an enforcement phase where the bandwidth allocation is enforced.

Our in-advance reservation model is a slotted reservation scheme with a minimum bandwidth allocation slot of 30 seconds. The user machine/set-top-box creates a bandwidth allocation plan based on the slot boundaries and then passes this plan to the server and network for admission control. The network managers then compare the bandwidth requirements of the new channel and compare it to the available bandwidth allocation plan offered by the user. The example in Figure 4.8 shows sample requests that may be sent to the network manager. By using 30 second bandwidth allocation slots, the network manager only needs to evaluate 180 slots for a 90 minute video, reducing the complexity of the admission control algorithm. The network manager then allocates the available resources to the new channel if available. If the available bandwidth does not exist, then the network manager can either 1) offer a new starting time which can satisfy the bandwidth allocation plan or 2) create a different network path to the server which can satisfy the request.

Passing bandwidth plans to the network manager as part of admission control creates rigid schedules. To allow VCR-window functionality, the resource reservation system must reserve bandwidth based on the maximum amount of "VCR-time" to be introduced by the viewer. The total time that the video can be delayed must be declared at admission control. The actual amount of VCR-time delay reserved depends on the guarantees that the user expects and the quality of service expected if the delay bounds are exceeded. The amount of VCR-time reserved may be also determined by economic factors (i.e. how much users are willing to pay for bandwidth that they may not use). For now, we assume the worst case for this delay, in that, all of the delay can occur at any interval. Therefore, in the calculation of the bandwidth allocation plan used for admission control, we create a bandwidth allocation plan that reserves the data such that at each point within the movie the video can be stopped for the maximum delay. Let *T* be the delay (in frames) for VCR functionality that is required from the user and *BwPlan(i)* be the bandwidth requirement on frame *i*. Then the new bandwidth allocation plan (in bytes/frame) can be defined as

$$NewBwPlan(i) = \max_{i - T \le j \le i} BwPlan(j)$$

Figure 4.9 shows a sample bandwidth allocation plan calculation that has the expected VCR-induced delay built into the bandwidth allocation plan.

4.5 Experimentation

The success of the VCR-window concept depends on how much data is available in the rewind area at any given time as well as the amount of buffering required for use as the rewind buffer. As the amount of buffering devoted to the rewind buffer increases, the amount of smoothing available diminishes. The success of the reservation system depends on the effectiveness of the video-on-demand system to utilize its bandwidth. In order to fully understand the impact of buffering on the VCR-window and the associated in-advance reservation system, we have digitized 17 full-length

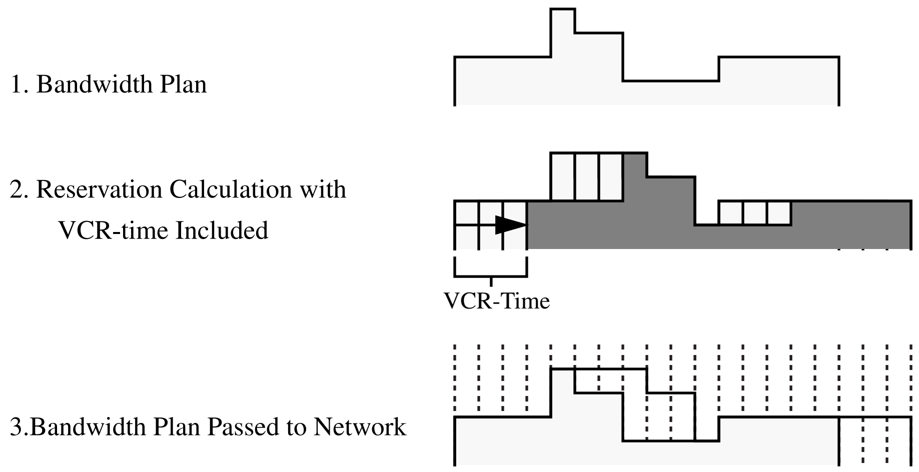

1. Bandwidth Plan
2. Reservation Calculation with VCR-time Included
VCR-Time
3.Bandwidth Plan Passed to Network

Figure 4.9: Bandwidth Reservation Calculation. 1) Client machine creates bandwidth allocation plan. 2) Client machine creates second plan that incorporates "VCR-Time" in the plan. 3) The VCR-time calculated bandwidth is passed to the network and server as part of admission control. This plan is denoted by the heavy solid line.

movies along with 3 seminars that were presented at the University of Michigan. For the experiments, we used the OBA algorithm with prefetching on the initial run.

4.5.1 VCR-window Experimentation

The success of the VCR-window depends on the amount of rewind buffer required for the guarantees expected from the user. Using the CBA or OBA algorithms with a reasonably large buffer, the buffer is full only a small number of times. Thus, very little additional buffering for use as a rewind buffer may be required.

Figure 4.10 shows for a given amount of time, *x*, the probability that *x* amount of video data is available in the rewind area with the rewind buffer size set to 0. For the *Speed* video, using a 25 MByte buffer results in over half a minute of video in the rewind area 53% of the time while using a 50 MByte buffer results in over half a minute of video in the rewind area 75% of the time. In addition, 15 seconds of video is available 74% and 91% of the time for the 25 and 50 Mbyte buffers. The *Seminar* video exhibits similar numbers to the *Speed* video. As shown in Figure 4.11, the percentage of time that the rewind area contains more than 30 seconds of video for the

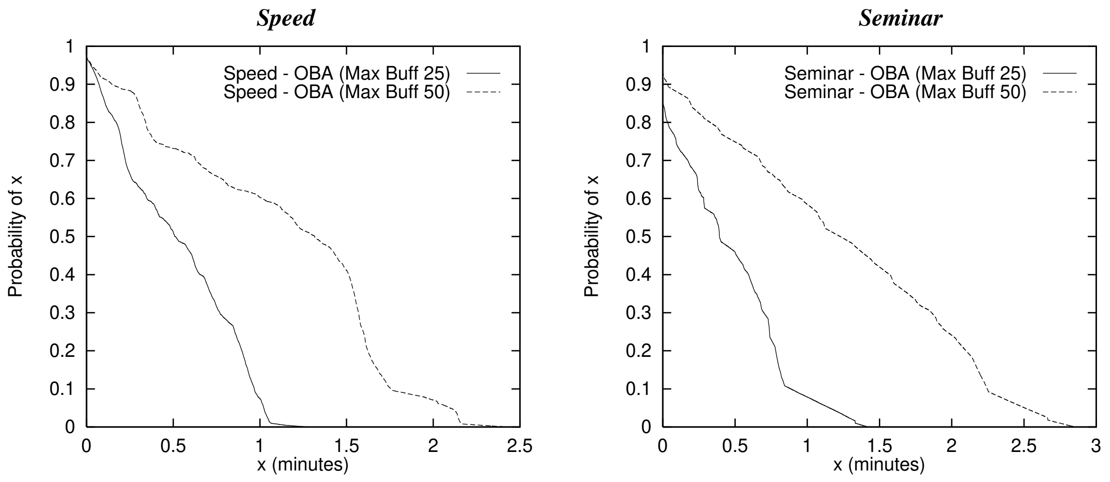


Figure 4.10: Buffer Rewind Probabilities. These graphs show for a given amount of time, *x*, the probability that *x* amount of video is available in the rewind area without renegotiation and with the rewind buffer set to 0. For 25 and 50 Mbyte buffers, this results in having 30 seconds of video available 52.8% and 75.3% of the time, respectively, for the video *Speed*. Similarly, the *Seminar* video had 30 seconds of video available 46.9% and 77.9% of the time, respectively.

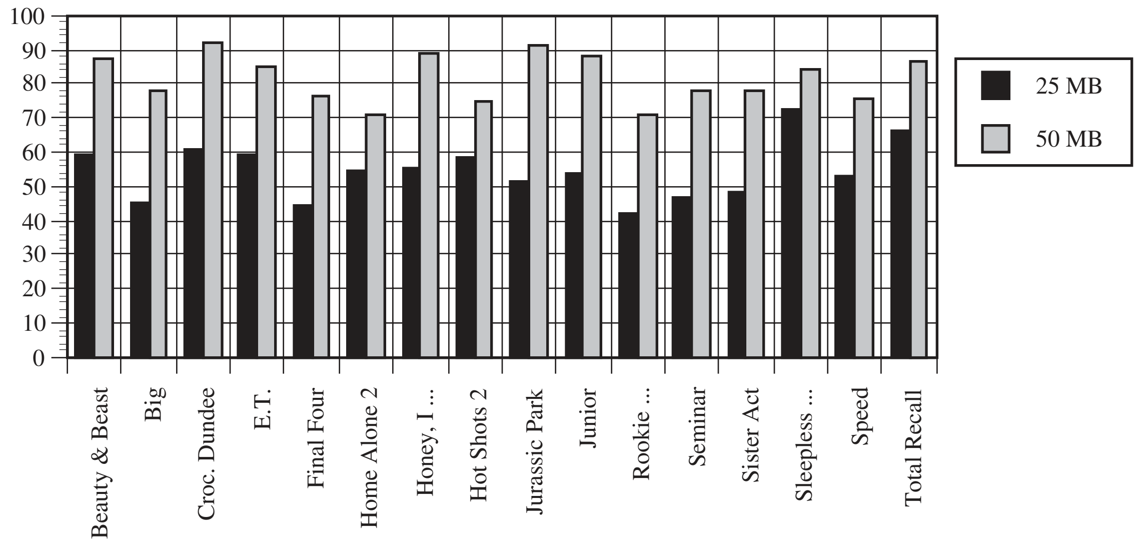


Figure 4.11: Buffer Rewind Times for all Movies. This figure shows the percentage of time that rewind area contains more than 30 seconds of video for the 25 and 50 MByte smoothing buffer with the rewind buffer size set to 0.

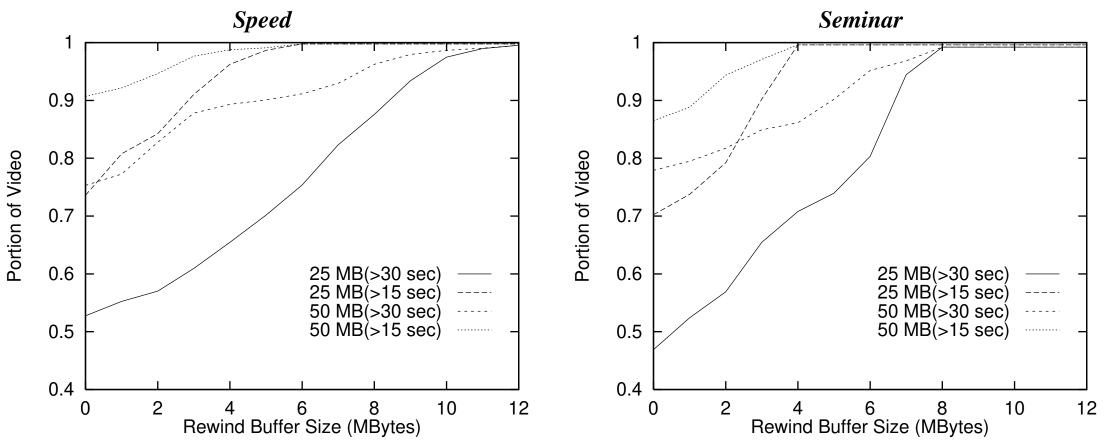


Figure 4.12: Buffer Rewind Size Requirements. These figures show the rewind buffer size requirement necessary to ensure that some percentage of the time, the VCR-window has 15 and 30 seconds of video in it for the movie *Speed*. As an example, with a 25 MByte smoothing buffer, in order to achieve 15 seconds of buffering 90% of the time, roughly 3MBytes of rewind buffer is required above the 25 MByte smoothing buffer.

rest of the videos exhibits similar numbers as well. Typically, the 25 and 50 MB buffers results in the rewind area with 30 seconds of video 45-60% and 75-90% of the time, respectively.

The addition of a rewind buffer shifts the probability curves from Figure 4.10 to the right, thus increasing the amount of time that is available in the rewind area. Figure 4.12 shows, the amount of buffering needed for the rewind buffer size in order to have the rewind area contain a certain percentage of video in the rewind buffer greater than 15 and 30 seconds. Note, these buffer rewind sizes are in addition to the 25 and 50 MByte buffers used for the smoothing of bandwidth requirements. As expected the lines for the same time (15 and 30 seconds) approach the same required rewind buffer size because this buffer size is determined by the same point (area) within the video. In addition, the amount of required rewind buffer space decreases as the size of the smoothing buffer increases. This is mainly due to the larger buffer sizes having more rewind area on average. In order to achieve at least 15 seconds of video in the rewind area 95% of the time, the movie *Speed* requires only 4 and 2 MBytes of rewind buffer for the 25 and 50 MByte smoothing buffers, respectively. The

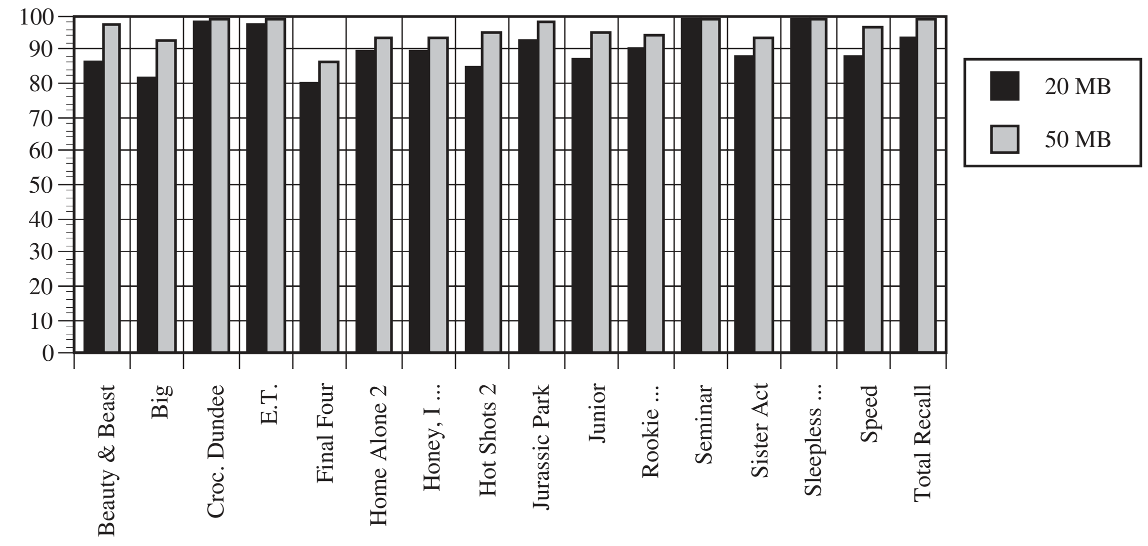


Figure 4.13: Buffer Rewind Sizes for All Movies. This figure shows the percentage of time that 30 seconds of video is available using an 8 MByte rewind buffer in addition to the 25 and 50 MByte smoothing buffer.

Seminar video approaches the 100% line faster than the *Speed* video. This is due to the smaller (and more constant) average bit rate of the *Seminar* versus the *Speed* video. If we take the average frame sizes for the videos and multiply it by the number of frames in 30 seconds of video (900 frames), the *Speed* video results in 11.1 MB while the *Seminar* video results in 7.7 MB. It is interesting to note that these videos approach 100% near these values.

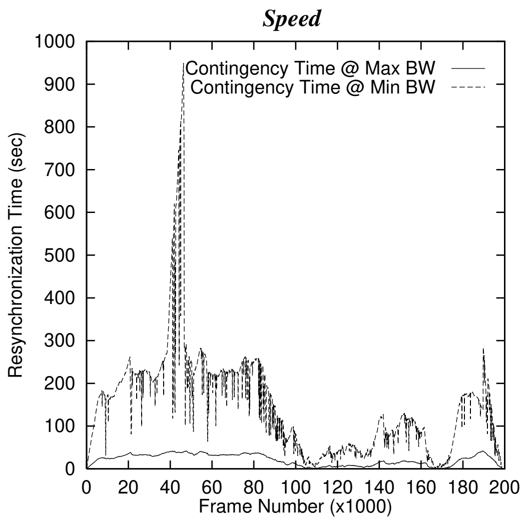
Figure 4.13 shows the percentage of time that 30 seconds of movie is available when using an 8 MByte rewind buffer. The highest bit rate video *Final Four* results in the smallest percentage of rewind times greater than 30 seconds, while the 3 smallest bit rate videos (*E.T., Crocodile Dundee,* and *Sleepless in Seattle*) result in the highest percentage of rewind times. This suggests that the rewind size is roughly correlated to the average bit rate that the encoded movie has. Thus, we expect that the use of tighter encoding schemes such as MPEG with B and P frames reduces the overall requirement of the rewind buffer size.

4.5.2 Accesses Outside the VCR-window

The use of contingency channels to serve requests outside of the VCR-window should not occur that frequently, however, if a user accesses an area outside of the VCR-window, the contingency channel needs to provide bandwidth for the client to return to its originally agreed upon reservation. To show the expected resource requirements for random accesses outside of the VCR-window, we assume that the network link layer is a 100 Mbit/sec link and that only 5% of the link (i.e. 5Mbits/sec) is reserved for the contingency channel. For accesses outside of the VCR-window, we then graphed the time required for resynchronization using the entire contingency channel bandwidth. as well as the re-synchronization time at the critical bandwidth. Because the critical bandwidth can be approximately the same or lower than the original bandwidth allocation, we force the minimum contingency allocation to be at least 90 kbytes/sec, which is approximately one quarter the average bit rate of the videos that we have digitized and approximately 14% of the contingency channel capacity. This avoids excessively long re-synchronization times due to a very small difference in the original bandwidth allocation and the critical bandwidth from the starting point.

Figure 4.14 shows the re-synchronization times for the videos *Speed* and *Seminar*. The re-synchronization times using the entire contingency channel are on the order of half a minute for both movies. For contingency channels that have twice as much bandwidth, the graph for the contingency channel at the maximum bandwidth is scaled in half. As shown by the graphs, the contingency times at the minimum are directly related to the amount of data in the buffer that need to be made up. As a result, the graphs for the contingency times are very similar to the buffer occupancy graphs in Figure 4.2.

Accesses just outside of the VCR-window may be able to take advantage of data that is already sitting in the buffer, thereby, reducing the amount of resources



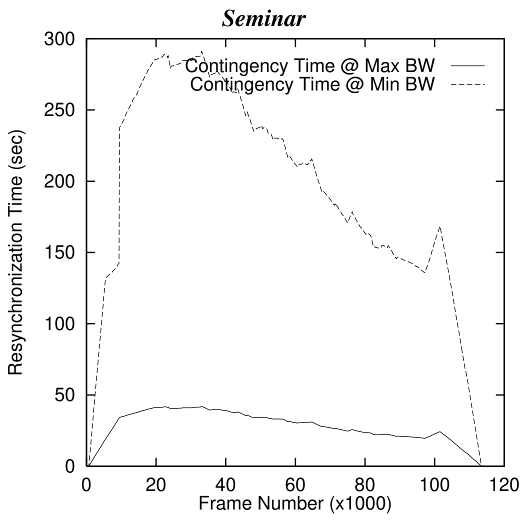


Figure 4.14: Random Access Times for Contingency Channel Usage. This figure shows the resynchronization time required for the bandwidth usage of a random access to return to the originally allocated bandwidth (using a 25 MB smoothing buffer and the OBA algorithm). The solid lines represent the time required for re-synchronization using the entire bandwidth of a 5Mbit/sec contingency channel. The dotted lines show the re-synchronization times for allocating the bandwidth of the contingency channel at the critical bandwidth from the random starting point or at 90kbytes/sec, which ever is higher.

required from the contingency channel. The reduction of contingency channel resources is, of course, directly related to what data is buffered and where the new point of play is. In Figure 4.15, we have graphed the contingency channel usage for forward accesses 10 seconds outside of the VCR-window. As shown, by the graphs, the average amount of time needed to re-synchronize with the original bandwidth allocations is much smaller than random accesses (as in Figure 4.14). As an example, the videos *Speed* and *Seminar* need, on average, 4 and 6 seconds of the entire contingency channel bandwidth to re-synchronize with the original bandwidth plans. The sharp peak in the contingency times for the movie speed results from an increase in bandwidth in the original bandwidth allocation plan. The increase in bandwidth is almost the same as the critical bandwidth from the starting point, resulting a very long contingency channel time. Allocating the contingency channel at its maximum bandwidth, however, removes this spike.

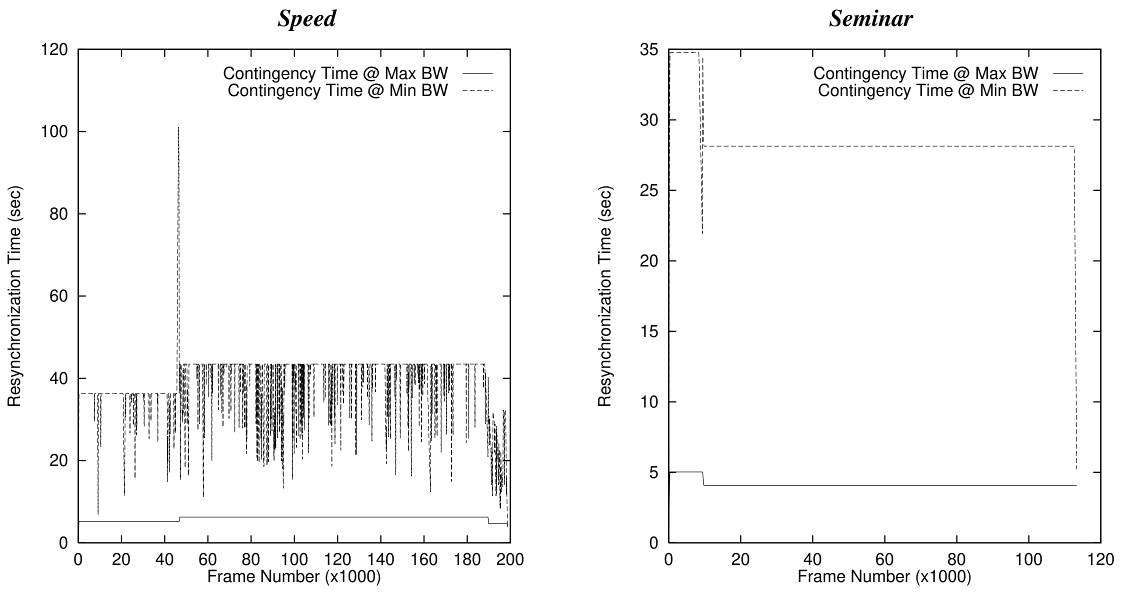
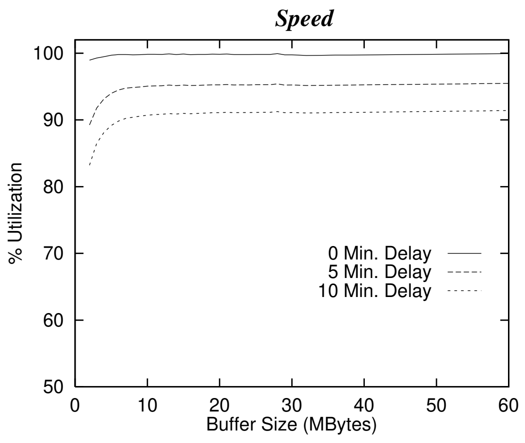


Figure 4.15: Contingency Channel Usage for Local Accesses. This figure shows the resynchronization time required for the bandwidth usage of accesses that are 10 seconds outside of the VCR-window (using a 25 MB smoothing buffer and the OBA algorithm). The solid lines represent the time required for re-synchronization using the entire bandwidth of a 5Mbit/sec contingency channel. The dotted lines show the re-synchronization times for allocating the bandwidth of the contingency channel at the critical bandwidth from the random starting point or at 90kbytes/sec, which ever is higher.

4.5.3 Bandwidth Reservations

For bandwidth reservations, one of the main concerns is the actual network utilization versus the amount of bandwidth that was allocated. For example, if the amount of bandwidth reserved is around 95% but only 50% of the bandwidth is actually used, then the reservation scheme may need to be modified to be more effective in utilizing the network. As shown in Figure 4.16, the reservations based on a 30 second period with no extra delay built into the reservation plan for VCR functionality yield reservation utilizations between 99% and 100% of what was reserved. Thus, the OBA allocation (with prefetching on the first run) yields bandwidth allocation plans that utilize nearly all the bandwidth reserved based on 30 second periods and suffers very little internal fragmentation of bandwidth allocations. Furthermore, the utilizations can be increased by aligning bandwidth change boundaries with the slot bound-



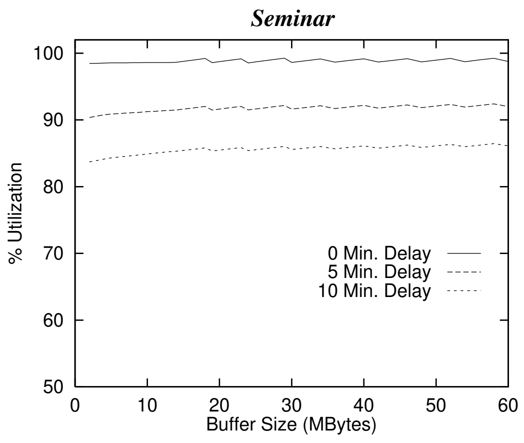


Figure 4.16: Reservation Utilization. These graphs show the reservation utilization for the video *Speed* and *Seminar* with reservations made on a 30 second period and reserved with the maximum additional delay expected during playback.

aries. Three trends are worth noting in Figure 4.16. First, for very small buffer sizes (< 5 MB), the utilization is hurt by two things, more bandwidth changes that are not aligned with slot boundaries and more bandwidth is reserved due to the limitation of the prefetch buffer in smoothing bandwidth requests. Second, because the utilization for the OBA algorithm is quite high, the utilization for the streams reach their limits fairly quickly. Finally, the *Seminar* video has lower utilization for the 5 and 10 minute delays because the video is shorter in length, thus, the 5 and 10 minute delay reservations make up a larger portion of their reservations. With tighter encoding mechanisms, the utilization can be expected to be higher with no other modifications to the buffer size or delay for VCR functionality.

The expected overall utilization of the network is not captured by the graphs in Figure 4.16 because they do not capture the peak reservations which may affect other bandwidth allocation plans. To establish a “lower bound” on the expected network utilization, we have modified the bandwidth allocation plans to have both the peak bandwidth reservation for the *entire* video and the expected VCR induced delay. A sample graph allocation is shown in Figure 4.17. As a result, the peak bandwidth allocation makes the reservation for the *entire* movie as one constant bandwidth res-

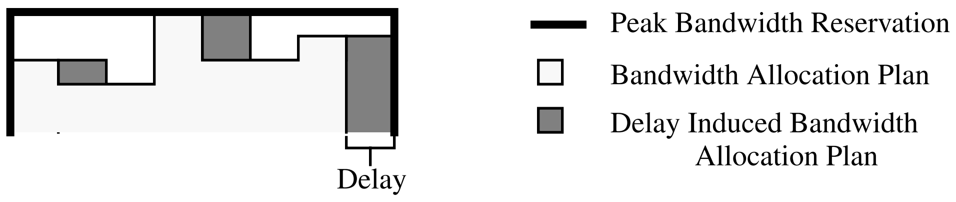

Peak Bandwidth Reservation
Bandwidth Allocation Plan
Delay Induced Bandwidth Allocation Plan
Delay

Figure 4.17: Peak Bandwidth Reservations. The heavy solid line shows the creation of a peak bandwidth allocation plan. This bandwidth allocation plan is used for the advanced reservations made in Figure 4.18.

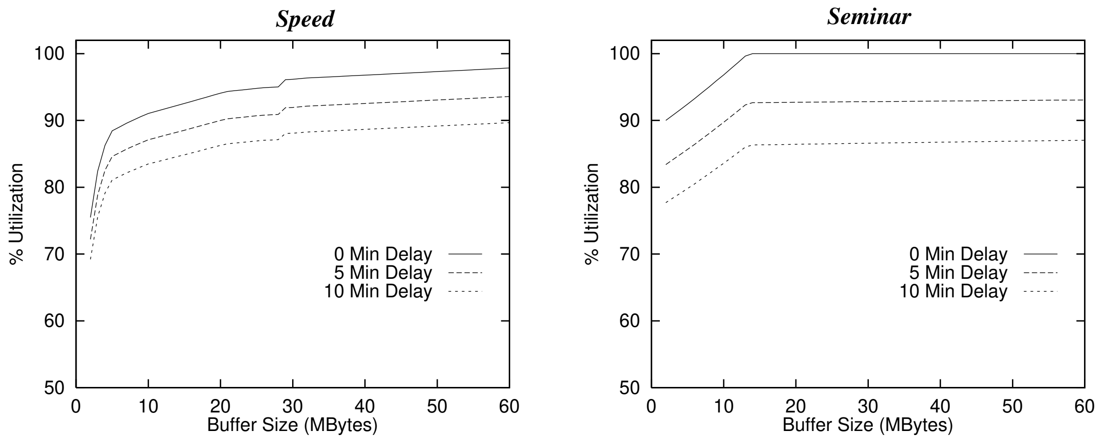


Figure 4.18: Peak Reservation Utilization. These graphs show the peak reservation utilization for the video *Speed* and *Seminar* with reservations made at the peak bandwidth allocation and with the maximum additional delay expected during playback.

ervation. We then graphed the expected bandwidth utilization based on these peak bandwidth allocations instead of the bandwidth reservations described in Section 4.4. As shown by Figure 4.18, we see that the bandwidth utilization has dropped from those shown in Figure 4.16. The bandwidth utilizations, however, are still reasonable. Using a 25 MByte smoothing buffer, no rewind buffer, and an extra 5 minutes of delay in the bandwidth reservations, the movie *Speed* has a utilization of 90.7% while the *Seminar* video has a utilization of 92.7%. Thus, even for peak bandwidth reservations with 5 extra minutes reserved, we expect that the bandwidth utilization can be held fairly high. The peak bandwidth reservations do not affect the *Seminar* video as

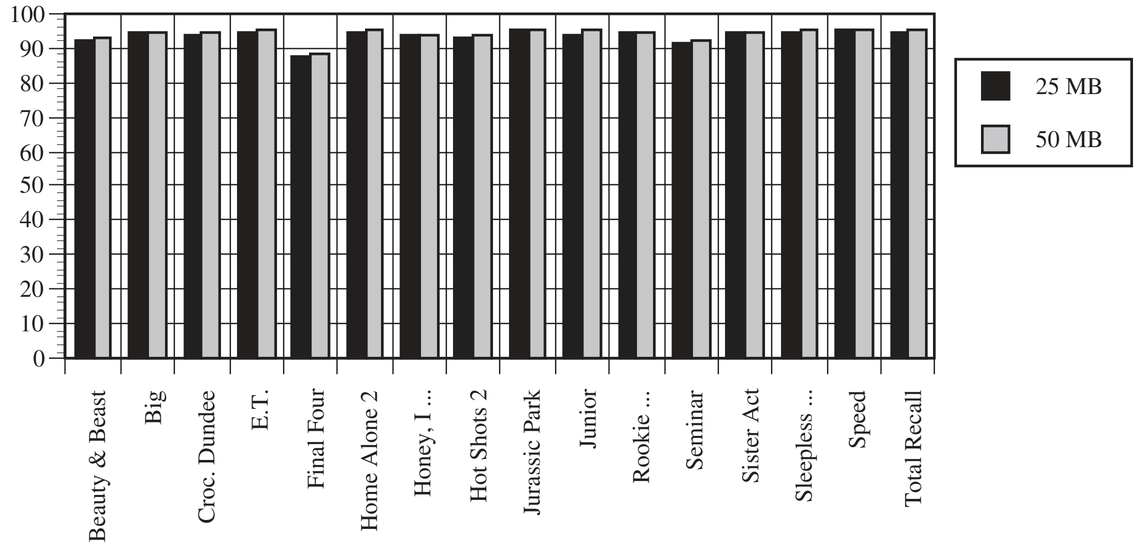


Figure 4.19: Reservation Utilization for Other Video Data. This figure shows the reservation utilization for bandwidth plans that are allocated in 30 second periods and have 5 minutes of delay added to the reservations. All utilizations were in the 90% to 95% range with the exception of the *Final Four* video.

much as the *Speed* video because it has less variation between frame sizes resulting in smaller peaks when they occur.

Finally, Figure 4.19 and Figure 4.20 show the normal reservation utilizations and peak reservations utilizations in the same exact way that Figure 4.16 and Figure 4.18 were made, respectively. In Figure 4.19, the normal reservation scheme with an additional 5 minute delay for both 25 and 50 MByte buffers are shown for all movies. They result in utilization ranging from 90% to 95% with the exception of the *Final Four* video. This result is expected as the *Final Four* video is only 41 minutes in length, thus, the extra 5 minutes accounts for 11% of the video. As expected the peak utilizations are generally less than the normal reservation method. In addition, the 25MB buffers are affected more because they cannot remove the peak burstiness as much as with a 50 MB buffer. Nonetheless, they exhibit fairly good utilizations. The video *Beauty and the Beast* video is affected the most by using a peak bandwidth reservation. The reason this occurs is that a peak of very large frame sizes could not be

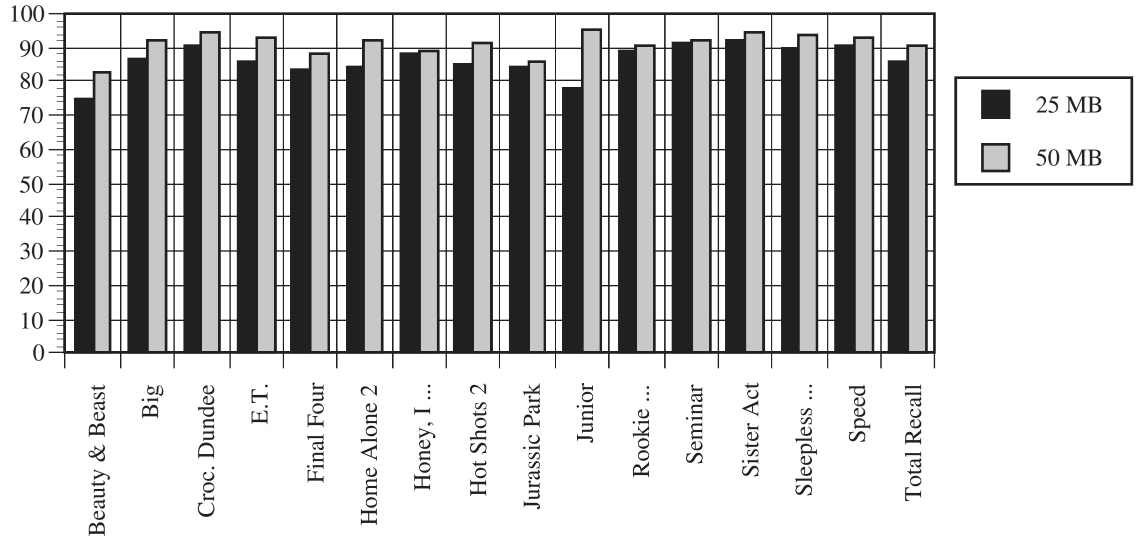


Figure 4.20: Peak Reservation Utilization for Other Video Data. This figure shows the peak reservation utilization for bandwidth plans that are allocated in 30 second periods and have 5 minutes of delay added to the reservations.

overcome with smoothing in a small area within the movie. We expect that these singular peaks may be scheduled to fit into the valleys of other bandwidth allocation reservations. Some movies exhibit no change in utilization from the reservation utilization to the peak reservation utilization. In these cases, the amount of buffering in the reservation utilization is enough to remove almost all of the burstiness and thus does not get affected as much by using the peak bandwidth requirement. In general, however, the overall expected bandwidth utilization of the network can be expected to be fairly high.

4.6 Conclusion

In this chapter, we have introduced the notion of *VCR-window* which allows a user to have full function VCR capabilities within a constrained region that does not change the bandwidth allocation requirements. We have also shown that providing 30 seconds of video available for 90% of the time can be implemented with a small amount of additional buffering, even for loosely encoded Motion-JPEG video. We expect that the majority of interactions that occur during the playback of video can be accounted for by using this technique. For users who want a guarantee of some amount of video *always* available in the rewind area, the required rewind buffer size is determined by a few scenes within the movie. For users who are willing to settle for lesser guarantees during the examine or scan phases, the VCR functionality can always be provided by the server which can fit the required video into the reserved channel capacity. Work on supporting scan operations from the server can be found in [9,16,74], while modifying compressed video to fit within a specified channel capacity can be found in [60,61].

We have also presented a slotted, in-advance resource reservation scheme to be used in conjunction with the VCR-window. The optimal bandwidth allocation algorithm results in very high network bandwidth utilization even under periodic scheduling boundaries. This is mainly due to the optimal bandwidth allocation algorithm minimizing the number of bandwidth changes as well as the peak required bandwidth. Nonetheless, the total amount of smoothing available depends on the long-term burstiness of the data itself. Using the advance reservation scheme in conjunction with the optimal bandwidth allocation algorithm allows users to have 5 to 10 minutes of "VCR-time" without degenerating the utilization. We expect that the lower bound for network utilization to be at least 80 percent. The 5 to 10 minutes of extra reserved "VCR-time" can be allocated for users to browse commercials or previews of other movies, assuming that they fit into the bandwidth reservation or are viewed at a slightly lower quality.

In the event more "random" access patterns are required such as jumps or scans of more than a couple of minutes in the video are required, renegotiation of bandwidth most likely is required or the reservation of bandwidth with that is a lot higher than actually used. The size and magnitude of these contingency channels depends on the percentage of times that the users in the video-on-demand system stray from the VCR-window. While we expect that the frequency of these occurrences to be quite small, the video-on-demand system should provide this flexibility. Our results indicate that allocating as little as 5 Mbits/sec of contingency channel can allow users that make local accesses outside of the VCR-window to re-synchronize with their original bandwidth allocation plans within a few seconds. These times are expected to be even smaller with tighter encoding of the video. Finally, for random accesses, the use of indexing schemes to allow access at distinct points within a video may allow the bandwidth requirements to be handled in a more efficient manner for accesses outside the VCR window.

CHAPTER 5

MPEG SOFTWARE VIDEO DECOMPRESSION

"The biggest difference between time and space is that you can't reuse time" - Merrick Furst

5.1 Introduction

This chapter addresses the issue of delivering data to the user for viewing using software video decompression. Video compression technologies take advantage of redundancy within a video stream to reduce the size of the videos. This redundancy occurs in three directions, in two dimensions between adjacent pixels in a single frame and in a third dimension between the different frames. By taking advantage of this redundancy, high compression ratios are possible, however, the reconstruction of the original frames may be slowed because of the dependencies within these large data sets. As a result decompression algorithms such as MPEG can be limited by both the sheer number of instructions required to decompress the video and by the data rates required from the memory system.

Work has begun on reducing the number of instructions, for example, by combining several byte operations into a single larger operation. Computer manufacturers, such as HP and DEC, will support such operations in hardware by using split data paths in their future computers [40]. This approach has also been implemented purely in software to support high-speed video decompression [5,19]. Other work has used loop-unrolling and other compiler techniques to speed up the decompression algorithms [64]. Reducing the number of operations executed, however, increases the rate of data consumption, requiring greater data bandwidth from the memory system.

Data bandwidth for user applications has traditionally been supplied through high speed cache memories. Applications such as MPEG, however, have been viewed as naturally cache inefficient because of the amount of data that it uses is large relative even to secondary cache sizes, which typically range from 64 to 256 Kbytes. This inefficiency is ironic because video compression such as MPEG is made possible by using portions of selected frames to predict the contents of other frames. Thus, the decompression relies on high bandwidth from memory, which is the primary limitation in the performance of a software MPEG [64].

In this chapter, we address the problem of reducing the number of access required to main memory during software video decompression by decreasing the cache miss ratios. Decreasing the miss ratios exhibited by MPEG decompression can be implemented in one of two ways: (1) by reducing the “working-set” of MPEG such that the *key* data that dependent macroblocks depend on stay in the cache longer, or (2) predicting the data to be accessed in the decompression and prefetching the data before it is needed.

In the next section, we further motivate this problem and provide the necessary background for the discussion of MPEG video decompression in software and processor caching. In Section 5.3, we describe two ways of reducing the “working-set” of a standard MPEG player as well as provide the experimental results. In Section 5.4, we describe how software-controlled prefetching for MPEG video decompression impacts the underlying architecture, both from a cache miss rate standpoint and from a memory bandwidth point.

5.2 Motivation

Three trends in computer architecture design will increase the importance of processor caching in future applications such as software video decompression. First, as Hennesey and Patterson have pointed out [43], processor speed improvements are far outpacing those for memory speeds, creating larger penalties for processor cache misses. As an example, current generation machines, with clock speeds near 100MHz typically have 70ns memory SIMMs, but the next generation DEC Alphas with clock speeds in excess of 230MHz will still be using 70ns SIMMs [17]. To somewhat alleviate this mismatch in speed, techniques such as expanded caching and prefetching can be used. Second, in order to achieve high processor clock speeds, on-chip caches have been made smaller to allow the cache to keep up with the processor. As examples, the Intel P6 processor has an 8K primary data cache backed by a 256K secondary data cache, while the Dec Alphas have an 8K primary data cache backed by a 512K data cache [42,17]. The final trend lies within the nature of the video and audio data. Video and audio compression and decompression algorithms generally perform many operations on data that is either 8-bits or 16-bits long. To help support software video decompression, computer manufacturers such as Hewlett-Packard will provide split data paths in future architectures to allow parallel computations on data that are only 8 or 16 bit computations. These future processors will consume data at a much faster rate than current generation processors. As a result, we expect the differences in cache performance to make larger and larger differences in the overall performance of decompression algorithms.

Decompression of MPEG in software generally is not associated with effective processor caching because of the large amounts of data that must be accessed. In an all I-frame encoded movie, no dependencies between frames exist, resulting in all the data being decompressed, displayed, and then thrown away with no re-access of data occurring across frames. With encodings that involve dependent P and B frames, the dependencies between frames generally do not survive long enough in the processor cache to be useful in decoding other frames. Consider the decoding of a 640x480 encoded video with two I frames surrounding two B frames that each depend on these I-frames (that is, IBBI). Each frame, once decompressed, requires 307200 (640x480) bytes of storage for the luminance channel and 76800 bytes for each of the two chrominance channels, resulting in up to 1.4 Mbytes of frame data being touched in the decompression of a single B frame. In comparison to the Intel P6 and the new Dec Alpha chips, each B frame touches enough data to fill (and empty) the on-chip caches 175 times, while filling the secondary cache 3 to 5 times. This figure does not include the data required by the application or window manager during its decompression. As a result, in the decompression of the following B frame, the *key* data from the I frames necessary to decode the dependent B frame are flushed out of the cache and have to be brought into the cache again.

5.3 Improving Processor Caching via Locality

In this section, we investigate the performance gain that may be available to algorithms using different traversal orders to decompress MPEG streams. We show that alternative traversals can result in improved decompression speeds and that these improvements are likely to increase with better instruction-level support for MPEG. Our goal is not to optimize the MPEG player for a specific architecture. Instead, we are interested in improving the caching behavior of such an application. These traversal orders come at the cost of additional overhead, but in some cases this overhead may well be made up for by improved cache hit rates. The use of markers within the frames can reduce this overhead considerably, making improved performance easier to achieve with a small cost in compression ratio.

To improve processor caching for video, we discuss two methods that allow for better cache performance for the playback of MPEG video clips. These approaches change the typical MPEG scanline decompression traversal of frames to improve the locality exhibited by dependent frames. In addition, we present two different algorithms that implement these methods depending on whether or not the MPEG video was coded with markers within the frame. In the following section, some motivation and background are presented. Section 5.3.1 introduces two possible techniques that can improve cache miss rates for MPEG video applications. In Section 5.3.3, a comparison of the new techniques to an unmodified MPEG player is presented. Finally, a summary and conclusion about the importance of caching to video applications is presented.

5.3.1 Vertical and Horizontal Striping

To improve the locality of the MPEG video player, macroblock dependencies in P and B frames on macroblocks from other frames must be exploited. The use of dependent macroblocks introduces coherence in 3 directions, between frames and between macroblocks in a single frame. By decompressing a stream a frame at a time and in scanline order, the standard decompression method exploits coherence in only one direction. By taking advantage of coherence in the other two directions, a reduction in processor cache misses is feasible.

Two methods can be used to take advantage of coherence between lines. The simplest way to improve locality is to shorten the length of the macroblock scanline and to reconstruct the frame in several vertical columns. By shortening the scanlines, the dependent macroblocks used in reconstructing one shortened scanline are more likely to still be in the cache when decompressing the next scanline. By using shortened scanlines, the frame is reconstructed with large vertical stripes. See Figure 5.1. We call this method *vertical striping*.

The second method involves grouping several lines into a larger “scanline”. To decompress these scanlines, the MPEG player reconstructs each of the scanlines within the larger scanline concurrently by decompressing the first macroblock in each scanline, followed by the second macroblock in each scanline, and so on. The tra-

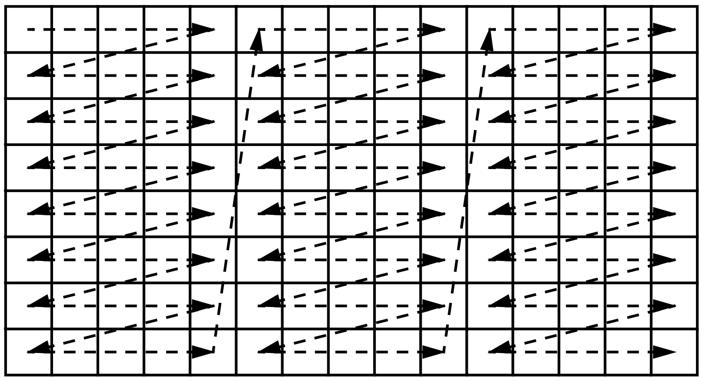


Figure 5.1: Vertical Striping Example. This figure shows a sample frame of video that is 15 macroblocks wide and 8 macroblocks high. The vertical striping traversal shortens all scanlines to improve coherence between scanlines, leading to large vertical stripes in decompression. Here, each vertical stripe is 5 macroblocks wide.

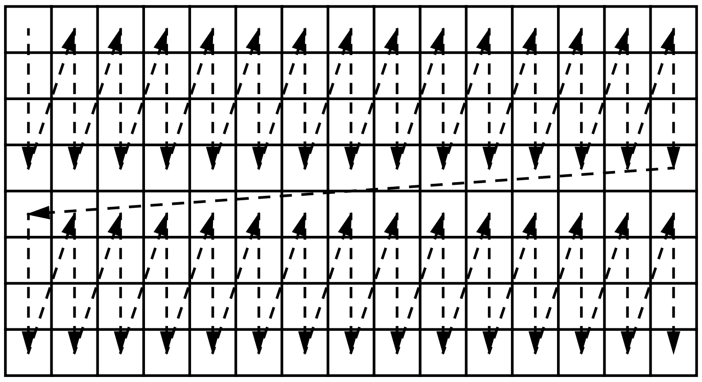


Figure 5.2: Horizontal Striping Example. This figure shows a sample frame of video that is 15 macroblocks wide and 8 macroblocks high. The horizontal striping traversal combines a few scanlines into a larger scanline that is decompressed in columns. Here, the traversal algorithm will multiplex between the 4 macroblock lines that are combined into the larger scanline.

versal results in a frame that is reconstructed with large horizontal stripes. This method is called *horizontal striping*. An example is shown in Figure 5.2.

To exploit coherence between frames, we then interleave the different frames while performing either a horizontal or vertical stripe traversal. For example, in using vertical striping, the first scanline of the first vertical stripe is decompressed,

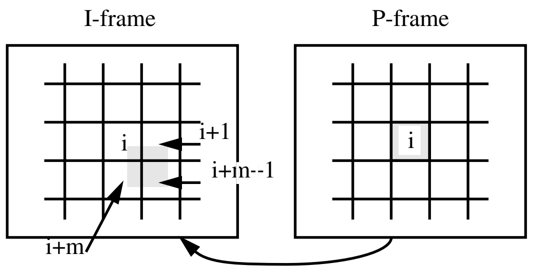


Figure 5.3: Macroblock Dependencies and Frame Interleaving. This figure shows a sample scenario of interleaving with *m* macroblocks per line. In the decompression of the *i*th P-frame macroblock with interleaving, the algorithms must make sure the I-frame stays "ahead" of the macroblocks that will depend on it. Thus, in the decompression of the *i*th macroblock in the P-frame, the macroblocks *i*, *i+1*, *i+m*, and *i+m+1* in the I-frame will be already decoded.

after which, all other first scanline-first vertical stripe are decompressed. One tricky situation with interleaving, however, is that the P and B frames may depend on data outside of the decoded area due to motion vectors. Consider the example shown in Figure 5.3. The macroblock i in the P-frame can depend on four other macroblocks from the I-frame that it depends on. Thus, with interleaving, we must make sure that the frames that are used in the decoding of other frames have their scanlines offset by the maximum motion vector. Typically, this offset at most one macroblock and is a fairly trivial calculation.

5.3.2 Implementing the Traversal Algorithms

Depending on how the MPEG video was encoded, two different algorithms can be used to implement horizontal and vertical striping. These stem mainly from the flexibility in the guidelines for encoding videos. Both of the algorithms require *two* passes to process the data, otherwise, the ability to change the order of decompression traversal is not possible. The first pass consists of finding the offsets to the macroblocks at the beginning of each macroblock scanline within the frames in order to allow different traversal algorithms. The second pass consists of decompressing the frames in the new traversal order. As an alternative, the MPEG standard could be rearranged to store the macroblocks in a set order, but this does not allow the MPEG player to be optimized for all architectures.

Unconstrained MPEG

The first method, called *unconstrained MPEG*, does not make any assumptions about the MPEG stream and therefore works with any MPEG encoded video stream. By making no assumptions about the stream, each of the macroblocks within a frame can potentially start at any bit offset within the frame. Thus, in order to find the macroblock corresponding to the beginning of each scanline, each macroblock in the frame must be completely parsed. Without any markers in the middle of the frame, this is the only alternative as macroblocks have no defining feature to look for. Because finding the beginning of each line can be costly, the information that is parsed in the first pass is saved for the second pass. As a result, in the first pass we store the offset of the beginning of *each* macroblock within the frame along with the past dc components and its motion vectors. This step is also required because skipped macroblocks are grouped and encoded together, thus, in decompression they must be separated into individually skipped macroblocks. The advantage of using an unconstrained player is that any traversal algorithm can be implemented at the cost of a more expensive first pass, which includes traversals that begin in the middle of the frames. The main disadvantage of this approach is that the memory requirements for storing the past dc components and motion vectors may become prohibitive to improving processor caching.

Constrained MPEG

The second method, called *constrained MPEG*, constrains the MPEG video to have a slice defined per line, which allows fast access to the beginning of each line in each of the frames. By inserting slice headers at the beginning of each line, efficiently searching for the beginning macroblock on each line is possible because each slice header introduces a byte aligned marker into the stream. In this algorithm, the first

B-Frame macroblock type	Total of type	Forward MB dependencies	Backward MB dependencies
I	171		
P	2117	7211	
B	3499		9958
Bidirectional	4203	14342	14457
Skipped	130		
Totals	10120	21553	24415

45968 total dependent MB accesses

Figure 5.4: Distribution of B-frame Macroblocks and Their Dependencies. This table shows the distribution of the B-frame macroblock types and the distribution (either forward for P macroblocks, and backward for B macroblock) of macroblock accesses to dependent frames. As shown by the table, the distribution of forward and backward accesses was fairly evenly distributed. It is interesting to note that each bidirectionally encoded macroblock accessed an average of 6.85 macroblock accesses *key* frames.

pass, then consists of finding all the slice header positions within the frame. Because these headers are always byte aligned, finding the offsets to the beginning of scan lines requires very little overhead. Very little additional memory is required to store the offsets of these slices. In the second step, the horizontal and vertical striping algorithms may be applied to decompress the frames of the GOP. The addition of slice headers per line compared to a slice per frame results in approximately a 7 percent increase in the size of the video clip. The additional slice header per line, however, allows the MPEG video to be more robust in the presence of transmission errors.

5.3.3 Evaluation of Algorithms

To compare these algorithms, we encoded two random clips from Walt Disney's *Honey, I Blew Up the Kid* using the Berkeley MPEG encoder. The videos contained frames that were 640x368 pixels in size and were encoded with one I-frame between every 11 B-frames. We then took a single GOP of the video (2 I-frames and 11 B-frames) to run the simulations on. These videos then resulted in over 30 million data references to decompress the GOP. Because both the cache simulations and run-times resulted in nearly identical results, we present only one of the clip's results here. The basis of our MPEG player was the Berkeley MPEG player version 2.0. The breakdown of how the 11 B-frames were encoded are shown in Table 5.4. It is interesting to note that 41% of the B-frame macroblocks were bidirectionally encoded, each accessing an average of 6.85 dependent macroblocks. In addition, for each B-frame macroblock that was forward (P) and backward (B) encoded, 3.4 and 2.8 dependent macroblock accesses were made on average, respectively.

To determine what the overhead of implementing a two pass algorithm using different macroblock traversals, we then modified the standard Berkeley MPEG player to perform the various decompression traversals. It is important to note that we made no attempt to optimize the old or new code in terms of the reconstruction of blocks and the inverse DCT operations. We simply added the traversal algorithms and optimized only the traversal algorithms. We performed our measurements using a 99MHz Hewlett-Packard 9000/735 computer with a 64 Kbyte first level cache and 256 Kbyte second-level cache. To obtain the cache measurements and instruction counts, we used the Ultrix-based tool *pixie*, which can generate the data addresses and instruction addresses for the execution of a program. To simulate various cache sizes, we used the *cheetah* cache simulator to obtain the cache miss ratios [79]. We used the same sample clip to generate the address traces and the performance statistics.

Baseline Decompression

To show the amount of additional overhead involved with using a 2-pass MPEG player, we first ran the *constrained* and *unconstrained* players in a scan line at a time traversal. That is, the same traversal that the original MPEG player uses. We recorded the performance in frames per second of each of the algorithms and also simulated their cache usage. The performance statistics are listed in Table 5.5.

	frames/sec (fps)	instruction count (millions)
original	9.47	100.0
constrained	9.43	100.7
unconstrained	9.00	105.8

Figure 5.5: Baseline Decompression Statistics. This table shows statistics obtained from running the sample clip on the *constrained* and *unconstrained* players using a scanline order exactly the same as the original MPEG player. The fps measurements were obtained by running the particular MPEG player 12 times on the MPEG clip, throwing out the low and high, and then averaging the rest. (Note: the frame rates were typically within 0.03 seconds of each other)

The most obvious number in Table 5.5 is the large number of extra instructions (5.8 million) required to parse a non-slice per line video over the standard MPEG player. The additional 6% overhead in instructions allows any traversal algorithm to be implemented instead of a scanline traversal. As expected, the performance of both 2-pass algorithms is worse than the original MPEG player because they are doing the same work as the original player with the additional overhead of passing through the data in the GOP two times. In the case of the *constrained* player, the amount of additional overhead to perform two passes is fairly small, and therefore without changing the MPEG standard, it is conceivable that the caching gain can outweigh the extra overhead of an additional pass. The constrained players, however, must implement traversal algorithms that decompress frames from the left to the right. As expected, the cache miss ratio is not significantly reduced or increased by using the same traversal algorithm as the original MPEG player. See Figure 5.6.

Vertical and Horizontal Striping

With vertical striping, we are able to obtain an increase in performance (in frames per second) with the vertically striped MPEG player. By simply changing the ordering of data accesses, the amount of caching improvement outweighs the need to parse the GOP twice for the *constrained* MPEG player. Please see Table 5.7. The

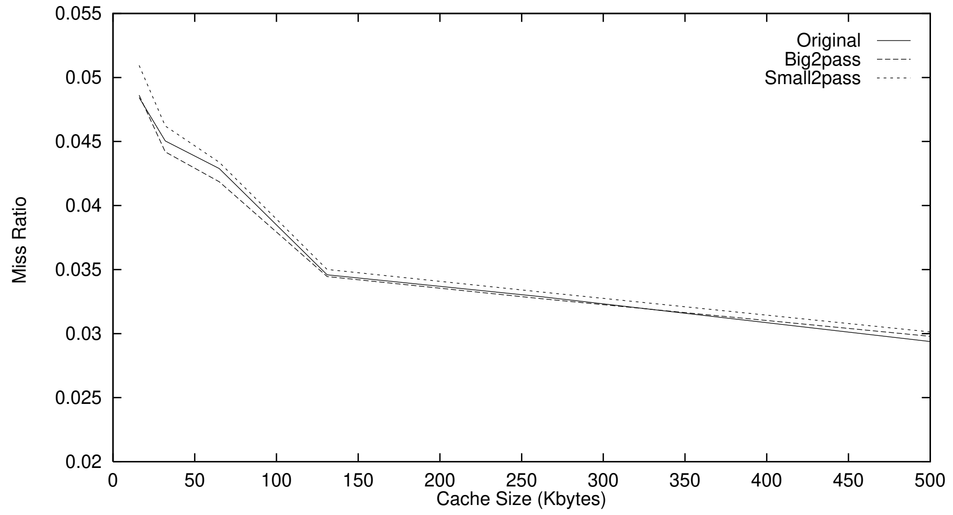


Figure 5.6: Scanline Miss Rates. In the graph, each line represents the cache miss ratio simulation for the different MPEG players using a scanline order exactly the same as the original MPEG player. These graphs were obtained by running *pixie* on the executable and then generating a data trace and processing it through the cache simulator *cheetah*. As expected, the cache miss ratios are not significantly different when using the same traversal orders.

	frames/sec (fps)	instruction count (millions)
original	9.47	100.0
constrained	9.47	100.5
unconstrained	9.35	106.7

Figure 5.7: Vertical Striping Statistics. This table shows the performance obtained from vertically striping the *constrained* and *unconstrained* players on the sample clip. These figures were obtained in the same manner as explained in Table 5.5.

unconstrained MPEG player, although improved over the scanline order, still suffers from a large amount of overhead occurred in the first pass. In addition, both players have additional “context-switching” times to alternate between the various scanlines within an image and between images.

As shown in Figure 5.8, the cache miss ratios for each of the new MPEG players are improved. For cache sizes of 64 to 128 Kbytes, an improvement of 7.5% to

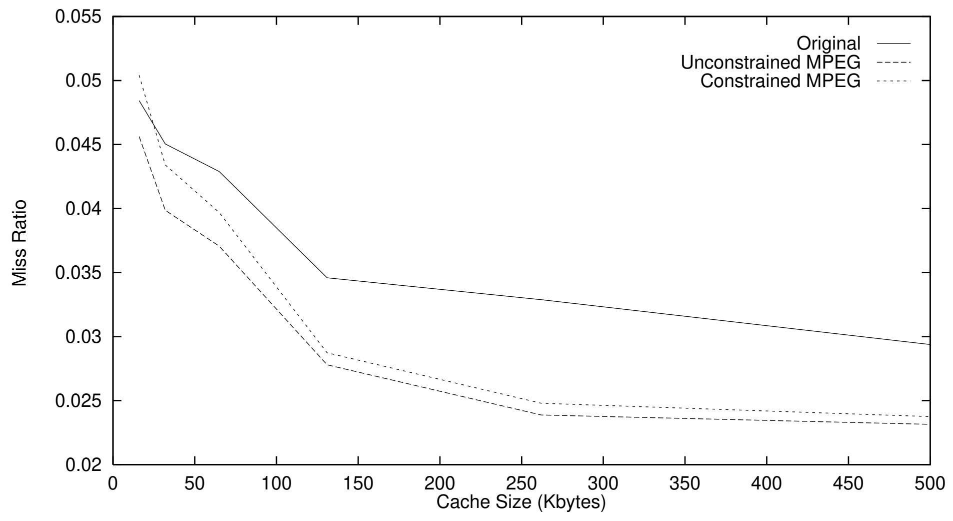


Figure 5.8: Vertical Striping Miss Rates. In the graph, each line represents the cache miss ratio simulation for the *constrained* and *unconstrained* vertically striped traversal. The original scanline MPEG player is also included for comparison. Each stripe in the traversal consisted of 8 macroblocks. As shown above, for cache sizes of 64 and 128 Kbytes, a decrease in cache misses of 7.5% and 19.6%, respectively, is obtained.

19.6% is obtained. For the HP 9000 architecture which has a 64 Kbyte primary data cache and a 256Kbyte level two data cache, a reduction of 7.5% in the miss ratio allowed us to gain a three tenths per second increase in performance of the player even though two passes were necessary. Thus, even minor changes in cache performance can give a reasonable return in overall performance.

For the horizontally striped MPEG players, the performance (in frames per second) is worse than the original MPEG player. For caches sizes between 64 and 128 Kbytes, the decrease in cache miss rates range from 10.7% to 20.4%. While these algorithms exhibit better miss ratios over the vertically striped traversals (See Figure 5.10), they are unable to overcome the time to perform "context-switches" between the different scanlines. That is, the overhead in changing between the different lines in the larger scanline is greater than the time gained from improved caching. Because the MPEG video standard encodes the macroblocks horizontally by

	frames/sec (fps)	instruction count (millions)
original	9.47	100.0
constrained	9.39	100.8
unconstrained	9.18	105.7

Figure 5.9: Horizontal Striping Statistics. This table shows the performance obtained from horizontally striping the *constrained* and *unconstrained* players on the sample clip. These figures were obtained in the same manners as explained in Table 5.5.

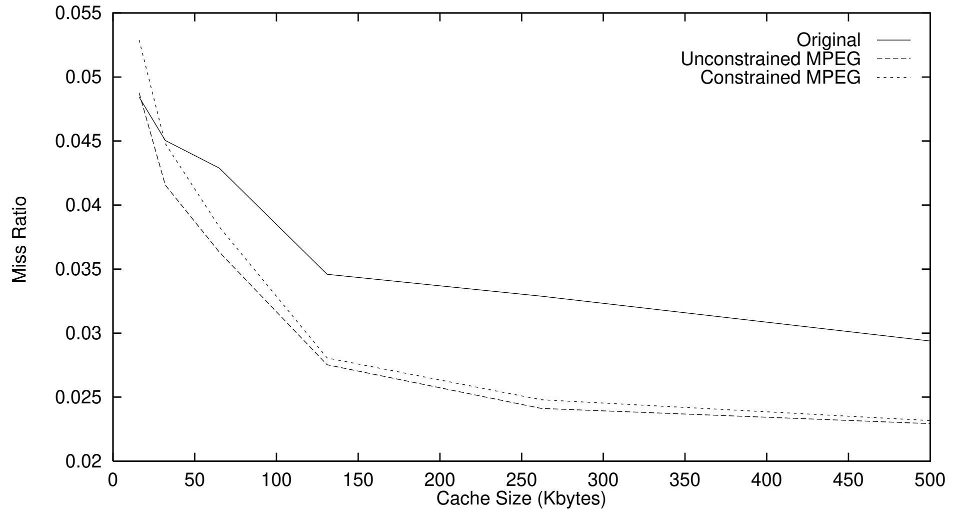


Figure 5.10: Horizontal Striping Miss Rates. In the graph, each line represents the cache miss ratio simulation for the *constrained* and *unconstrained* horizontally striped players. The original scanline MPEG player graph is shown for comparison purposes. Each stripe in the traversal consisted of 6 macroblocks. As shown above, for cache sizes of 64 and 128 Kbytes, a decrease in cache misses of 10.7% and 20.4%, respectively, was obtained.

scanline, applying an algorithm which makes many vertical macroblock “context-switches” does not pay off.

Our results also suggest that in systems that have both a primary and second level cache that the miss rates for the second level cache can be reduced as well. Using the unconstrained player and a secondary cache size of 256K (as found in the Intel P6), vertical striping reduces the secondary cache miss rate by 25%, while horizontal striping reduces it by 27%. The constrained player has approximately the same results. For systems like the Pentium P6, accessing the secondary cache does not incur as large a penalty as accessing main memory because the secondary cache is on chip. Thus, using vertical and horizontal striping may also help in other levels of the caching hierarchy.

A Look Into The Future

In future architectures, we expect that the data used in decompression to be consumed faster with split data paths. Because of this, the cache miss ratios presented may be deceiving because byte accesses that occur in MPEG result in a single word being hit multiple times. However, in future architectures with split data paths, these byte accesses may be combined into a single access causing the data to be consumed at a much faster rate with less hits.

To estimate what cache miss ratios may be in the future, we modified the original MPEG player and the *constrained* MPEG player from the last section and simulated the operations which could be performed in parallel in future architectures with split data paths. By running these modified MPEG players through *pixie* and *cheetah*, we were able to *estimate* the miss ratios for split data path computers.

As shown in Figure 5.11, the cache miss ratio for the original MPEG player can be expected to rise anywhere from 40 to 50% for cache sizes of 64 to 128 Kbytes. The vertical stripe traversal for the *constrained* algorithm has a rise in cache miss ratio from 30 to 40% for cache sizes of 64 to 128 Kbytes. Just as important, the difference in cache misses for the original MPEG players and the *constrained* vertical striped traversal algorithm are magnified under a split data path architectures, emphasizing the importance of good cache behavior in future architectures.

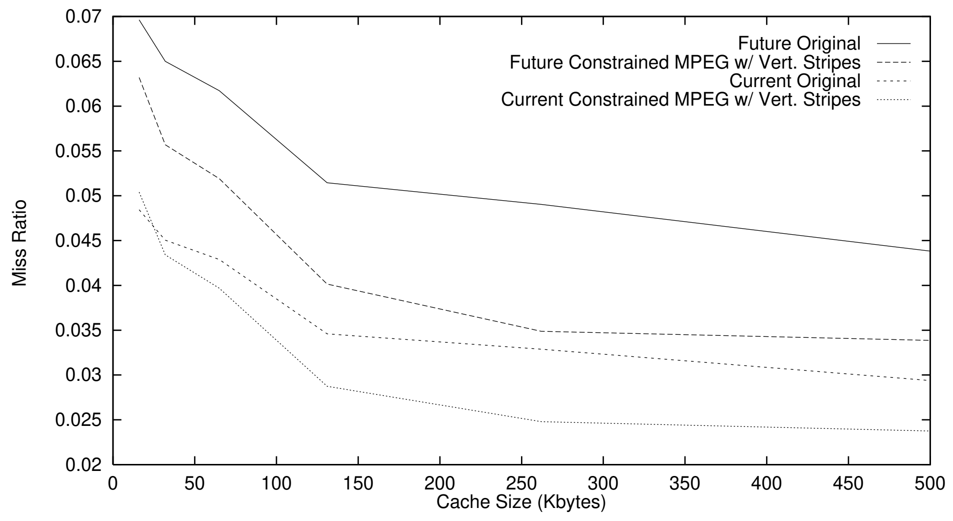


Figure 5.11: Future Architecture Miss Rates. This graph shows the estimated increase in miss rates for architectures that support split data paths with parallel operations on its subwords. In future architectures, the cache misses is estimated to increase from 30% to 40% for cache sizes of 64 and 128 Kbytes.

5.3.4 Working-Set Summary

In this section, we have described two methods to improve processor caching for decoding MPEG video in software. Decompressing an MPEG video clip in a scan-line order as the standard implies results in exploiting coherence in video streams in only one dimension. By increasing the coherence within a frame and between frames, a decrease in miss ratios is easily obtainable.

For *constrained* videos that have slices defined per line, the overhead of parsing the slice headers is small enough that an increase in processor caching can outweigh a first pass to find all slice headers. The *unconstrained* MPEG player suffers mainly from having to parse the movie bit-by-bit twice. On the machines we sampled, lower cache miss rates could not compensate for the increase of over 5 million instructions. With newer processor architectures, however, the cost of a cache miss may become large enough such that the gain in processor caching can outweigh the approximately 5% instruction increase from the first pass. Given that the unconstrained MPEG player finds *all* macroblock addresses, parallelizing the new traversal algorithms becomes possible with each processor working on a smaller area of the entire video. Parallelization of the *constrained* MPEG player is feasible but has to be parallelized in thin strips causing the additional caching performance between lines to be diminished.

Vertically striping the traversal algorithm tends to provide the best performance because the "context-switches" between different scanlines is minimized. Horizontal striping suffers from having to perform many "context-switches" in order to keep dependent blocks within the cache.

5.4 Improving Processor Caching via Prefetching

Re-ordering the decompression traversal of macroblocks can reduce the working-set size so that the cache misses are reduced, however, re-ordering the traversal algorithms can result in additional overhead in both the number of instructions executed as well as the amount of memory required. The alternative to this re-ordering is to anticipate the data accesses and to prefetch the data out of slower secondary memory before it is needed. For prefetching to be effective, we must be able to 1) predict probable sequences of memory references, 2) accurately predict the addresses accessed by those references, and 3) predict these values far enough in advance so that memory bus contention does not increase. For MPEG video decompression, the reconstruction of macroblocks exhibits these three characteristics. In particular, in the reconstruction of macroblocks, the data is accessed in a regular well-known pattern, making MPEG video decompression a good candidate for prefetching of data.

In this section, we investigate the affects that architectural support for prefetching can have in the caching performance of software MPEG video decompression. The goal of this research is not to determine the best cache sizes and policies for MPEG video streams. We are, however, interested in showing how a prefetching instruction can be beneficial in software video decompression. As a result, our experiments are aimed at showing the applicability of prefetching for video decompression and not at the overall improvement in performance speed.

In the rest of this section, we describe both hardware and software mechanisms for prefetching that have been introduced in the literature. We then present our experimental results on how MPEG video decompression in software can be impacted by having a prefetch instruction in software.

5.4.1 Data Prefetching

The prefetching of data from main memory into the processor cache hierarchy can be either hardware or software-controlled. Hardware-controlled prefetching techniques typically involve prefetching data based on either an access or miss for data in the cache. As an example, on a cache miss, the prefetch hardware, in addition to fetching the line that missed in the cache, may fetch the following line as well in anticipation of a sequential access. Software-controlled techniques require hardware support for executing the prefetch instruction in addition to either compiler inserted or user inserted prefetch instructions into the code itself.

One of the earliest proposed uses of hardware-controlled prefetching was introduced by A.J. Smith [76]. In his research, Smith identified three possible techniques for prefetching: (1) always prefetch, (2) prefetch on misses, and (3) tagged prefetches. On an access to cache line *i*, the *always prefetch* technique always prefetches the next cache line, *i+1*. The *prefetch on misses* technique prefetches cache line *i+1* only if the access to cache line *i* missed the cache. Finally, the *tagged prefetch* technique is an extension of prefetch on misses, except the prefetch of cache line *i+1* is delayed until cache line *i* is accessed again, increasing the probability that cache line *i+1* will be needed. In related work, Jouppi extended Smith's work to prefetch data into FIFO stream buffers that are separate from the cache hierarchy [46]. On a cache miss, these FIFO queues are then filled sequentially from the missed cache line. Jouppi found that this technique is much more effective at removing instruction references than data references because instructions tend to be accessed more sequentially than data.

Hardware techniques that base prefetching on data cache misses have limited application because of irregular patterns of access. To increase the accuracy of prefetching, additional hardware in the form of a table must be used. Fu and Patel have shown data prediction tables to be useful for vector applications, where data may be accessed in non-unit stride lengths [33]. Data prediction tables have also been shown effective at limiting unnecessary prefetching [3]. The use of prediction tables for video applications has also been proposed. Zucker, Flynn, and Lee focus their work on the comparison of hardware prefetching techniques for multimedia applications [86]. In their work, they propose the use of a *stride prediction table* (SPT) for predicting data accesses to main memory. Using the stride prediction table of 128 entries and with larger sized cache, the SPT mechanism can remove 70 to 90% of misses that would have otherwise occurred with the same size cache and associativity. Their work does not address the potential problem that the memory bandwidth bottleneck may have on the amount of prefetched data that can be retrieved.

The alternative to hardware-controlled prefetching is to add support for explicit prefetching operations, such as loads with no target register. These instructions can be added by the compiler or, in the case of important application code, by hand. In the present work, we do not address how these instructions are introduced into the MPEG decoder implementation. Porterfield showed that software-controlled prefetching can be more effective than the simpler hardware mechanisms [65]. Klaiber and Levy have also showed that software-controlled prefetching can be useful in reducing the cost of memory references by using a software-controlled prefetching into a separate *prefetch buffer* [50].

5.4.2 Prefetching and MPEG Video Decompression

Among the steps in the decompression of a single 16x16 pixel macroblock are the inverse DCT on each block within the macroblock and the reconstruction of the blocks. The inverse DCT of each block within a macroblock requires a fairly small amount of data, exhibiting a very low miss rate in even a small cache. On the other hand, the reconstruction of the macroblocks requires a large amount of data to be accessed, resulting in high miss rates. The reconstruction of dependent B frame macroblocks, however, accesses data in a regular pattern once the motion-vectors have been decoded. Thus, the accesses that are made in the reconstruction of macroblocks are key in reducing the memory access times.

For the blocks that are dependent on other key frames, accesses to the key frame data follow a very regular pattern. Typically, each block accesses 8 bytes from the key frames for each line within the block. Two possibilities exist for prefetching of this data. The first method, *aggressive* prefetching assumes that the 8 bytes accessed can cross the cache line boundaries, hence, the data for two cache lines are prefetched from memory. As a result, *aggressive* prefetching brings in a potentially large amount of data that may not be used. The second method, *conservative* prefetching assumes that the data is completely on the line that is prefetched. While *conservative* prefetching only fetches the line that is going to be accessed, a possibility exists that the data needed crosses the cache line boundaries, and that some necessary data will not have been prefetched.

5.4.3 Architectural Model

The implementation of a prefetch instruction in hardware can be accomplished in many different ways. Our baseline system is a load/store architecture with a two-level cache subsystem similar to the Intel Pentium P6 [42]. The Intel P6 chip consists of an 8 kbyte, 2-way set-associative, 32 byte per line primary data cache and an 8 kbyte, 2-way set-associative, 32 byte per line primary instruction cache. These

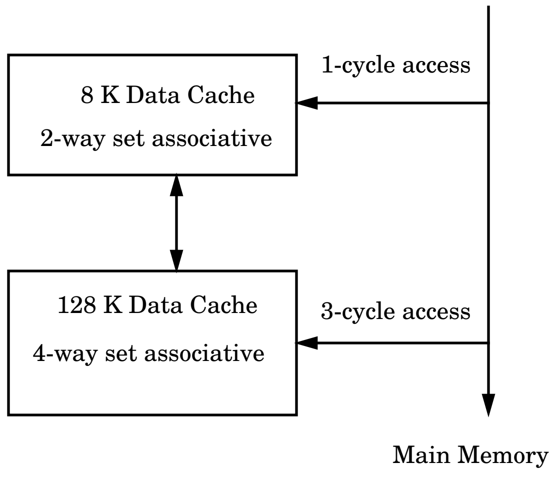


Figure 5.12: Simulated Data Cache Hierarchy. This figure shows the Pentium P6 based data cache hierarchy that we assume for our measurements. The prefetch buffer and logic is separate from the data cache and is accessed on a cache miss for prefetched data.

caches are backed by a unified secondary cache that is a 256 kbyte, 4-way set-associative, 32-byte-per-line cache. In order to simplify the analysis of our data, we assume for our model that the secondary cache is a split instruction and data cache, each of size 128 kbytes. Figure 5.12 shows the architecture that we are assuming.

To access the cache hierarchy, we assume that a cache hit in the primary cache takes 1 cycle, while a miss in the primary cache with a hit in the secondary cache requires 3 cycles as in the Pentium P6 chip. We also assume that there is an 8-entry write buffer. We assume that the write buffer asynchronously writes data back to main memory. In the event that the write buffer is full, we assume that the processor stalls until an entry in the write buffer is freed, thus, in times when many writes are outstanding, the write buffer appears to be synchronous.

5.4.4 Experimental Set-up

To study the possible effectiveness of prefetching for MPEG video decompression we captured randomly chosen scenes from the Walt Disney Movie *Honey, I Blew Up the Kid* (selected on the basis of its low rental price) and MPEG encoded the videos using the Berkeley MPEG encoder. The videos contained frames that were 640x368 pixels in size and were encoded with one I-frame between every 11 B-frames. We then took a 13-frame sequence from the video (2 I-frames and 11 B-frames for each sequence) to run the simulations on. (A small number of other sequences have produced very similar simulation results.)

The Berkeley MPEG player (version 1.2) compiled for a DEC Alpha AXP was used as the software MPEG video decompression implementation. We used the address generation tool ATOM to obtain the data addresses for the software decompressing the sample clips [20,77]. ATOM allows for the generation and analysis of data addresses concurrently, removing the need to store large address traces on disk. Using ATOM, we instrumented the MPEG player code and analyzed the data references that were made for the decompression of the sample clips. For our simulations, we have removed the initialization code from the miss rate statistics in order to obtain a more accurate evaluation of the steady-state performance of the MPEG player. In addition, we ran the MPEG code with the "-no_display" option, which removes the contribution of displaying the video frames to the screen. We expect that the display of the video frames will benefit from prefetching as well because the uncompressed frame of video needs to be copied to the frame buffer or video RAM.

5.4.5 Experimentation

From our simulations, we were interested in determining several key factors in the performance of a software-controlled prefetching instruction. We are interested in the performance trade-offs of placing data in a prefetch buffer versus placing the data directly in the cache, and the effectiveness of prefetching (how much data that is prefetched is actually used). We are also interested in examining the main memory access time and whether or not enough memory bandwidth exists to satisfy the prefetch requests.

Primary Cache Miss Rate 2.565

	refs (millions)	% of all refs	%miss rate	% of all misses
I mblock reconstruction	3.3	7.29	1.53	4.35
P mblock reconstruction	3.8	8.31	2.68	8.70
B mblock reconstruction	5.6	12.14	2.73	12.93
Bi mblock reconstruction	8.2	17.84	4.43	30.79
Skipped mblock recon.	0.3	0.67	2.47	0.65
Inverse DCT	10.4	22.73	0.32	2.86
Parsing of recon. block	6.3	13.71	3.67	19.60
Other	7.9	17.31	2.98	20.12

Secondary Cache Miss Rate: 36.28

	refs to L2 (thousands)	% of all refs to L2	%miss rate	% of all L2 misses
I mblock reconstruction	51.1	4.35	46.98	5.63
P mblock reconstruction	102.3	8.70	57.92	13.89
B mblock reconstruction	152.0	12.93	57.73	20.57
Bi mblock reconstruction	362.0	30.79	54.02	45.84
Skipped mblock recon.	7.5	0.64	37.13	0.66
Inverse DCT	33.6	2.86	14.12	1.11
Parsing of recon. block	230.4	19.60	5.08	2.75
Other	236.5	20.12	17.2	9.54

Figure 5.13: Baseline Cache Statistics. This figure shows the cache statistics gathered from running our sample MPEG encoded video clip on an 8k, 2-way set associative primary cache and a 128k, 4-way set-associative secondary cache.

Baseline Decompression

Using ATOM, we generated the data reference misses that occur in the 8kbyte/128kbyte cache hierarchy for our sample clip and calculated the overall contribution towards the miss rate of the separate decompression steps. We did not generate the data references for the initialization code or for the dithering or X window code.

The results of these simulations for our cache hierarchy for the first scene are shown in Figure 5.13. For the 11B encoded video clip, the inverse DCT operations account for 22% of the data references generated during playback, but with a miss ratio of 0.32%, contribute only 3% of the total misses in the primary data cache. On the other hand, the reconstruction of macroblocks accounts for 46% of the references, and because of higher miss rates, contributes to 57% of the total primary data cache misses. At the secondary cache level, the reconstruction misses that account for 57% of the primary cache misses, miss the secondary cache more than 50% of the time! The misses in the secondary cache from the reconstruction of macroblocks contribute to 88% of the secondary cache misses. Thus, when a data reference for reconstruction misses the primary cache, it is very likely to also miss in the secondary cache. Note that the inverse DCT had a very low miss ratio even in the secondary cache. These results suggest two things. First, the working set of the inverse DCT is fairly small and as a result, exhibits a high cache hit rate. Second, the reconstruction of macroblocks that depend on other frames results in a high miss rate, mainly due to the volume of key data continually brought into the cache.

Prefetchability

One important factor in determining the usefulness of a compiler supported prefetch instruction is whether or not the data that is to be accessed can be determined in advance enough to start the prefetching. For the decompression of MPEG encoded video, the merging of data in dependent frames from key frames makes well known accesses once the motion-vectors to the key frames have been parsed. To test the prefetchability of the MPEG encoded video clips, we determined where in the MPEG decompression code the motion vectors were parsed for each of the various macroblock types. Once the motion vectors are determined, the data accesses for the reconstruction of the macroblock can then start to be prefetched. To find out how much time (in number of instructions) is available during the reconstruction of the macroblocks, we mark all the data that is expected to be used in the reconstruction

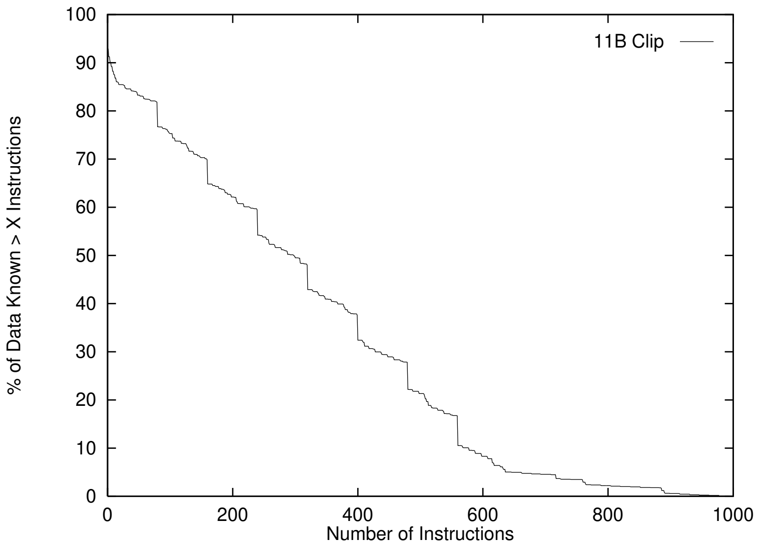


Figure 5.14: Prefetchability of MPEG video. This figure shows the percentage of data that will be accessed for reconstruction and the number of instructions the data to be accessed is known ahead of time.

(i.e. the data we expect to prefetch) and then start a timer. When the data is accessed, we then simply mark the time when it is first accessed.

Figure 5.14 shows the results of the prefetchability tests for the sample MPEG clip. As shown by this graph, the percentage of data that is available even 100 instructions before it is used is quite high. For example, 85% of the data accesses for reconstruction are known at least 20 cycles in advance, while 75% of the data accesses are known at least 100 cycles in advance. Another interesting trend on the graph is the stepping down of the prefetchability graph. These 8 steps essentially correspond to the 8 rows that are accessed for the reconstruction of the 8x8 blocks within a macroblock.

Effects of Prefetching on Secondary Cache Miss Rate

To show the effect that prefetching can have on the secondary cache miss rate, we have simulated our base cache hierarchy with various secondary cache sizes. In Figure 5.15, we have graphed the miss ratios for the secondary cache for the original MPEG decompression algorithm and for the MPEG algorithm using prefetching, both

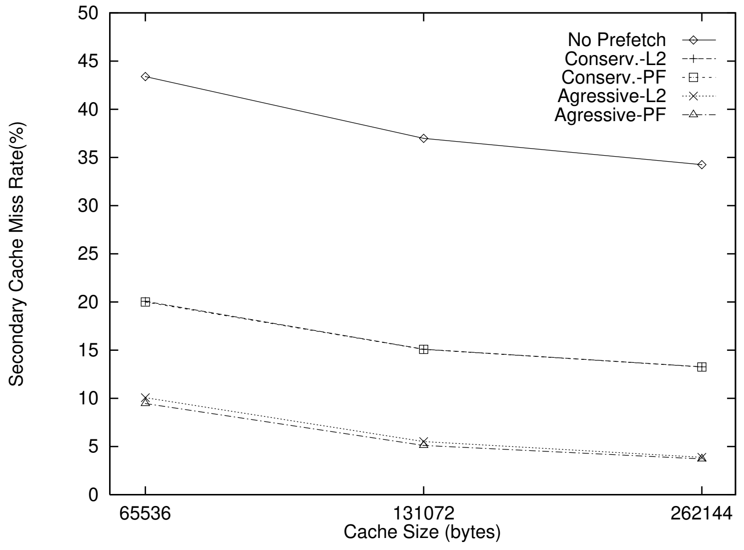


Figure 5.15: Cache Performance with Prefetching. This figure shows the miss ratios for conservative and aggressive prefetching for MPEG video decompression into either the level 2 cache (L2) directly or into a separate prefetch buffer.

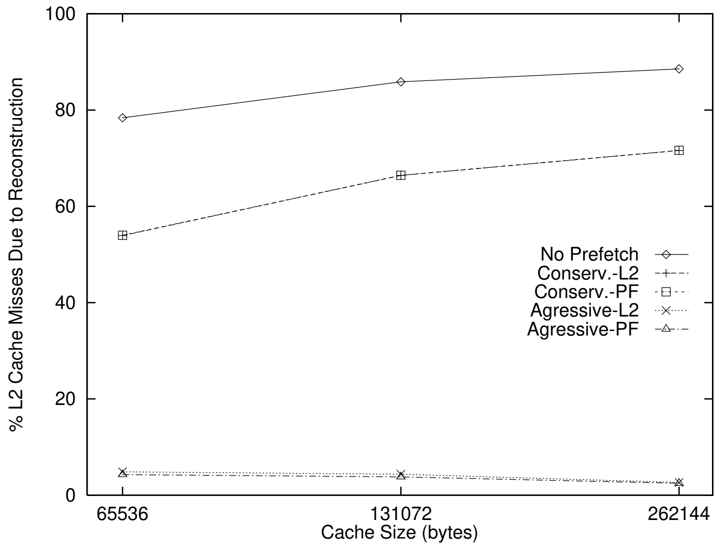


Figure 5.16: Cache Misses Due to Reconstruction of Macroblocks. This figure shows the percentage of secondary cache misses due to the reconstruction of macroblocks.

into the secondary cache and into a separate prefetch buffer. Figure 5.16 shows the percentage of secondary cache misses due to reconstruction of macroblocks with and without prefetching.

Several observations can be made from these graphs. First, prefetching directly into the secondary cache and prefetching into a separate buffer result in approximately the same cache miss ratios for the secondary cache. This suggests that prefetching directly into the secondary cache adds little additional cache pollution. Secondly, the miss ratio reduction across the various caches sizes is fairly constant. The conservative prefetching algorithm reduces the secondary cache miss ratio by approximately 55%, while the aggressive prefetching algorithm results in a reduction of over 80%. As Figure 5.15 shows, without prefetching, approximately 80% to 90% of the secondary cache misses are due to the data accesses during the reconstruction of macroblocks. With conservative prefetching, the number of cache misses is reduced on the order of 15 to 25%, while the reduction in cache misses with aggressive prefetching is much larger. As an example, with aggressive prefetching the reconstruction of macroblocks can be reduced from 86% to only 4.2% using our baseline 8 Kbyte/ 128 Kbyte cache hierarchy. This results in a 97% reduction in the reconstruction contribution towards the L2 Miss rate!

Aggressively prefetching data can remove a large percentage of main memory accesses; however, several concerns arise from aggressively prefetching data. First, prefetching can impose an overhead to load unused data in. Second, prefetching will not be effective if insufficient memory bandwidth is available.

Effects of Prefetching on Data Utilization

To measure the utilization of prefetched data, we tagged all the prefetched data that was brought into the cache hierarchy and measured the number of words within the cache line that were accessed before the tagged cache line was evicted from the cache. Using the different prefetching styles (aggressive and conservative) and the different areas to place prefetched data (in the secondary cache or into a separate buffer), we graphed the utilizations for differing cache sizes. The results are shown in Figure 5.17. As shown by the graph, all the different prefetching methods

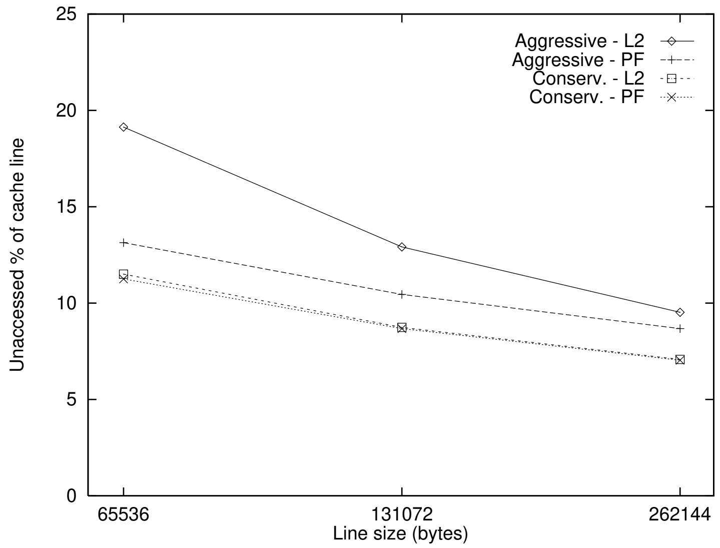


Figure 5.17: Utilization of Prefetched Data. This graph shows the utilization of cache lines for the various prefetching methods. The utilizations are based on the number of words accessed per cache-line with the utilization of a cache line being calculated when it is evicted from the cache hierarchy.

result in cache line utilizations greater than 80%. The conservative prefetching schemes perform better than their counterpart aggressive prefetching scheme. Conservative prefetching achieves higher utilizations because the prefetched data tends to pollute the cache less. Aggressive prefetching into the L2 cache benefits from a larger cache size. For small cache sizes, aggressively prefetching data tends to pollute the cache more than with larger cache sizes that have more room for the prefetched data. In addition, prefetching into a separate prefetch buffer results in higher utilizations than their counterpart algorithms that prefetch directly into the L2 cache. This result is not unexpected because having separate prefetching buffers increases the "effective" size of the cache. As the graph shows, both approaches achieve high utilizations.

In Figure 5.18, we have graphed the utilization of cache lines for different cache line sizes, keeping the secondary cache size constant at 128 kbytes. The larger cache lines result in lower utilizations because, with a constant cache size, more cache lines are evicted earlier. As expected, prefetching into a separate prefetch

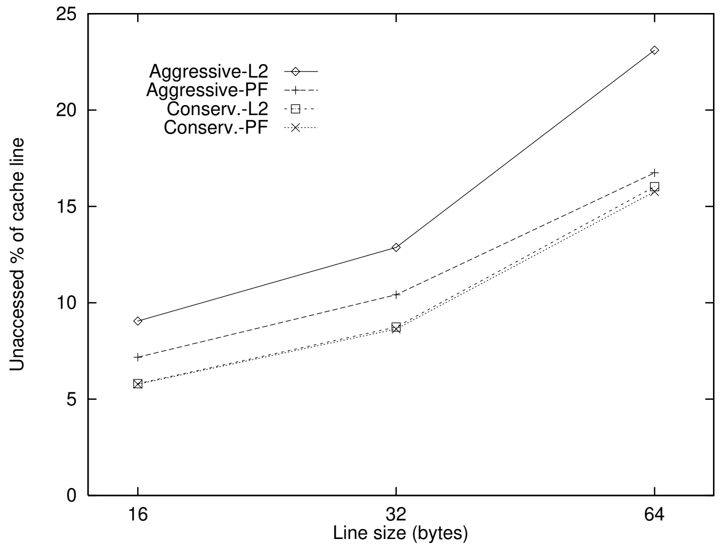


Figure 5.18: Utilization of Prefetched Data. This graph shows the utilization of cache lines for the various prefetching methods using a 128 kbyte secondary cache with varying line sizes. The utilizations are based on the number of words accessed per cache-line with the utilization of a cache line being calculated when it is evicted from the cache hierarchy.

buffer results in slightly higher cache utilizations because the “effective” size of the cache is increased. In addition, aggressive prefetching has a slightly lower cache line utilization because it pollutes the cache less. As the figure shows, the prefetching algorithms exhibit high cache line utilizations across the various cache line sizes.

Effects of Prefetching on Memory Bandwidth

Any application that has a large data consumption to instruction ratio may not have enough opportunity to prefetch effectively. To examine the memory bandwidth requirement, we have simulated the main memory bus and the reads, writes, and prefetches that are needed. Recall that for our simulations we have ignored the contribution of the instruction fetches. For our simulations, we use the cycles per instruction (CPI) due to data access penalty as found in previous cache work [10]. The $\mathrm{CPI}_{data\_access}$ can be defined as:

$$CPI_{data\_access} = \frac{\text{total data access penalty}}{\text{number of instructions executed}} .$$

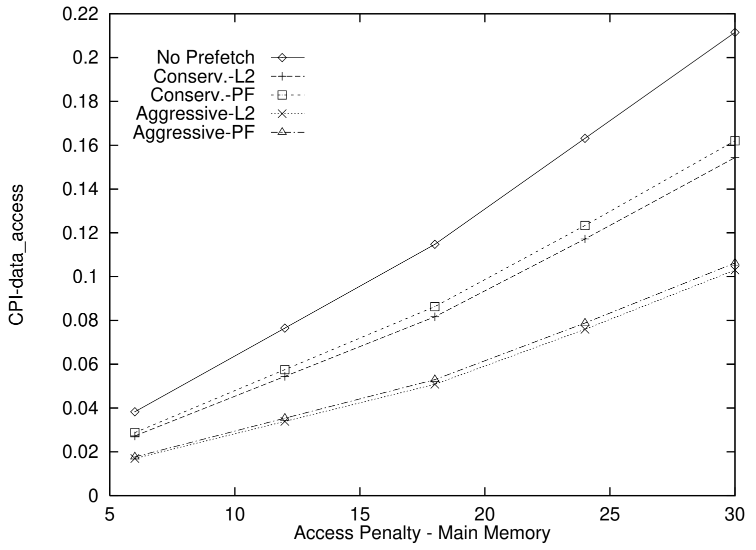


Figure 5.19: Prefetching Bandwidth. This figure shows the $\mathrm{CPI}_{\mathrm{data\_access}}$ for conservative and aggressive prefetching for MPEG video decompression into either the level 2 cache (L2) directly or into a separate prefetch buffer. The access penalties are in cycles. This simulation includes the stalls due to bus contention for data access, however instruction fetches were not simulated.

For our simulations, the number of instructions executed was very nearly constant. As a result, the graphs show the relative number of stall cycles that each of the various prefetching routines incurred. We then graphed the $\mathrm{CPI}_{\mathrm{data\_access}}$ for the original MPEG code for various main memory access penalties. In addition we graphed the *conservative* and *aggressive* prefetching algorithms for prefetching into both the secondary cache and a separate prefetch buffer.

As shown in Figure 5.19., the prefetching algorithms reduces the number of stall cycles during the reconstruction of the macroblocks. Several interesting observations can be made from the graph. First, because the data that is to be prefetched is known well ahead of time, the prefetching of data is always beneficial for the reconstruction of macroblocks. Second, the conservative $\mathrm{CPI}_{\mathrm{data\_access}}$ for prefetching into the secondary cache directly or prefetching into a separate prefetch buffer results in similar $\mathrm{CPI}_{\mathrm{data\_access}}$ penalties. The high utilization of data in the prefetch buffer tends to eliminate any distinction on levels of pollution. This suggests that prefetching directly into the secondary cache can be effective for MPEG decompression without the need for additional prefetch buffers. Finally, the aggressive prefetch algorithm results in lower CPI_{data_access} penalties, even though the aggressive algorithm prefetches more data than the conservative algorithm prefetches. The reason for this is the high probability that data accesses for the next macroblock make to the prefetched data for the current block. Because of this, aggressive prefetching has an initially high number of cycle stalls for the initial macroblock prefetches of each line. After this initially high number, however, the aggressive prefetching prefetches data for the next macroblock and not the current macroblock, resulting in data that has longer period of time between prefetch and data access. With the time between macroblocks having a large number of instructions between them, there is ample time between macroblocks to fetch data, resulting in very low CPI_{data_access} penalties. Figure 5.19 shows conservative prefetching either into the secondary cache directly or into a separate prefetch buffer reduces the CPI_{data_access} by roughly 25%. The aggressive prefetching results in a reduction in CPI_{data_access} of approximately 50%. These results suggest that enough memory bandwidth exists during the decompression of MPEG video to allow for effective software prefetching.

As previously mentioned, future architectures will provide support for video decompression by allowing multiple byte operations in a single word to be handled in one instruction. As a result, the memory bandwidth may become more of a critical issue in prefetching. For the reconstruction of macroblocks in MPEG, this results in the combining of data from key frames to occur more rapidly, placing a higher demand on the memory subsystem. As a result, prefetching may not be as feasible. We simulated the operations which could be performed in parallel in future architectures with split data paths, allowing us to *estimate* the potential rise in CPI_{data_access}. For all the algorithms, the expected rise in CPI_{data_access} is expected to be approximately 50% across all the algorithms, suggesting that prefetching in future processor may also result in low cache miss rates. See Figure 5.20.

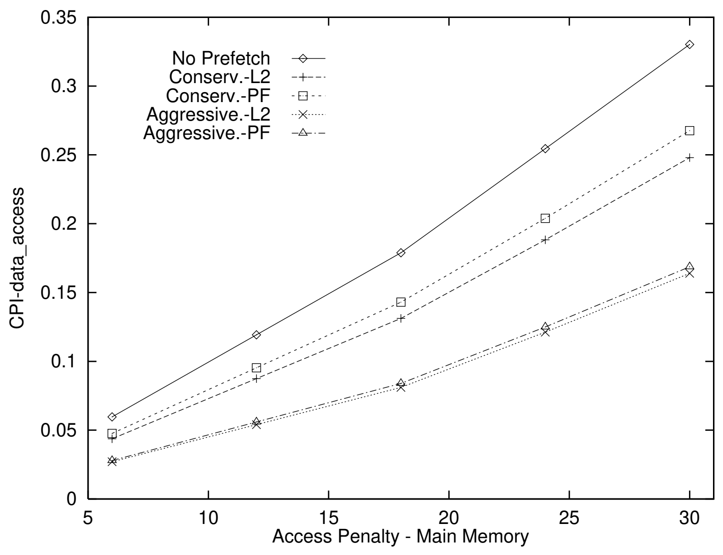


Figure 5.20: Prefetching Bandwidth in Future Processors. This figure shows the expected increase in the $\mathrm{CPI}_{\mathrm{data\_access}}$ for future processors with split data paths. The access penalties are in cycles. This simulation includes the stalls due to bus contention for data access, however, instruction fetches were not simulated

5.4.6 Software-Controlled Prefetching Summary

In this section, we have examined the utility of prefetching for the software decompression of MPEG encoded video. The software decompression of an MPEG encoded clip results in poor cache utilization for the reconstruction of macroblocks. By prefetching data accesses for the reconstruction of these macroblocks, the number of cache misses, and hence stalls due to main memory access, can be drastically reduced.

The effective use of prefetching requires that the prefetch algorithm knows when to initiate a prefetch and what data is to be accessed [76]. For MPEG encoded video, both of these criteria are met. Because the prefetching of data for the reconstruction of macroblocks occurs with different stride lengths, the use of a prefetch instruction at the programming level or compiler level can help in the optimization of the performance of MPEG decompression in software.

5.5 Summary

In this chapter, we have examined the effect that burstiness has on the decompression of MPEG encoded video on processor caches. Because the memory bottleneck from processor stall will be the limiting factor in the software decompression of MPEG video [64], minimizing the on-demand accesses to main memory is critical in future architectures. In order to reduce the data cache misses that occur, we have examined two approaches for reducing cache misses.

The first approach of reducing the working-set size through the re-ordering of macroblock traversals is a purely software approach to reducing data cache misses. While this method is flexible, it requires two passes to be performed; the first pass is required to find the offsets to the beginning of macroblock scanlines, while the second pass does the actual decompression. Using either *vertical* or *horizontal* striping of the traversal algorithms leads to better cache performance. In the vertical striping case, using a current generation Hewlett-Packard, we were even able to better performance, even though it had an overhead of 1.5% more instructions over the standard traversal algorithms. Horizontal striping suffers from the requirement of context switching between multiple scanlines.

The second approach of keeping the cache warm by prefetching data that will be accessed is a solution that requires both hardware and software support for prefetching. Prefetching is an attractive alternative because the overhead in issuing a prefetch instruction is not that large, but the benefits of prefetching can reduce the large number of stall cycles that are required on a required access to main memory.

As processor speeds and memory speeds continue to diverge, the timely retrieval of information from main memory will be the bottleneck of these processors. One of the key factors in the performance of application will be processor cache performance.

CHAPTER 6

CONCLUSIONS AND FUTURE WORK

"The best thing about the future is that it comes one day at a time" - Abraham Lincoln

6.1 Research Contributions

In this dissertation, we have focused on the problem of handling burstiness introduced by video compression standards from both an external and internal point of view. From an external point of view, our work has focused on the efficient handling of variable-bit-rate video for the delivery of stored video across networks. From an internal point of view, our work has focused on how to effectively decompress video.

For the external handling of video, the work presented here is (to our knowledge) the first of its kind. We have introduced the notion of *critical bandwidth allocation* for the delivery of compressed prerecorded video. Our first paper is the first paper that identifies the different network requirements of live and stored video applications [22]. The CBA algorithm produces bandwidth plans that are monotonically decreasing. For systems where retrievals all follow these monotonically decreasing bandwidth plans, admission control is made simpler. That is, the network manager need only ask - *Is there enough bandwidth to start the flow of data*? For limited buffer sizes, the CBA algorithm produces plans for the continuous playback of video that (1) have the fewest number of bandwidth increases, (2) have the smallest peak bandwidth requirements, and (3) have the largest minimum bandwidth requirements. This work has been modified by other institutions in similar algorithms. An effort at the University of Massachusetts creates bandwidth plans by always starting each run at the last run's critical point. For our CBA algorithm, this has the result of breaking the bandwidth increases into smaller steps, while the decreases in bandwidth are the same. These bandwidth plans have the same smallest peak bandwidth, but also minimizes the variability in bandwidths. Their plans, however, require many more bandwidth changes, requiring more interaction and overhead in the network. Because the range of bandwidth values is fixed by only two runs within the movie, it is important to limit the number of changes in the areas outside of these runs.

We have extended the critical bandwidth allocation technique and have introduced an *optimal bandwidth allocation* technique. The optimal bandwidth allocation algorithm, in addition to the three properties of the CBA algorithm, also minimizes the number of bandwidth changes required for the continuous playback of video. As a result, the OBA plans require very few changes in bandwidth for relatively small amounts of buffering.

The use of smoothing for the delivery of stored video, implies that the plan may be inflexible to change. We have introduced the *VCR-window* technique which allows for full-function VCR capabilities within a small locality without requiring a change in the bandwidth reservations that were made. In the event that access is required outside of the VCR-window, a renegotiation of network bandwidth will be required. We have shown how contingency channels can be used for accesses outside of the VCR-window. Because contingency channels cannot always be guaranteed, the VCR-window is useful in minimizing the number of occurrences when the contingency channel will be used.

For the display of compressed video, we have investigated how software MPEG decompression affects processor data caching. A lot of efforts have focused on optimizing the code in software with such techniques as combining byte operations and loop-unrolling. While these works improve the performance of the MPEG players, they are still limited by the memory bottleneck that occurs with software decompression. Our work considers the impact that decompression has at the architectural level and what changes are necessary to alleviate the memory bottleneck. We have introduced *horizontal* and *vertical striping* for the re-ordering of the MPEG macroblock traversal. By re-ordering the macroblock traversals, the effective “working-set” is reduced, allowing macroblock data from key frames to survive longer in the cache. This technique, however, requires additional instruction overhead to implement. For software-controlled prefetching, we have examined how software-controlled prefetching can reduce the number of stalls that are required during the decompression of MPEG video. We have shown that the processor miss rates can be reduced by approximately 80% and that enough memory bandwidth exists to support prefetching of the video data.

6.2 Future Directions

The work presented in this thesis is a first step towards the efficient implementation of video-on-demand playback systems. The CBA and OBA algorithms allow for high utilization of bandwidth plans. The use of these techniques in implementing an end-to-end system are still not completely understood. From the video server side, the bandwidth allocation plans are more constant than those delivered from a statistically multiplexed server. As a result, we expect that because the CBA and OBA plans have very constant bandwidth allocations that the current work in video-on-demand servers is applicable. One assumption that this work has made is that the buffer size for smoothing remains constant. For set-top-boxes, that consist of a dedicated disk, the bandwidth plans can be made based on the size of the disk. For systems that use the smoothing buffer for other purposes, more or less buffer may be available during the playback of video. Changing bandwidth plans in light of this may prove to be interesting.

The VCR-window and the use of the contingency channel are a step towards implementing a true interactive video-on-demand system, however, the interactivity aspect is still not completely understood. The size required for the VCR-window depends on the encoding of the video as well as the interactivity that users require. If most users have very minor adjustments to their bandwidth plans, then the VCR-window can filter many adjustments through buffering. The size requirement for the contingency channel is also dependent on the actual usage patterns in the video-on-demand system. These buffer requirements and contingency channel requirements probably will require an iterative refinement through implementation and experimentation.

The effects of software video decompression on hardware are still not fully understood. Because of the many interactions between the decompression code, the operating system, and the actual hardware, a more in-depth analysis of these dependencies may allow for decompression to be handled more efficiently. In our work, we have ignored the effects that other processes in the system have on cache pollution. For a single user system, the operating system interference will probably be minimized due to the user devoting his or her attention to the video being played back. For systems that must handle video in addition to other tasks, the context switching times and the effects this has on cache pollution may require a more in-depth analysis. Finally, we have examined the impact that MPEG video decompression has on processor caching. As video compression standards continue to evolve, the impact that the standards have on the architecture may change.

APPENDICES

APPENDIX A

PSEUDO-CODE FOR BANDWIDTH SMOOTHING ALGORITHMS

This appendix contains the pseudo-code for the various bandwidth smoothing algorithms presented in this dissertation. The implementation and optimization details have been omitted.

Average Allocation Algorithm

```
num_groups = num_frames/group_size;
for(i=0; i < num_groups; i++) {
    group_ave = average of frames in group i;
    allocation[i] = group_ave;
    }
```

Sliding Window Smoothing

Let `average(x,y,extra)` be the average of all frames in the groups x to y (inclusive) subtracting "extra" bytes from the total sum (before the calculation of the average). Then, the sliding window smoothing pseudo-code can be given as:

```
 1   extra = 0;
 2   for(i=0; i < num_groups; i++) {
 3      max_average = 0;
 4      group_ave = average(i , i, extra)
 5      max_average = group_ave;
 6
 7      /**** Find maximum average within window ****/
 8      for(j=1; j < groups_in_win ;j++) {
 9         ave = average(i , i+j, extra)
10         if (ave > max_average)
11            max_average = ave;
12         }
13
14      /**** set bandwidth for group to max_average ****/
15      allocation[i] = max_average;
16      extra = group_size*(max_average-group_ave);
17      }
```

Starting at the beginning of the *i*th group, for a window size of *N* groups, the sliding window smoothing algorithm calculates the average bandwidth allocation for the first group within the window (lines 3-4). In this case, this is just the average frame size for the *i*th group. It then compares this with the average of the next group, the next 2 groups, up to the next *N-1* groups (lines 5-10). The allocation for the group is then set to the maximum average found within the window, effectively prefetching large bursts of frames (line 12). Line 13 calculates the extra data that will be prefetched as a result of this increased bandwidth allocation and is then applied to the next window of groups.

Critical Bandwidth Allocation

```
 1   max_ave(start,stop)
 2   {
 3      Find maximum average from start to stop
 4      Set c_bw to the maximum average and c_pt
 5         the point that caused this average
 6   }
 7
 8   CALCULATE_CBA()
 9   {
10      i = 0;
11      while (i <= end) {
12         max_ave(i,end);
13         set bandwidth from i to c_pt to c_bw;
14         i = c_pt + 1;
15         }
16   }
```

The critical bandwidth allocation algorithm uses the subroutine *max_ave* in lies 2 through 6 to find the maximum running frame average from the starting point that was passed in to the end of the movie. Thus, the critical bandwidth allocation algorithm consists of

Critical Bandwidth Allocation with Maximum Buffer Constraint

```
int doONE_CBA(start,stop)
{
    Find maximum average from start to stop
        such that buffer does not underflow or overflow
    Set c_bw to the maximum average and c_pt
        the point that caused this average
    Set point_reached to the point that using
        c_bw and C_pt will reach
        (point_reached is the end of the frontier)
    Set next_type to INCREASE, END, or DECREASE
        depending on whether the next run will require
        an increase or decrease. Set to END if stop
        is reached.
}
CALCULATE_CBA()
{
    i = 0;
    while (i <= end) {
        nt = doONE_CBA(i,end);
        run_end = c_pt;
        if (nt == INCREASE) {
            search between c_pt and point_reached
                (using doONE_CBA) to find starting point
                for next run such that the point reached
                in the next run is maximized.
            set run_end = starting point
            }
        set bandwidth from i to run_end to c_bw;
        i = run_end + 1;
        }
}
```

Optimal Bandwidth Allocation with Maximum Buffer Constraint

```
int doONE_CBA(start,stop)
{
   Find maximum average from start to stop
      such that buffer does not underflow or overflow
   Set c_bw to the maximum average and c_pt
      the point that caused this average
   Set point_reached to the point that using
      c_bw and C_pt will reach
      (point_reached is the end of the frontier)
   Set next_type to INCREASE, END, or DECREASE
      depending on whether the next run will require
      an increase or decrease. Set to END if stop
      is reached.
}

CALCULATE_CBA()
{
   i = 0;
   while (i <= end) {
      nt = doONE_CBA(i,end);
      run_end = c_pt;
      if (nt != END) {
         search between c_pt and point_reached
            (using doONE_CBA) to find starting point
            for next run such that the point reached
            in the next run is maximized.
         set run_end = starting point
         }
      set bandwidth from i to run_end to c_bw;
      i = run_end + 1;
      }
}

```

APPENDIX B

THEOREMS

Theorem 1 : *The critical bandwidth allocation algorithm with no buffer limitation results in a strictly decreasing sequence of bandwidth allocations*

Proof: To prove this theorem, we start by proving that the critical bandwidth following the first critical point CP_0 must have a bandwidth CB less than the first critical bandwidth CB_0. To see that the run following CP_0 must have a critical bandwidth less than CP_0, assume that this run has a critical bandwidth CB determined by a critical point CP. By the definition of critical bandwidth, CB_O must be greater than the average frame size from frame 1 to frame CP. This requires the following inequality to be true

$$CB_0 > \frac{(CP_0 \cdot CB_0) + (CP - CP_0)CB}{CP}$$

That is, CB_0 must be greater than the average of all frames from 0 to CP by definition. We then rewrite CP_0 as [CP - (CP - CP_0)] and substitute it in the first term, leaving

$$CB_0 > \frac{(CP \cdot CB_0 - (CP - CP_0)CB_0) + (CP - CP_0)CB}{CP}$$

Then by rearrangement, this requires

$$CB_0 > CB_0 + \frac{(CP - CP_0)(CB - CB_0)}{CP}$$

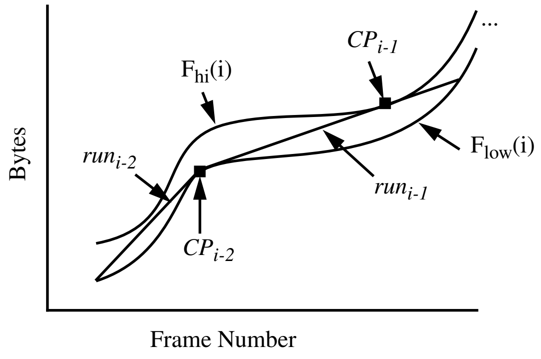
to hold. Since CP is greater than CP_0, CB must be less than CB_0.

By recursively re-applying this to the remaining portion after CP_0, we see that the critical bandwidth algorithm must result in a monotonically decreasing sequence of bandwidth requirements.//

Theorem 2 : *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 with (1) the smallest number of bandwidth increases possible, (2) the minimum peak bandwidth requirement, and (3) the largest minimum bandwidth required.*

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 must have (1) 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 previous runs 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 $i\text{-}1$ must have decreased the bandwidth over run $i\text{-}2$. This implies that run $i\text{-}2$ was determined by a critical point on $F_{low}(i)$ and that run $i\text{-}1$ starts on $F_{low}(i)$. In addition, the critical point for run $i\text{-}1$ must be on $F_{hi}(i)$. This situation is shown in the figure below:.



We now note that because run $i\text{-}1$ connect $F_{low}(i)$ and $F_{hi}(i)$, *any other* bandwidth plan not co-linear with run $i\text{-}1$ must have a run that has a slope less than that chosen by run $i\text{-}1$. 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 $i\text{-}1$. 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) because run *i-1* connects $F_{low}(i)$ and $F_{hi}(i)$. 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 for run *i-1*. Thus, any other bandwidth plan must also increase the bandwidth requirement before crossing the frontier of run *i-1* (see the figure on page 153). 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 from Chapter 3, we note that in 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 connect $F_{hi}(i)$ and $F_{low}(i)$, *any other* bandwidth plan no 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. //

Theorem 3 : *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 optimal critical bandwidth algorithm results in (1) 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
[*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. Let the [*inc*, *dec*] run be run *i*. By Property 2 from Chapter 3, 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. Let the [*dec*, *inc*] run be run *i*. By Property 3 from Chapter 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 his 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, will result 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. Any bandwidth higher results 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 will maximize 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 F_{hi} and F_{low} 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.//

BIBLIOGRAPHY

[1] D. Anderson, Y. Osawa, R. Govindan, “A File System for Continuous Media”, *ACM Transactions on Computer Systems*, Vol. 10, No. 4, Nov, 1992, pp. 311-337.

[2] C.M. Aras, J.F. Kurose, D.S. Reeves, H. Schulzrinne, “Real-time Communication in Packet Switched Networks”, *Proceedings of the IEEE*, Vol. 82, No. 1, pp. 122-139, Jan. 1994.

[3] J.L. Baer, T.F. Chen,“An Effective On-Chip Preloading Scheme to Reduce Data Access Penalty”, In Proceedings of *Supercomputing ‘91*, November, 1991, pp. 176-186.

[4] A. Banerjea, E.W. Knightly, F.L. Templin, H. Zhang,“Experiments with the Tenet Real-Time Protocol Suite on the Sequoia 2000 Wide Area Network”, In Proceedings of *ACM Multimedia 1994*, San Francisco, CA, Oct. 1994, pp. 183-191.

[5] V.Bhaskaran, K.Konstantinides,“Real-Time MPEG -1 Software Decoding on HP Workstations”, In Proceedings of *Digital Video Compression: Algorithms and Technologies 1995*, San Jose, CA, Feb. 1995.

[6] H. Bheda, P. Srinivasan,“A High-Performance Cross-Platform MPEG Decoder”, In Proceedings of *SPIE Digital Video Compression on Personal Computers: Algorithms and Technologies*, San Jose, CA, Feb. 1994, pp. 241-248.

[7] C-Cube Microsystems, “Designing JPEG Video Systems with the C-Cube CL550”, C-Cube Microsystems, Milpitas, CA, 1991.

[8] CCITT Recommendation MPEG-1, “Coded Representation of Picture, Audio, and Multimedia/Hypermedia Information,” ISO/IEC 11172, Geneve Switzerland, 1993.

[9] M.S. Chen, D.D. Kandlur, P.S. Yu,“Support for Fully Interactive Playout in a Disk-Array-Based Video Server”, In Proceedings of *ACM Multimedia 1994*, San Francisco, CA, Oct. 1994, pp. 391-398.

[10] T.F. Chen, J.L. Baer,“Reducing Memory Latency via Non-blocking and Prefetching Caches”, In Proceedings of *the Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, Boston, Massachusetts, Oct. 1992, pp. 51-61.

[11] D.M. Cohen, D.P. Heyman,“A Simulation Study of Video Teleconferencing Traffic in ATM Networks”, In Proceedings of *IEEE INFOCOM 1993*, pp. 894-901.

[12] A. Dan, D. Sitaram, P. Shahabuddin,“Scheduling Policies for an On-Demand Video Server with Batching”, In Proceedings of *ACM Multimedia 1994*, San Francisco, CA, Oct. 1994, pp. 15-23.

[13] A. Dan, P. Shahabuddin, D. Sitaram, D. Towsley,“Channel Allocation under Batching and VCR Control in Movie-On-Demand Servers”, *IBM Research Report RC19588*, Yorktown Heights, NY, 1994.

[14] M. Degermark, T. Kohler, S. Pink and O. Schelen, “Advance Reservations for Predictive Service”, In Proceedings of *5th Intl. Workshop on Network and Operating System Support for Digital Audio and Video*, Durham, New Hampshire, April 18-21, 1995, pp. 3-14.

[15] J.K. Dey, C.S. Shih, M. Kumar,“Storage Subsystem in a Large Multimedia Server for High-Speed Network Environments,” In Proceedings of *IS&T/SPIE Symposium on Electronic Imaging Science and Technology*, San Jose, CA, Feb. 1994.

[16] J. Dey-Sircar, J. Salehi, J. Kurose, D. Towsley,“Providing VCR Capabilities in Large-Scale Video Servers”, In Proceedings of *ACM Multimedia 1994*, San Francisco, CA, Oct. 1994, pp. 25-32.

[17] Digital Equipment Corporation, Digital Equipment Specification Sheet, Feb. 1995.

[18] S.H. Early, A. Kuzma, E. Dorsey,“The VideoPhone 2500 - Video Telephony on the Public Switched Telephone Network”, AT&T Technical Journal, Jan/Feb. 1993, pp. 22-32.

[19] S. Eckart,“High Performance Software MPEG Video Player for PCs”, In Proceedings of *IS&T/SPIE Digital Video Compression: Algorithms and Technologies 1995*, Feb. 1995.

[20] A. Eustace, A. Srivastava,“ATOM: A Flexible Interface for Building High Performance Program Analysis Tools”, WRL Technical Note TN-44, Digital Equipment Corporation, July 1994.

[21] C. Federighi, L. Rowe,“A Distributed Hierarchical Storage Manager for a Video-on-Demand System”, In Proceedings of *1994 IS&T/SPIE Symposium on Electronic Imaging: Science and Technology*, San Jose, CA Feb. 1994.

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

[23] W. Feng, S. Sechrest,“Critical Bandwidth Allocation for the Delivery of Compressed Prerecorded Video”, *Computer Communications*, Vol. 18, No. 10, Oct. 1995, pp. 709-717.

[24] W. Feng, F. Jahanian, S. Sechrest,“An Optimal Bandwidth Allocation Strategy for the Delivery of Compressed Prerecorded Video”, CSE-Technical Report 260-95, University of Michigan, Sept. 1995.

[25] W. Feng, F. Jahanian, S. Sechrest,“An Optimal Bandwidth Allocation Strategy for the Delivery of Compressed Prerecorded Video”, *To appear ACM/Springer-Verlag Multimedia Systems Journal*.

[26] W. Feng, F. Jahanian, S. Sechrest,“A Network Cost Model for the Critical Bandwidth Allocation Approach,” In Proceedings of *IASTD/ISMM International Conference on Distributed Multimedia Systems and Applications*, Stanford, CA, Aug. 1995.

[27] W. Feng, F. Jahanian, S. Sechrest,"Providing VCR Functionality in a Constant Quality Video-On-Demand Transportation Service", In Proceedings of *3rd IEEE International Conference on Multimedia Computing and Systems,* Hiroshima, Japan, June 1996.

[28] W. Feng, F. Jahanian, S. Sechrest,"Providing VCR Functionality in a Constant Quality Video-On-Demand Transportation Service", CSE-TechReport 271-95, Dec. 1995.

[29] W. Feng, S. Sechrest,"Improving Data Caching for Software MPEG Video Decompression", In *IS&T/SPIE Digital Video Compression: Algorithms and Technologies 1996*, San Jose, CA, Feb. 1996.

[30] D. Ferguson, Y. Yemini, C. Nikolaou,"Microeconomic Algorithms for Load Balancing in Distributed Computer Systems", In *IEEE 8th International Conference on Distributed Computing Systems*, San Jose, CA, 1988, pp. 491-499.

[31] D. Ferrari, A. Banerjea, H. Zhang,"Network Support for Multimedia: A Discussion of the Tenet Approach", *Computer Networks and ISDN Systems*, Vol. 26, 1994, pp. 1267-1280.

[32] D. Ferrari, A. Gupta and G. Ventre, "Distributed Advance Reservation of Real-Time Connections", In Proceedings of *5th Intl. Workshop on Network and Operating System Support for Digital Audio and Video*, Durham, New Hampshire, April 18-21, 1995, pp. 15-26.

[33] J. Fu, J. Patel, and B. Janssens,"Stride directed prefetching in scalar processors", In Proceedings of the *25th International Symposium on Microarchitecture*, December 1992, pp. 102-110.

[34] D.J. Gemmell, H.M. Vin, D. Kandlur, P.V. Rangan, L.A. Rowe,"Multimedia Storage Servers: A Tutorial", *IEEE Computer*, Vol. 28, No. 5, May 1995, pp. 40-49.

[35] D.J. Gemmell, J.Han,"Principles of Delay Sensitive Multimedia Data Storage and Retrieval," *ACM Transactions on Information Systems*, Vol. 10, No. 1, Jan. 1992, pp. 51-90.

[36] D.J. Gemmell, H.M. Vin, D. Kandlur, P.V. Rangan, L. Rowe,"Multimedia Storage Servers: A Tutorial", *IEEE Computer*, May 1995, Vol. 28, No. 5, pp. 40-49.

[37] M. Ghanbari, V. Seferidis,"Cell-Loss Concealment in ATM Video Codecs", *IEEE Transactions on Circuits and Systems for Video Technology,* Vol. 3, No. 3, June 1993, pp. 238-247.

[38] K.L. Gong, L.A. Rowe,"Berkeley MPEG-1 User's Guide", University of California - Berkeley, Jan. 1995.

[39] Pawan Goyal, Harrick M. Vin,"Network Algorithms and Protocol for Multimedia Servers", In Proceedings of *INFOCOM 1996,* San Francisco, CA, March 1996, pp. 1371-1379.

[40] L. Gwennap,"New PA-RISC Processor Decodes MPEG Video", *Microprocessor Report*, Jan 24, 1994, pp. 16-17.

[41] L. Gwennap,“LSI Delivers MPEG Decoder for Digital TV”, *Microprocessor Report*, May 10, 1993, pp. 20-21.

[42] L. Gwennap,“Intel’s P6 Uses Decoupled Superscalar Design”, *Microprocessor Report*, Feb. 16, 1995, pp. 9-15.

[43] John Hennessy and David Patterson, *Computer Architecture: A Quantitative Approach*, Morgan Kaufmann Publishers, 1990.

[44] HP. “HP 9000 Series 700 Model 735”, Specification Sheet, http://www.dmo.hp.com/wsg/products/735ds.html, 1994.

[45] K. Jeffay, D.L. Stone, T. Talley, F.D. Smith,“Adaptive, Best-Effort, Delivery of Audio and Video Data Across Packet-Switched Networks”, In Proceedings of *Third International Workshop on Network and Operating System Support for Digital Audio and Video*, La Jolla, CA, Nov. 1992, pp. 3-14.

[46] N.P. Jouppi, “Improving Direct-Mapped Cache Performance by the Addition of a Small Fully-Associative Cache and Prefetch Buffers”, In Proceedings of *The 17th Annual International Symposium on Computer Architecture*, pp. 364-373, 1990.

[47] H. Kanakia, P.P. Mishra, A. Reibman,“An Adaptive Congestion Control Scheme for Real-Time Packet Video Transport”, In *Proceedings of ACM SIGCOMM 1993*, September 1993, pp. 20-31.

[48] D. Kandlur, M. Chen, Z.Y. Shae,“Design of a Multimedia Storage Server” , In *IS&T/SPIE Symposium on Electronic Imaging Science and Technology*, San Jose, CA, Feb. 1994.

[49] H.P. Katseff, B.S. Robinson,“Predictive Prefetch in the Nemesis Multimedia Information Service”, In Proceedings of *ACM Multimedia 1994*, San Francisco, CA, Oct. 1994, pp. 201-209.

[50] A.C. Klaiber, H.M. Levy,“An Architecture for Software-Controlled Data Prefetching,” In Proceedings of *the 18th Annual International Symposium on Computer Architecture*, pp. 43--53, 1991.

[51] J.F. Kurose, R. Simha,“A Microeconomic Approach to Optimal Resource Allocation in Distributed Computer Systems”, *IEEE Transactions on Computers*, Vol. 38, No. 5, May 1989, pp. 705-717.

[52] S. Lam, S. Chow, D. Yau, “An Algorithm for Lossless Smoothing of MPEG Video”, In Proceedings of *ACM SIGCOMM 1994*, 1994.

[53] D.H. Lee, “Design of a Motion JPEG (M/JPEG) Adapter Card”, In Proceedings of *SPIE Digital Video Compression on Personal Computers: Algorithms and Technologies*, San Jose, CA, Feb. 1994, pp. 2-12.

[54] W. Lee, J. Golston, R.J. Gov, Y. Kim, “Real-time MPEG Video Codec on a Single-Chip Multiprocessor”, In *Proceedings SPIE Digital Video Compression on Personal Computers: Algorithms and Technologies*, San Jose, CA, Feb. 1994, pp. 33-43.

[55] D.J. LeGall, “A Video Compression Standard for Multimedia Applications,” *Communications of the ACM*, Vol. 34, No. 4, (Apr. 1991), pp. 46-58.

[56] T.D.C. Little, D. Venkatesh,“Prospects for Interactive Video-On-Demand”, *IEEE Multimedia*, Vol. 1, No. 3, Fall 1994, pp. 14-24.

[57] P. Lougher, D. Shepherd,“The Design of a Storage Sever for Continuous Media”, *The Computer Journal*, Vol. 36, No. 1, Feb. 1993, pp. 32-42.

[58] S. Low, P. Varaiya,“An Algorithm for Optimal Service Provisioning Using Resource Pricing”, In Proceedings of *IEEE INFOCOM 1994*, Vol. 1, pp. 368-373.

[59] P. Pancha, M. El Zarki,“MPEG Coding for Variable Bit-Rate Video Transmission”, *IEEE Communications Magazine*, Vol. 32, No.5, May 1994, pp. 54-66.

[60] P. Pancha, M. El Zarki,“Prioritized Transmission of Variable Bit Rate MPEG Video”, In *IEEE GLOBECOM 1992*, Dec. 1992, pp. 1135-1139.
U. Penn.

[61] P. Pancha, M. El Zarki, “Bandwidth Allocation Schemes for Variable Bit Rate MPEG Sources in ATM Networks,” *IEEE Transactions on Circuits and Systems for Video Technology*, Vol. 3, No. 3, June 1993, pp. 190-198.

[62] P. Pancha, M. El Zarki,“Bandwidth Requirements of Variable Bit Rate Sources in ATM Networks”, In Proceedings of *INFOCOM 1993*, March 1993, pp. 902-909.

[63] A. Parekh,“A Generalized Processor Sharing Approach to Flow Control in Integrated Services Networks”,Ph.D. Thesis, The Massachusetts Institute of Technology, 1992.

[64] K. Patel, B.C. Smith, L.A. Rowe,“Performance of a Software MPEG Video Decoder”, In Proceedings of *ACM Multimedia 1993*, Anaheim, CA, August 1993, pp. 75-82.

[65] A.K. Porterfield,“Software methods for improvement of cache performance on supercomputer applications”,Technical Report COMP TR 89-93, Rice University, May 1989.

[66] S. Ramanathan, P. V. Rangan,“Adaptive Feedback Techniques for Synchronized Multimedia Retrieval over Integrated Networks,” *IEEE/ACM Transactions on Networking*, Vol. 1, No. 2, April 1993, pp. 246-260.

[67] P. Venkat Rangan, H.M. Vin,“Designing File Systems for Digital Video and Audio”, In *Proceedings of the 13th ACM Symposium on Operating Systems Principles,* Operating Systems Review, Vol. 25, No. 5, October 1991, pp. 81-94.

[68] P. Venkat Rangan, H.M. Vin,“Efficient Storage Techniques for Digital Continuous Multimedia,” *IEEE Transactions on Knowledge and Data Engineering*, Vol. 5, No. 4, Aug. 1993, pp. 564-573.

[69] D. Reininger, D. Raychaudhuri, et. al, “Statistical Multiplexing of VBR MPEG Compressed Video on ATM Networks”, In Proceedings of *IEEE INFOCOM 1993*, March 1993, pp. 919-926.

[70] A.A. Rodriguesz, K. Morse, “Feasibility of Video Codec Algorithms for Software-Only Playback,” In Proceedings of *SPIE Digital Video Compression on Personal Computers: Algorithms and Technologies*, San Jose, CA, Feb. 1994, pp. 130-139.

[71] L.A. Rowe, K. D. Patel, B.C. Smith, K. Liu,"MPEG Video in Software: Representation, Transmission, and Playback", In Proceedings of *High Speed Networking and Multimedia Computing, IS&T/SPIE Symposium on Electronic Imaging, Science, and Technology*, San Jose, CA Feb. 1994.

[72] L. Rowe,"Video Compression, What to Do When Everything is Changing", *Invited Talk Usenix 1994.*

[73] L.A. Rowe, B.C. Smith,"A Continuous Media Player", In Proceedings of the *3rd International Workshop on Network and Operating System Support for Digital Audio and Video*, San Diego, CA, Nov. 1992.

[74] P. J. Shenoy, H. M. Vin, "Efficient Support for Scan Operations in Video Servers", In Proceedings of the *3rd ACM Conference on Multimedia*, October, 1995.

[75] M. Slater,"Inside the Pentium Successor", *PC Magazine*, Vol.14, No.7, April 11, 1995, p. 29.

[76] A.J. Smith, "Cache Memories", *ACM Computing Surveys*, Vol. 14, No. 3, pp. 473-530, Sept. 1982.

[77] A. Srivastava, A. Eustace,"ATOM: A System for Building Customized Program Analysis Tools", WRL Research Report 94/2, Digital Equipment Corporation, March 1994.

[78] D. Stone, K. Jeffay,"Queue Monitoring: A Delay Jitter Management Policy", In Proceedings of the *4th International Workshop on Network and OS Support for Digital Audio and Video*, 1993.

[79] R. Sugumar,"Multi-Configuration Simulation Algorithms for the Evaluation of Computer Designs", Ph.D. Thesis, The University of Michigan, 1993.

[80] H.M. Vin, P. Goyal, A. Goyal, A. Goyal,"A Statistical Admission Control Algorithm for Multimedia Servers", In Proceedings of *ACM Multimedia 1994*, San Francisco, CA, Oct. 1994, pp. 33-40.

[81] H.M. Vin, P.V. Rangan,"Designing a Multi-user HDTV Storage Server," *IEEE Journal on Selected Areas in Communications*, Vol. 11, No. 1, Jan. 1993, pp. 152-164.

[82] G.K. Wallace, "The JPEG Still Picture Compression Standard," *Communications of the ACM*, Vol. 34, No. 4, (Apr. 1991), pp. 30-44.

[83] P. Willis,"MPEG-2 Digital TV All Set to Go", *Electronics World and Wireless World*, Vol. 99, No. 5, May 1993, pp. 356-357.

[84] L. Wolf, L. Delgrossi, R. Steinmetz, S. Schaller and H. Wittig, "Issues of Reserving Resources in Advance", In Proceedings of *5th Intl. Workshop on Network and Operating System Support for Digital Audio and Video*, Durham, New Hampshire, April 18-21, 1995, pp. 27-37.

[85] H. Zhang, S. Keshav,"Comparison of Rate-Based Service Disciplines", In Proceedings of *ACM SIGCOMM '91*, Sept. 1991, pp. 113-121.

[86] D. Zucker, M.J. Flynn, R. Lee,"A Comparison of Hardware Prefetching Techniques for Multimedia Benchmarks", Stanford University, Computer Science Lab Technical Report 95-683, December, 1995.