INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

ProQuest Information and Learning
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA
800-521-0600

University of Alberta

Efficient Transmission Scheduling Schemes for Video-on-Demand Systems

by

Fulu Li

A thesis submitted to the Faculty of Graduate Studies and Research in partial fulfillment of the requirements for the degree of **Master of Science**.

Department of Computing Science

Edmonton, Alberta
Spring 2000

University of Alberta

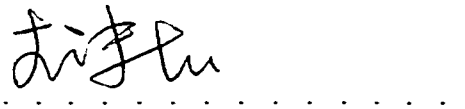
Library Release Form

Name of Author: Fulu Li

Title of Thesis: Efficient Transmission Scheduling Schemes for Video-on-Demand Systems

Degree: Master of Science

Year this Degree Granted: 2000



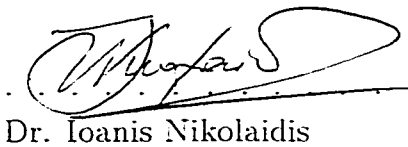
Fulu Li
710 210 Woodridge Crescent,
Nepean, Ontario
Canada, K2B 8E9

Date: Jan. 19, 2000

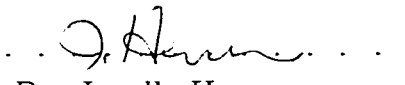
University of Alberta

Faculty of Graduate Studies and Research

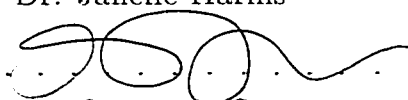
The undersigned certify that they have read, and recommend to the Faculty of Graduate Studies and Research for acceptance, a thesis entitled **Efficient Transmission Scheduling Schemes for Video-on-Demand Systems** submitted by Fulu Li in partial fulfillment of the requirements for the degree of **Master of Science.**



Dr. Ioanis Nikolaidis



Dr. Janelle Harms



Dr. Giancarlo Succi

Date: 01/17/2000

Abstract

Video–on–Demand (VoD) systems face scalability problems due to the need to satisfy numerous requests for several different videos given the limited bandwidth of the communication links. In order to provide scalable solutions, existing VoD proposals can be roughly divided into two categories: (a) *scheduled multicast* and (b) *periodic broadcast*. Scheduled multicast schemes collect user requests over successive time intervals. The users requesting the same video within the same interval will receive the video stream through a *single* multicast of the entire video. Periodic broadcast schemes operate by periodically broadcasting the same video.

In this thesis we propose (a) a novel scheduled multicast scheme based on a time–dependent bandwidth allocation approach, (b) a Trace–Adaptive Fragmentation (TAF) scheme for periodic broadcast of Variable Bit Rate (VBR) encoded video, and (c) a Loss–Less and Bandwidth-Efficient (LLBE) protocol for periodic broadcast of VBR video. We have designed, simulated and evaluated the proposed schemes and the simulation results demonstrate the benefits, flexibility and feasibility of the proposals.

... the cluster of technologies we call the Internet has very distinctive powers: a unique ability to complement, to reinforce and to enhance many of our most powerful traditional approaches to university teaching and the process of learning..
- Neil Rudenstine, Harvard President

Acknowledgements

My heartfelt gratitude is due to Dr. Ioanis Nikolaidis for his support and guidance during the course of this work and for his generous travel support for presentation at NOSSDAV'99, NJ, USA. I am grateful to the support staff of the Department of Computing Science for various timely helps. I thank my friends for their constructive comments on this work. I thank my wife for her assistance.

Contents

List of Figures

Chapter 1

Introduction

With the popularity of the Internet, the advent of high-speed network and the improvement of data compression technology, Media–on–Demand (MoD) is becoming a major application of future digital networks. In pure-MoD, a dedicated unicast stream is established for each user request. Pure-MoD does not scale well with the user population and becomes very expensive when a large number of concurrent requests have to be accommodated. Obviously, MoD and, in particular, Video–on–Demand (VoD) systems face scalability problems in a limited bandwidth environment. To provide scalable MoD, new efficient protocol architectures must be employed that target bandwidth efficiency.

1.1 Media on Demand

MoD refers to media services in which a user is able to request from a server any media content at any time. It encompasses many applications such as video-on-demand, news-on-demand, distance learning, home shopping, training programs etc. The following discussion focuses on VoD, but the principles are equally applicable to other media objects.

Throughout this work we shall make the following assumptions about the VoD architecture: A dedicated link, such as cable or satellite, is used to distribute the video data from the video server to the user. The prerecorded videos are stored at the video server. The server reads the video data from local high-performance storage media. Each client consists of a "set–top box" and secondary (disk) storage to read and write the video data. The user who wishes to watch a particular video starts receiving the video data transmitted by the server and the video data can be cached at the user's disk before it is decoded and displayed.

1.2 CBR vs VBR

Video sources are typically encoded (i.e. compressed) in order to reduce their storage and bandwidth requirements. One approach to video encoding, called Constant Bit Rate (CBR) encoding, produces a data stream with nearly constant bandwidth. Typically, CBR encoding operates by modifying the quantization scale, on–the–fly, during compression in order to maintain constant bit rate at the output of the encoder

[36]. The variable quantization causes the encoded video to be of variable quality. For open–loop Variable Bit Rate (VBR) encoding, the quantization scale remains constant throughout the encoding process, which often produces highly variable bit rate. For a given video and for the same perceptible quality level, the bit rate for CBR video is typically 2 times or more the average bit rate of VBR video [9]. In addition, VBR encoding schemes are becoming commonplace (MPEG-1 and MPEG-2) and extensive libraries of video material are already available in VBR form. Therefore, there is potential to obtain further performance improvements by using VBR video.

It is well known that VBR compressed video exhibits significant rate variability even when the rate is computed over time intervals as large as several minutes [7]. The excessive traffic rate variability is frequently called *burstiness*. The burstiness complicates the design of efficient real-time storage, retrieval and transport mechanisms capable of achieving high resource utilization, i.e., high bandwidth efficiency. The so-called *smoothing* is essentially the averaging of the bit rate over certain time intervals in order to reduce the rate variability. Smoothing produces less bursty workloads, however, it comes at the cost of introducing delay, delay jitter and additional buffers at the receiver. The resulting delay and delay jitter control requirements, as well as their corresponding buffer dimentioning require complex mechanisms from the underlying network.

1.3 Scheduled Multicast and Periodic Broadcast

To reduce the cost of VoD systems and provide scalable VoD services, efficient techniques have been proposed, which can be roughly divided into two categories: (a) *scheduled multicast* and (b) *periodic broadcast*. Scheduled multicast is accomplished by grouping many requests for a given content arriving over a period of time and serving these requests by a *single* multicast stream. The technique of collecting requests is also called *batching*. Batching increases the efficiency of VoD. However, scheduled multicast schemes are demanding in terms of bandwidth for popular videos. For example, suppose there exists a popular video 120 minutes long. If the batching interval (also called *responsiveness delay*) for scheduled multicast is 5 minutes and there is a

request for every batching interval for this video, then 24 concurrent transmissions of the same video are underway of any point in time. Thus, the batching of popular videos results in low bandwidth efficiency. Instead, periodic broadcast schemes broadcast the videos periodically, i.e., every w seconds a new stream is started transmitting over the network for a given video. Thus, the worst service delay experienced by any subscriber is guaranteed to be less than w minutes. For a popular video, periodic broadcast schemes substantially reduce the bandwidth requirements compared with the scheduled multicast schemes.

In periodic broadcast schemes, the entire video is partitioned into smaller successive and non-overlapping segments before broadcasting, the operation of which is called *fragmentation*. Each segment of the video is then continuously broadcast on a separate channel. The particular fragmentation strategy used is a central point of any periodic broadcast scheme.

Periodic broadcast schemes bypass the need to process individual user requests and the transmission schedules used by periodic broadcast are determined off–line. In this sense, the periodic broadcast schemes for VoD can be seen as an evolution of earlier *teletext broadcast* schemes [5]. The computation of the schedules can account for the popularity of the videos and the maximum tolerable latency between the time a subscriber activates its set–top box ("tunes-in") and the point at which the uninterrupted playout of the entire video can start.

Finally, it has been repeatedly shown in the literature that the popularities of videos follow the Zipf distribution. A typical skew factor for a Zipf distribution representing video preferences is 0.271 (see [10]). That is, most of the demand (80%) is for a few (10 to 20) very popular videos. Periodic broadcast schemes can then be used for the popular set ("hot set") of videos while a form of scheduled multicast can be used for the less popular set ("cold set").

For the "cold set", we propose a scheduled multicast scheme for the distribution in the call admission control process based on a time-dependent traffic characterization of VBR video, the details of which are presented in Chapter 2. On the other hand, we propose a lossy periodic broadcast scheme for the distribution of "hot set" VBR videos with client-imposed constraints on its I/O bandwidth, which is presented in Chapter 3. Finally, We propose a lossless periodic broadcast scheme for the distribution of

"hot set" VBR videos aiming at achieving the least required bandwidth for a given start-up delay provided that per-video server and client bandwidth are coupled, the details of which are given in Chapter 4.

1.4 The Related Work

1.4.1 Traffic Characterization and Call Admission Control

In the area of VBR video traffic characterization and call admission process, Chang and Kung proposed a time–dependent piece-wise Constant–Bit–Rate bandwidth allocation scheme for VBR video traffic in [8]. In their proposal, they assume the client buffer must be preloaded with the initial frames of the video before the user starts playout. This, however, requires scheduling the prefetching of data to ensure that a large buffer is filled in advance of playout, placing high demand upon the buffer and complicating the bandwidth allocation for the call admission control.

McManus and Ross presented a dynamic programming methodology to find optimal piece-wise constant rates and intervals to characterize VBR video traffic for a variety of optimization criteria such as the minimization of the maximum rate subject to a maximum initial delay and the constraint on receiver buffer [29]. Their algorithm is close to impossible to use in practice since it is very slow (the computational complexity is $O(N^5)$, the N here stands for the number of frames in a video, typically larger than 10^5). Although they claim that a grid, which uses one element to represent a group of successive frames, can accelerate the computation, obviously, the introduction of a grid will no doubt result in much inaccuracy due to the coarse representation of the video traffic.

Knightly and Zhang proposed a Deterministic Bounding INterval-length Dependent (D-BIND) model to capture the worst-case bit rates over successive time intervals [23], which is a time–invariant VBR video traffic characterization with the computational complexity of $O(N)$ to calculate the multiple rate-interval pairs. Their proposal is to describe the worst case bit rate boundings and it is not efficient since the allocated bandwidth can not be renegotiated after the worst case is encountered.

As far as renegotiation is concerned, the work on optimal smoothing in [37] includes an accompanying Renegotiated CBR (RCBR) service class [16], which again

has to operate in real–time but the system is free to reject the renegotiation demands of the stream, thus resulting in a statistical QoS setup. The alternative to renegotiation that is presented in this thesis is to use a time-dependent envelope description of the traffic. In this way, the computation complexity is moved to before the transmission (i.e., in the call admission process).

Zegura et al gave a survey and present some new results in renegotiated service in [41]. They pose the renegotiation problem as consisting of two steps. The first step is responsible for generating a feasible transmission schedule and the second is to produce a renegotiation schedule. The algorithms presented in [41] are used to calculate the renegotiation points in the second processing step with the input data from a feasible transmission schedule which can be calculated through the optimal smoothing algorithm [37].

Finally, the MPEG-specific scheme presented in [42] by Zhao et al produces an envelope to capture the maximum bit rates of different frame types (I, B, P frames) among all of the GOPs (Group of Picture) of the video trace to do VBR video call admission control. Obviously, this approach is not general and not efficient in terms of the required bandwidth.

No matter which traffic characterization is chosen, the scheduled multicast scheme is often employed during the call admission process to reduce the cost of the VoD systems and provide scalable services.

1.4.2 Continuous Periodic Broadcast

The alternative to on-demand batched multicast is the continuous periodic broadcast. The staggered periodic broadcast of entire videos [10] is a scheme that requires bandwidth inversely proportional to the startup latency objective. For example, on a 155 Mb/sec link, multiplexing the broadcast of ten two-hour long movies (each encoded at 3 Mb/sec) we cannot provide a startup latency better than 24 minutes. Moreover, this type of staggered broadcast demands large secondary storage capacity and high secondary storage I/O capabilities at the set–top box. The solution to these issues is the fragmentation of the complete video into a sequence of segments. The first segment can be short such that its broadcast can be repeated very often resulting in short startup latency. The complexity lies in the timing of the broadcast

of subsequent segments such that no starvation of the receiver occurs. Moreover, the receiver can employ secondary storage to store (completely or partially) segments in anticipation of their playout.

The Pyramid Broadcasting (PB) presented by Viswanathan and Imielinski [39] was the first scheme to reduce the startup latency using fragmentation. In PB, each segment is broadcast on a separate channel. However, PB's drawbacks are (a) the large client buffer size which is usually more than 70% of the entire video and (b) the high disk bandwidth required to write data to disk as quickly as received. A number of efficient protocols have been proposed to address these issues. Aggarwal et. al. proposed the Permutation-based Pyramid Broadcasting (PPB) [3] which reduces the buffer size requirements down to approximately 50% of the video. Skyscraper Broadcasting (SB) presented by Hua et. al. [19] substantially reduces the disk to approximately 10% of the video using a novel fragmentation technique and a different broadcasting scheme. Recently, Hua et. al. proposed a Client-Centric Approach (CCA) [18] which incorporates the restriction on how many channels a receiver can download at any point in time.

Overall, the above periodic broadcast protocols (PB, PPB, SB, CCA) share a similar structure, that is, equal bandwidth for all channels and increasing size of segment lengths. CCA has shown the best performance by making maximum use of the client bandwidth and keeping a lower buffer space requirement. The bandwidth requirements of this family of protocols, for reasonable playout latencies, is roughly 7 times the video consumption bandwidth.

Another approach to periodic broadcasting schemes, that of Harmonic Broadcasting and its variants, exhibits a different structure, i.e., decreasing bandwidth for the channels and equal segment lengths. In this category fall Juhn and Tseng's Harmonic Broadcasting (HB) [21], Paris, Carter and Long's Cautious Harmonic Broadcasting(CHB) [30], Quasi-Harmonic Broadcasting (QHB) and Polyharmonic Broadcasting (PHB) [31]. These schemes aim to reduce the start-up latency and improve the bandwidth efficiency. PHB has given the best performance, especially as far as bandwidth efficiency is concerned. Typical storage requirements are near 40% of the video size and transmission bandwidth are roughly 5 times the video consumption bandwidth.

Until recently, all periodic broadcast techniques assumed CBR encoded videos. Recently, [36] proposed a series of multiplexing schemes such as Group of Picture (GoP) smoothing, buffered multiplexing and Join–the–Shortest–Queue (JSQ) prefetching to improve the data loss performance for the periodic broadcast of VBR encoded video (VBR-B). Notably, to this day, all of the existing VBR periodic broadcast schemes use a lossy multiplexing approach.

1.5 Thesis Contributions

The contributions of this thesis can be summarized as follows:

1. In chapter 2, we propose a time-dependent characterization of the VBR video traffic that maintains high network utilization while requiring zero buffering at the receivers and the network itself. We take each video trace as a whole without any fragmentation. We illustrate how to generate an abbreviated representation of VBR video in the form of a k–step envelope, which is subsequently used to develop the transmission schedules in the call admission control procedure of batched user requests. We present both an exact optimal algorithm and a fast sub-optimal heuristic to generate the k–step traffic envelope. Our proposal does not depend on the video encoding patterns and it gives the designer a great deal of flexibility. We note that most previous approaches of performing VBR VoD call admission control do not take into account time–dependent envelopes. Using only a few parameters to characterize the traffic of a VBR video in a time–invariant manner can result in poor link utilization while extensive underlying network support for scheduling and buffering traffic is still necessary [8]. In contrast, the proposed scheme essentially trades increased call admission computation for more efficient bandwidth utilization.

   We also consider a call admission scheme for the multicast flows that result from the batching of user requests. Multiplexing is used to carry the VBR video traffic in order to increase the network utilization. Even with multiplexing, the high rejection rate of user requests will still be a challenge because the total aggregate bandwidth can exceed the available link bandwidth. For this reason, a placement optimization method is used in the call admission process

to reduce the rate variability and the rejection rate. The running time of the call admission for realistic examples was found to be within 15 seconds, making it very well suited in VoD/MoD systems where an even larger delay penalty has to be paid anyway because of the delay related to the efficient implementation of batching.

2. For VBR video broadcast, due to the significant bit rate variability of VBR video, different fragmentation schemes can lead to drastically different aggregate traffic when multiplexing the periodically-broadcasted segments together. Therefore, different fragmentation schemes may result in very different packet loss rate. However, all of the periodic broadcasting schemes use "rigid" fragmentation techniques without any regard to their influence on the packet loss performance.

   In chapter 3, we consider the design of a dynamic *Trace-Adaptive Fragmentation* (TAF) strategy which takes the particular video traffic into account to achieve the minimal packet loss which can further improve the performance of VBR video broadcast and may well require far fewer resources.

   TAF operates by deriving a set of fragmentation choices for the given constraints (the number of segments, the startup latency and the uninterrupted playout) and selecting the one with the least packet loss rate for the aggregated segments based on the particular video trace.

3. In chapter 4, we present a novel Loss–Less and Bandwidth-Efficient (LLBE) protocol aiming at achieving the minimal bandwidth requirement while achieving exactly zero packet loss for the periodic broadcast of VBR video.

   LLBE can achieve any arbitrary a–priori per–video startup latency (i.e., delay from "tune–in" of the set–top box to start of uninterrupted video playout). The latency is deterministic, that is, *exactly* w seconds of startup latency is required. In this sense, LLBE provides predictable latency for the client devices, which can be considered preferable and more "fair" compared to the bounded but random startup latency.

   In LLBE, the entire video is fragmented into segments. Each segment is assigned

different bandwidth and is transmitted on a separate channel. The client set–top box starts downloading the information of all the segments of a specific video as soon as the client "tunes–in". As soon as the first segment is downloaded completely, the uninterrupted playout of the video can begin. Because of the schedule construction employed in LLBE, subsequent segments are guaranteed to be completely downloaded just prior to the point when they are needed to be consumed by the playout process.

1.6 Reading Guidance

For Chapter 2, 3 and 4, each chapter deals with a specific problem on efficient transmission scheduling of VoD systems, regarding which we propose the corresponding schemes and experimental results. For the convenience of stating each problem, the variables and symbols used in each chapter are self-contained. There is no uniform definition for them between different Chapters.

Chapter 2

An Efficient Scheduled Multicast Scheme

It is well known that efficient call admission schemes require precise traffic characterization. For the traffic with high bit rate variability such as VBR encoded video, it is a challenge to characterize the traffic precisely using only a few parameters. The proposed time–dependent envelope traffic characterization using a few simple parameters tries to guarantee both efficient use of the links and reasonable processing cost in the call admission procedure. To improve the efficiency of VoD systems, an efficient scheduled multicast scheme is employed in the call admission process, which is accomplished by grouping many user requests for the same video over a given batching interval and serving these requests by a single multicast stream. For heterogeneous clients, different clients may have different buffer sizes. Therefore, a zero buffer assumption in the proposed scheme is used to better support heterogeneous clients. Essentially, we trade more computation for better efficiency in terms of bandwidth and user requests acceptance.

2.1 The Envelope Generation Scheme

In this section, we describe an approach to generate an abbreviated representation of a stored VBR media stream in the form of a step–wise bandwidth envelope. A reasonable question is why not use the frame bitrate values of the stored video. The answer is that the bandwidth fluctuations of many VBR streams occur too frequent to be usable as an envelope representation. For example, a typical 2 hour feature video at 25 frames per second, corresponds to more than 170000 frames. If the computing resources are available, then a fine-grained envelope to the point of individual frames can be produced and used. However, in this work we show that significant bandwidth gains can be achieved with a much smaller number of steps. We also note that the bandwidth allocation renegotiation is subject to latencies (e.g., signalling protocol delays) that indicate it may not be possible to renegotiate the bandwidth on a frame-by-frame basis.

We assume that the VBR stream is captured and stored and that we can derive a sequence of values representing the instantaneous source activity in terms of bits generated by the source per unit of time. The time unit is selected such that fluctuations in the bit rate in a time scale smaller than this unit are inconsequential to the

nature of the application. For example, a natural choice for a unit of time for video traffic is a video frame period. For the sake of clarity we use the term *frame bitrate* to refer to the bitrate measurement during the frame period for a VBR sequence, thus making the correspondence to video traffic obvious.

In generating the step–wise envelope, each step can contain an arbitrary number of successive frames. The bandwidth corresponding to each step is equal to the maximum bandwidth required by a frame belonging to this step, i.e., equal to the maximum instantaneous bandwidth demand during the time period "covered" by the envelope step. Figure 2.1 illustrates the nature of the envelope construction. The purpose of the envelope is as follows: If the bandwidth allocated for a stream's connection follows a time–varying pattern according to the constructed envelope, the stream's transmission will suffer no delays since the allocated bandwidth satisfies all the worst case bandwidth demands of the source. Clearly, this approach wastes bandwidth by allocating more bandwidth than needed but as will become clear, the bandwidth waste can be minimized.

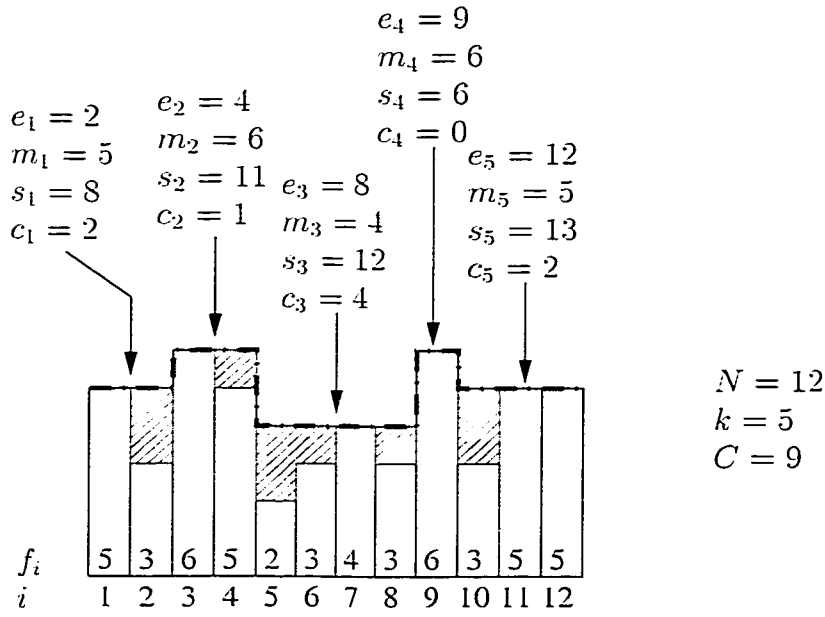


Figure 2.1: Example use of the defined notation for a sequence of 12 frames split into 5 segments ($N = 12, k = 5$).

Finally, the envelope generation problem can be formulated as follows: Given a frame bitrate sequence represented by f_i, $i = 1, \ldots, N$ and a number k, we wish

to split the frame bitrate sequence into k non-empty and pairwise non-intersecting steps/segments[1]. Let us use e_i, $i = 1, \ldots, k$ to denote the segments such that e_i stands for the *last* index of the frame sequence that belongs to the i-th segment, hence, the i–th segment spans all the frames from $f_{e_{i-1}+1}$ to f_{e_i}, inclusive. Clearly $e_k = N$, while for notational convenience we also introduce $e_0 = 0$. We define the following quantities:

$$m_{ij} = \max\{f_i, \cdots, f_j\} \quad \forall j \geq i, \quad i, j = 1, \ldots, N \tag{2.1}$$

$$s_{ij} = \sum_{k=i}^{j} f_k \quad \forall j \geq i, \quad i, j = 1, \ldots, N \tag{2.2}$$

The "cost" of a segment from frame i to j is therefore:

$$c_{ij} = (m_{ij}(j - i + 1)) - s_{ij} \quad \forall j \geq i, \quad i, j = 1, \ldots, N \tag{2.3}$$

Note that $c_{ii} = 0$. We further introduce the following notation using a single index, i, representing the corresponding segment, for (a) the maximum frame in a segment, (b) the sum of frames in a segment and (c) the cost of a segment:

$$m_i = m_{e_{i-1}+1 e_i} \quad \forall i = 1, \ldots, k \tag{2.4}$$

$$s_i = s_{e_{i-1}+1 e_i} \quad \forall i = 1, \cdots, k \tag{2.5}$$

$$c_i = c_{e_{i-1}+1 e_i} \quad \forall i = 1, \cdots, k \tag{2.6}$$

An example of the notation used here is given in Figure 2.1 for an example sequence of 12 frames split into 5 segments. Note there are several different ways to split the same sequence into the same number of segments. The one which we wish to derive is the one that minimizes the wasted bandwidth. Specifically, for a given k we wish to find the partition of f_i's into the segments of e_i's that minimizes the following expression:

$$C = \sum_{i=1}^{k} c_i \tag{2.7}$$

[1]Hereafter, the terms "step" and "segment" are used interchangeably.

We will call this minimization problem the k-**ENV** problem. Essentially, we wish to determine $k-1$ integer values for e_i, $i = 1, \cdots, k-1$ (since $e_k = N$ is fixed) where $e_1 < e_2 < \ldots < e_{k-1} < e_k$.

2.1.1 The Exact Solution for k–ENV

Despite its seemingly hard combinatorial nature, the k-**ENV** problem can be restated as a *single–pair k–edge shortest path problem in a directed acyclic graph.* The construction of the corresponding directed graph from an instance of the envelope problem can be accomplished as follows: Consider a graph of $N+1$ vertices. Let the vertices be labelled using the indices from 1 to $N+1$. We introduce directed edges (i, j) for $j \geq i$ with weight corresponding to $c_{i,j-1}$, as defined in the previous section. Essentially, an edge (i, j) with $j \geq i$ corresponds to taking the frame sequence from i to $j-1$ as one step of the envelope. Figure 2.2.a presents an example of this construction. As we have observed earlier, $c_{ii} = 0$, and hence the weights of the $(i, i+1)$ edges are zero. No other edges, except for the ones defined herein appear in the graph. Consider now the operation of the classic Bellman–Ford algorithm for determining the shortest paths in a graph (section 3.6 of [24] and also present in virtually all algorithms textbooks), whereby determining the shortest path between a source and all possible destinations can be seen as a matrix multiplication problem, with the ordinary addition operator taking the place of the multiplication and the minimization operator taking the place of addition. According to this formulation of the shortest path problem, and if the weight matrix is denoted by A, then A^k is a matrix containing the shortest path between any two i, j pairs that contains *no more than k edges.* The path from vertex 1 to vertex $N+1$ represented in the element $a_{1,N+1}$ of A^k corresponds to the minimization we seek over $C = \sum_{i=1}^{k} c_i$.

First, we note that the construction of the graph implies that any path from 1 to $N+1$ corresponds to a sequence of envelope steps. The shortest path minimizes the cost of the sum of the weights of the edges, and hence it minimizes the sum of the costs of the envelope steps.

Note that due to the construction of the graph, the shortest path does not include a cycle because no (directed) cycles are present in the graph in the first place. Namely,

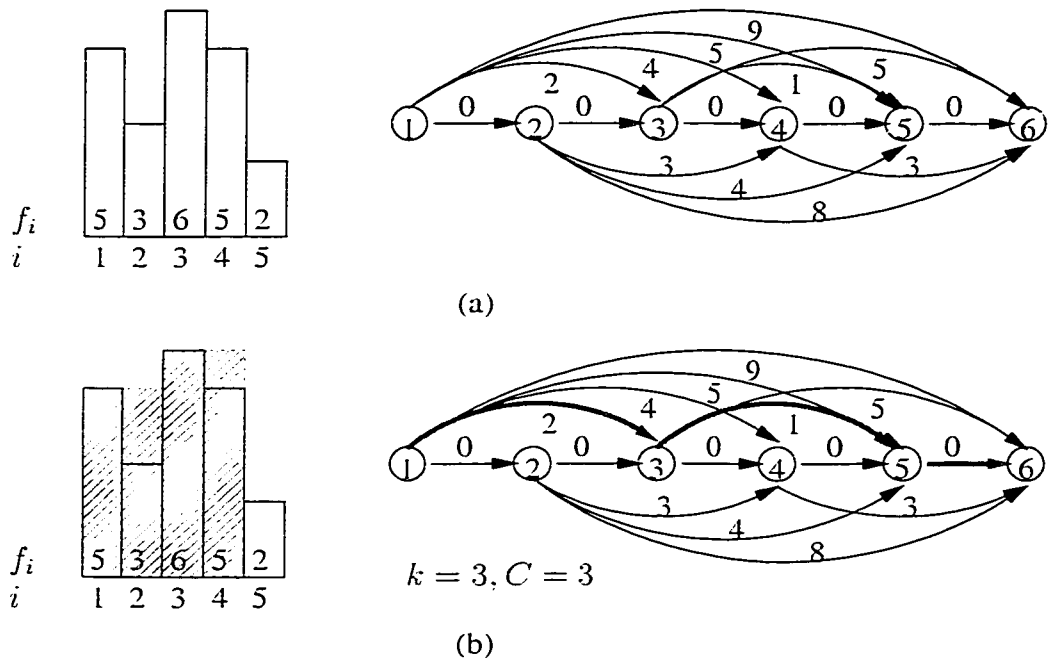


Figure 2.2: (a) The graph instance associated with an example instance of the k–**ENV** problem and (b) The optimal solution for $k = 3$ and its corresponding shortest path (thick gray(red) lines).

the labelling of the vertices from 1 to $N + 1$ can be seen as a topological order under which no edge points "backwards", i.e., no edge (i, j) exists with $j < i$. Hence, any shortest path constructed (and for that matter *any* path) on this graph is necessarily a simple (acyclic) path. The single-source shortest path problem can be solved using the Bellman-Ford algorithm. The running time for the shortest path algorithm is $O(kN^3)$. However, the space requirement ($O(N^2)$) of the algorithm still rules it out for practical use since for a typical video of two hours, the number of frames (N) is about 170,000 and the corresponding space requirement is several Gigabytes.

2.1.2 A Bottom–Up Heuristic for k–ENV

We present an algorithm for the k-ENV problem that generates suboptimal solutions. Because of the similarities in the content of a media stream, as in the case of MPEG streams, the resulting error relative to the exact optimal solution of the k-ENV problem is generally small. The algorithm's features are its space requirement of only $O(N)$ and its running time of $O(N^2)$. The algorithm can run for complete video

traces in commodity–produced scientific workstations with physical memory much less than 1G Bytes.

The algorithm operates in a bottom–up fashion. First, the entire sequence is covered in N envelope steps, each step corresponding to one frame. The total cost is initially zero. At each iteration we merge together two successive envelopes if this merge will bring the *smallest increase* in the overall cost. Hence, the cost is guaranteed to increase monotonically as we move from i–step to $(i-1)$–step envelopes. The merging stops when we reach the target number of envelope steps, k. A pseudocode of the proposed algorithm is presented in Figure 2.3.

```
steps = N;
while steps > k do
        cost_min = ∞ :
        for i = 1, ..., steps − 1 do
            cost_incr = c_{e_{i−1}+1 e_{i+1}} − c_{e_{i−1}+1 e_i} − c_{e_i+1 e_{i+1}};
            if cost_min > cost_incr then
              cost_min = cost_incr;
              marker = i;
            endif
        endfor
        for i = marker, ..., steps − 1 do
            e_i = e_{i+1};
        endfor
        steps − −;
endwhile
```

Figure 2.3: Pseudocode for the bottom–up heuristic solution to the k–**ENV** problem.

The intuition behind the suboptimal nature of the algorithm is as follows: by only merging envelopes, a step in a j–step envelope properly contains an integer number of steps that were present in the i–step $(i > j)$ envelope. Hence situations were partial overlap between envelopes in the i–step and the j–step envelope exists are not explored by this algorithm. The result of the limited search is a suboptimal solution but also the faster, $O(N^2)$, running time. Figure 2.4 presents an example of running the algorithm for gradually decreasing values of k (decreasing k towards the bottom of the figure).

2.1.3 Step–Wise Traffic Envelope Efficiency

In order to properly appreciate the savings in bandwidth caused by the stepwise envelopes, we first define b_e as the total bandwidth used up by the particular envelope. Intuitively, this is the integral of the step–wise bitrate over the entire period of the frame sequence. Let us also introduce b_m as the average *constant* bandwidth required by the stream in order to be transmitted over the time period which is equal to the frame sequence length, i.e. the rate at which the stream should have been sent if there were no concerns about the prompt real–time delivery of individual frames, as long as the entire sequence of frames was delivered in time equal to the sequence length. Furthermore, we let b_p represent the peak bandwidth which, if it were assigned to the stream, would be sufficient to deal with even the highest instantaneous demands of the source.

In the VBR video traces we used, even though the ratio of the largest over the smallest frame in a sequence is two orders of magnitude or more, the ratio of the peak frame size (and hence of the peak bitrate b_p) over the average frame size (and hence of the average bitrate b_m) is typically in the range between 6 and 18. The b_e is more than b_m but certainly less than b_p. In fact $b_e = b_p$ when $k = 1$, that is, the single–step envelope trivializes the scheme to a maximum bandwidth allocation. Since b_e is always larger than b_m we will use $(b_e/b_m) - 1$ as the measure of how much "worse" the envelope is doing in terms of additional bandwidth compared to the average bandwidth. The envelope scheme presented in this work strikes the middle ground between average and maximum bitrate for the entire trace but in addition, and contrary to rate–constrained versions of traffic characterization (e.g. Leaky Bucket–constrained sources), it *guarantees* the delay– and buffer–free delivery of the stream (that is, apart from transmission and propagation delays). Hence, the fact that it cannot achieve the huge savings that may be achievable by LB–constrained streams is compensated by the zero demand in buffer space from the network and virtually zero buffer demands from the receiving endpoint.

First we need to illustrate how the exact k–**ENV** algorithm behaves compared to the fast heuristic algorithm for the same problem. Figure 2.5 shows clearly the advantage of the optimal algorithm. However, the decrease of bandwidth waste with

increasing number of steps follows essentially the same slope in both the optimal and the heuristic algorithm. Note also that the minimum bandwidth waste by both algorithm is achieved at around the same number of steps[2]

We next present the bandwidth waste for complete video segments of 40,000 frames from traces available from [35]. The step–wise envelopes are produced using the heuristic k–**ENV** algorithm, hence, if one was able to run the exact algorithm on the same instances, an even better bandwidth waste figure would be attainable. All the 19 traces available to us demonstrated similar behavior. We present results from only three of the traces that we consider typical of the set: a newscast, a sporting event, and a feature video in Figure 2.8. Depending on the content, significant savings can be achieved even with modest number of steps in the envelope (10-100 steps).

Ultimately, the bandwidth waste vs. k depends on the particular frame sequence. For example, on an MPEG–compressed video stream, the k–**ENV** algorithm will tend to group together the frames that belong to a particular scene at a certain level of activity, such that consecutive low–activity or high–activity scenes are grouped together. Another property that we have observed is that the algorithm separates (occasionally in a single–frame envelope step!) very large frames, because their inclusion would inflate the overall cost. Thus, it isolates "outliar" frames and essentially requires that extra bandwidth is only occasionally reserved for the sake of dealing with such exceptionally sized frames.

2.1.4 System Synchronization

To apply the proposed time-dependent (k-ENV) bandwidth allocation scheme in the deterministic call admission control process, the bit rate renegotiation time points in terms of the constructed traffic envelope must be precisely synchronized to prevent packet loss since renegotiations are not instantaneous and the latency between signalling change of allocated bandwidth and its response may cause the bit rate remains unchanged across the renegotiation time points, which could result in overflow.

Similar to Chang et al [8], this problem can be solved by allowing extra delay period between the request for bandwidth increase/decrease in terms of envelope and

[2]The example of Figure 2.5 has been restricted to the first 500 frames of a video sequence because of the computation load and memory demands of the optimal algorithm.

its response, which is a multiple of frame time. This can be done by extending the time interval for a given bit rate in terms of envelope if the bit rate is higher than the rate during the adjacent time intervals across the renegotiation points. The idea behind this operation is illustrated in Figure 2.6. Obviously, the number of steps in the original envelope could be reduced by the step extension of the synchronization operation.

2.2 The Call Admission Control

In this section we present a call admission algorithm that can be performed on step–wise bandwidth envelope streams. First, we wish to clarify that call admission schemes that operate on a time–invariant representation of the connection traffic (as the one presented in this work) can summarize the state of the currently accepted connections in very little memory. Typically, this is accomplished by storing two values: the aggregate bandwidth and the aggregate buffer commitments. As connections are accepted (terminated), these aggregate quantities are increased (decreased) but the memory required for their representation remains fixed. On the other hand, schemes that operate on a time–dependent representation of the connection traffic may have to utilize significantly more space in order to accurately capture the time–dependent resource usage. Since we are not dealing with the potential of statistical multiplexing as we indicated earlier, no buffer resource allocation exists. In fact, a bandwidth allocation is performed exactly to *avoid* any buffer allocation in the network. However, the bandwidth call admission is complicated by the requirement to represent arbitrary time–dependent bandwidth commitments. For this reason we use the same representation that we used for an individual source, namely, a sequence of k steps where k is enough to represent the merged traffic from the envelopes of the constituting connections. Figure 2.9 depicts the generation of the new aggregate traffic from a new stream and an already calculated aggregate (can be zero if no stream is currently transmitted). Note that, as the merge is performed, if segments e'_i and e''_j overlap in time, then they will lead to the creation of a segment that describes their aggregate traffic, say e_l, such that $m_l = m'_i + m''_j$. Moreover, if k' and k'' are the number of steps for the current aggregate and the new stream respectively, then

their merged envelope contains k steps where $k \leq k' + k''$.

2.2.1 Call Admission Constraints and Objectives

In a MoD system, the acceptance of a connection is performed not only based on the system resources available at the time the request is made, but also on a prediction of how the acceptance of the request will influence the future requests. For example, one objective in request batching is to postpone the point in time at which a stream starts transmission in order to maximize the number of satisfied requests by one transmission. In the proposed call admission, three constraints/objectives need to be considered:

1. *Responsiveness Constraint:* The start time of transmission must be set within T time units from the point the request is received by the system, otherwise the request is considered rejected.

2. *Bandwidth Constraint:* The superposition of the new stream on the already transmitting streams must not exceed the available link bandwidth capacity B.

3. *Placement Objective:* The superposition of the new stream on the already transmitting streams, subject to items 1 and 2 above must be such that it minimizes the following expression:

$$v = \max_i m_i - \min_i m_i \tag{2.8}$$

   where m_i is the rate corresponding to the i-th segment as produced by the superposition of the new and the previously accepted streams.

4. *Shifting Optimization:* The call admission process can be accelerated using a shifting optimization method when choosing the placement point such that the expression of $\max_i m_i - \min_i m_i$ can be minimized for a given shifting window size (constraint(1)).

Ignoring constraint (1) can lead to an always feasible solution for any number of requests but the start of the transmission would gradually be postponed towards infinity, an obviously unwelcomed side-effect. Constraint (2) is a natural restriction

of the system. Objective (3) and equation 2.8 in particular appears as a form of "variance reduction" of the produced superposed stream. In reality, one cannot talk about "variance" in the current context since everything is deterministic. Instead, the purpose of objective (3) is to produce an aggregate stream *without extreme fluctuations* in the allocated bandwidth. In the ideal case: $v = \max_i m_i - \min_i m_i = 0$ which implies that the superposition is constant bandwidth. What is accomplished by this objective is that the future connections are not influenced by the particular "shape" that the superposition of the previous streams exhibits. Consequently, the superposition of drastically heterogeneous streams (extreme differences in "peaks" and "valleys" in their envelope representation) is gradually "smoothed" out by enforcing constraint (3).

Constraint (1) implies that the start time, denoted by T_a, of a new stream can only be placed within the first T time units after the request was received (see Figure 2.9). Hence the search for the T_a that optimizes (3) need only be performed in the 0 to T range. If the new stream arrives before an already schedule start time for the same stream, then this request will be served by a multicasting stream and the call admission process for this request is ended. If there exists no such a stream in the soon-to-be-serving stream list, the resulting call admission algorithm will try successively to place the new stream starting at 0 up to T. If, by placing the new stream at a certain offset, $T' < T$, it is discovered that there exists an m_i in the aggregate traffic envelope such that $m_i > B$ then this T' is immediately rejected from the feasible solutions and we advance the placement of the new stream to $T' + 1$. Eventually, From all the T' from 0 to T that were identified as feasible ($m_i < B \ \ \forall i$) we pick the one that minimizes equation 2.8. While it appears to be expensive to run this algorithm, we should note that the merge of sequence of k' and one of k'' steps takes time $k' + k''$ (it is essentially a merge of two sorted sequences). Thus, the overall calculation takes $O(T(k' + k'')) = O(Tk)$ and it is closely tied to the selection of T. However, in principle, the longer T is, the more time we can afford to spend in the call admission procedure.

However, to accelerate the call admission process, we present a shifting optimization method. Instead of shifting the stream by one frame time every time, the algorithm skips the amount of time based on the information of the steps at which

the maximum and minimum rates are reached for m_i.

Note that as time progresses, the representation of the bandwidth commitments can be truncated progressively from the lower time values, as they get updated to reflect the current wallclock time.

2.2.2 Call Admission Experiments

In the presented experiments we used a set of 9 MPEG streams (see [35] for a discussion about the source and the characteristics of these traces). It has been repeatedly shown in this literature that the popularities of videos follow the Zipf distribution. A typical skew factor for a Zipf distribution representing video preferences is 0.271 (see [10]). In our experiments, the user requests for the videos are thus determined by the Zipf distribuion model. The fomula to calculate the popularity for each video is as follows:

$$P_i = \frac{1/(1-0.271)^i}{\sum_{i=1}^{9} 1/(1-0.271)^i}, \quad i = 1, \ldots, 9. \tag{2.9}$$

The user request interarrivals follow a Possion distribution with an interarrival time of 5, 10 and 15 seconds apart. The rate of requests was predetermined to be in a range which leads the system close to overload in order to identify clearly the impact of the call admission and the batching on the performance. The available link bandwidth was set to $B = 155$ Mbps. The step–wise bandwidth envelopes for the individual streams were produced for various values of k. A user request stream was simulated with average interarrival times of 5, 10 and 15 seconds. Each request was processed independently. Batching would register a "hit" if a request for a stream would arrive before an already schedule start time for the same stream. All the requests were assumed to allow the same time T during which the stream was expected to start its transmission.

The call admission took no more than 16 seconds (in the worst case) on a Pentium II processor. This is insignificant time compared to T which was set to roughly 80 seconds (2000 frame times) and 400 seconds (10000 frame times) respectively. As a guideline, it is suggested that when such an algorithm is run, we pre–compensate for the fact that some the call admission worst case running behavior may occur and

instead of searching in the 0 to T range we search in the t_0 to T where t_0 is the worst case running time of the call admission (see the conclusions for some related discussion). Finally, but importantly, the user requests for the particular items (1 through 19) were following a Zipf distribution. The simulation will be eventually terminated when it is up to 24 hours from the beginning and we assume that long time is enough to capture the typical behavior of users' requests and the steady video traffic. Our key findings are:

High granularity envelopes may not be valuable for higher frequency request arrivals: In Figures 2.10 and 2.11 one can see the utilization and rejection percentage as a function of the number of envelope steps for different request arrival rates. Observe the extreme points in each curve. Consider in particular the distance (in the y–axis) between the smallest and largest value in the rejected requests percentage. The more frequent the requests, the lesser the impact of moving from a coarse–grained envelope (low k) to a fine–grained envelope (high k). Thus, if computation complexity is at a premium and we know that requests are frequent, we should not rely on extremely fine envelopes alone in order to drastically decrease the percentage of rejected requests. A better idea is to increase the number of steps *in conjunction* with an increase to T.

The finer the envelopes the lower the utilization: The decrease in the utilization in Figure 2.10 has to be put in the proper context together with the simultaneous decrease in rejected requests of Figure 2.10. That is, the utilization is decreasing at the same time that more requests are being accepted! This is the clear advantage of the presented envelope and call admission techniques. The feasible admission points for a new request increase as the envelopes get finer, thus it is possible to have, at the same time, both lower utilization (because the total bandwidth used by the connections is smaller when he envelopes are finer) and to have more requests accepted.

Diminishing returns in terms of rejected requests for increasing admission window, T: Obviously, one cannot hope on improvements by increasing T if the requests arrive very far apart. However, Figure 2.15 illustrates a subtler point: for $T = 10000$ the difference in the rejected connections between the low and high arrival rate of requests is very small. The reason is that the $T = 10000$ is sufficiently large to virtually accept most connections. The difference is that (as can be deduced

from Figure 2.14) that at higher request rates the resulting connections are "packed" closer together (resulting in higher utilization) while for lower request rates they are "spread" over a longer period of time and hence the drastically different utilization for virtually the same request rejection percentage.

Batching improves the rejection percentages and not utilization: In Figure 2.16 we can clearly see the impact of batching on the percentage of rejected requests as a function of the number of envelope steps at a request interarrival time of 10 seconds. The finer the envelope, the more the accepted requests but the the difference between batching and no–batching is consistently in the 30 to 40 % range. The utilization results are not provided because they are essentially identical. The impact of batching is on the request rejection, and to a much lesser extent on the utilization.

In addition, note that T is crucial part of the call admission complexity. Thus, what the results in 2.15 and 2.14 suggest is that in a real system we should probably adjust T based on the request arrival rate that we observe in order to keep T as small as acceptable in order to accelerate the call admission computation. Also, Figures 2.12 and 2.13 reveal the expected worse behavior of the peak–rate allocation ($k = 1$).

Finally, note that when the experiments are expanded to 19 streams, a slightly worse percentage of rejected requests is recorded. This is a natural consequence of increasing the choices for picking one stream (according to the Zipf distribution always). The more the streams, the less likely that several requests for the same stream will be seen together over a period of time since there are more now selections to pick from. Thus, for larger sets of streams the deterioration in the number of rejected requests comes from the reduced gains of the batching scheme. It may thus be advisable to expand T whenever the set of the choices for the streams expands.

The call admission process presented in this work schedules the start of the transmission of a requested stream such that the impact on the admission of future requested streams is minimized (this is accomplished through an optimization process). The running time of the call admission for realistic examples was found to be within 15 seconds, making it very well suited in VoD/MoD systems where batching is employed and an even larger delay penalty has to be paid anyway because of the delay related to the efficient implementation of batching.

2.3 Comparative Study

2.3.1 Comparison with PCBR

A bandwidth allocation scheme named Piecewise Constant Bit Rate (PCBR) has been proposed by Chang et al in [8]. PCBR generates a rate profile with piecewise constant bit rate segments of variable length. Given the receiver's buffer size, PCBR tries to make each segment as long as possible based on two cost functions $MaxBW(T)$ and $MinBW(T)$ which stand for the maximum allowed bandwidth that can be applied from time 0 to T without overflowing the receiver buffer and the minimum allowed bandwidth that can be applied from time 0 to T without underflowing the receiver buffer respectively. In our strategy, we assume there is no buffer requirement at all in the whole network and we generate the envelopes with arbitrary number of steps primarily based on the bandwidth efficiency.

Although they try to make the segments as long as possible in the PCBR scheme, they have no idea in advance that how many segments there is going to be for a given video stream and a given buffer size. In contrast, our proposed technology can generate arbitrary number of CBR segments to match the network conditions flexibly.

In their scheme, in order to make the segment as long as possible, they introduce the preload policy and they picked the preload policy to be half of the buffer size in all their experiments. Typically, the preloaded buffer size before the video is played is as much as several Mega bytes which is equal to the sum of several thousand of frames. However, they did not take care of the preloads in their admission policy. The profile of the video generated by PCBR does not include the preloaded data. In their admission, they assume that all of the requested videos are played with desired starting time zero. Then the problems are how they deal with the preloads, how they fill up the several Mega bytes data in the buffer in advance, how long it would be to fill up that much data before the video can be played and how they can figure out how much the additional bandwidth could be. From the technical point of view, the rate to fill up the buffer can not be very high, then there must be a long delay before the movie can be played. Obviously, there are some technical problems that they need to take into account.

Moreover, in their admission, they assumed the tolerable scheduling delay for a

video request was 10 minutes. Although they introduced that long delay for their schedule computation, they did not take advantage of the batching property which can be implemented while fitting the videos. In our approach, we apply batching technology in our admission scheme while aggregating the videos in order to satisfy a large number of customers who subscribe the same movie with one single transmission which will dramatically increase the bandwidth utilization and astronically reduce the rejection rate.

Further, they assumed video subscribers are fixed and each requested video was selected round robin from the available streams. These assumptions make their admission lack of flexibility. In our scheme we did not assume fixed video subscribers, instead we assume the request arrival follows a Possion distribution and the selection of the movies follows a Zip-f distribution, the values of which are known to closely match the popularities generally observed by video store rentals [1].

Finally, the optimization criteria in their admission scheme is to get the least aggregate peak rate within the video playback period; while the optimization goal in our admission approach is to minimize the difference between the aggregate peak rate and the aggregate bottom rate so as to minimize the rate variation. Apparently, our strategy can guarantee both high bandwidth utilization and low rate variation, hence our optimization criteria should be superior to theirs. Figure 2.17 shows the simulation results about the rejection rates with batching for their optimization criteria vs. ours. Figure 2.18 illustrates the results regarding the bandwidth utilization with batching for their optimization criteria vs. ours. From the results we can see, using Chang et al's admission strategy can get around 7 percent higher rejecting rate in general while the utilization rates are almost the same. Especially for a low number of steps, Chang's admission strategy can lead to both higher rejection rate and lower utilization.

The following Figure 2.7 shows the experimental results regarding the introduced delay by their preloading scheme for a given buffer space vs. the bandwidth efficiency in our k-ENV for the same number of steps with smoothing by the same delay or without smoothing. According to their approach, half of the buffer size is chosen as the preloads size. For the smoothing interval in our case, this is calculated based on how many frames they delayed for their preloads. Since the preloads size is half of the

buffer size in their approach, the buffer size needed in our case for the same number of frames in their preloads should be smaller than that in their case even in the worst case.

From the results, we can see that for a small buffer, the number of steps generated by their algorithm is huge. In that situation, the envelope efficiency for our k-ENV with the same number of steps is close to 100 percent. However, for a large buffer, to preload so many frames in advance is really a headache in their case either for the introduced delay or for the buffer to prefetching that much data, even for the bandwidth allocation.

As for our second scenario with smoothing, the envelope efficiency is determined by two major factors: the number of steps and the smoothing interval. The longer the smoothing interval, the lower the bandwidth waste. Similarly, the larger the number of steps, the higher envelope efficiency can be reached. However, when the buffer space in PCBR is decreased, the number of CBR segments, i.e., the number of steps, generated by PCBR algorithm will be increased and the delay time introduced by preloads will be decreased accordingly which results in shorter smoothing interval for k-ENV with smoothing. Therefore, the envelope efficiency for k-ENV with smoothing compared with PCBR for a decreasing buffer space could not be strictly monotonic. As we can see, the bandwidth waste for k-ENV with smoothing for a decreasing buffer space in PCBR is going down in general except a little increase when the buffer space is decreased from 2 MB to 1 MB.

2.3.2 Comparison with PCRTT

The dynamic programming methodology of McManus and Ross calculates Piecewise Constant-Rate Transmission and Transport (PCRTT) parameters which minimize receiver memory, i.e., buffer size, the number of rates, the maximum rate and the build-up delay in [29]. In their schemes, they introduce two functions, received(t) and removed(t); received(t) is the number of packets received by the receiver up through time t, removed(t) is the number of packets removed from the receiver buffer up through t frame-times. While we always assume received(t) and removed(t) are the same in our strategy, since we do not use buffer in the whole network. Once receiving the arriving frames, just transmit it in our case because we can always

guarantee that at any moment the transmission rate can be larger than the arrival rate. From their definition of the receiver memory, that is, B = max [received(t) - removed($n-1$)], we can see that it is impossible to expect that the receiver buffer can reach zero for their approach. In fact, the memory requirement ranges from 4 Mega Bytes (MB) to 7 Mega Bytes (MB) typically for a movie with 40,000 frames which can be played for 30 minutes or so. For a longer movie, the memory requirement can be larger accordingly. Their Dynamic Programming (DP) algorithm for the optimal PCRTT parameters seems to be a NP hard problem, which is why they give a DP heuristic. For the DP heuristic they presented to minimize the receiver memory, firstly they find the optimal K-sequence based on the associated cost function and the K breakpoints are all lying on removed(t), then they calculate the maximum amount by which removed(t) dominates the piecewise linear approximation defined by this K-sequence, finally they shift the approximation above by that much to avoid the receiver starvation. From their process, we can see that the final raising of the approximation even enlarges the buffer size by that much instead of minimizing the gap according to their definition of receiver memory. The complexity of optimal DP algorithm is O($N^2*C^2*(K+N)$), where K is the number of segments, N is the number of frames in the movie trace, C is the total number of fixed length packets in the video, typically, C is much larger than N. Therefore, the complexity of their algorithm is worse than $O(N^5)$. We have implemented their heuristic algorithm which has the complexity of $O(k*N^3)$. For K is 8 and N is 8000 (about 5 minutes for playout), it takes more than 24 hours on a Pentium II machine. Thus, for a typical movie with 80,000 (about 50 minutes for playout), it will take several months to calculate the schedule. Thus, their algorithm is close to impossible to be used practically because of its intolerable speed. Although they mention that they can use a grid, which uses one element to represent a group of frames, to solve the speed problem, it is obvious that the choosing of the grid will significantly affect the efficiency of the algorithm in itself due to its inaccurate representation of the video traffic. Our heuristic to find the optimal K-step envelope is much faster than theirs. For a movie with 40,000 frames, it typically takes less than 5 minutes to finish.

As for the packing admission approach they presented, there is no optimization criteria in it. They assume that whenever a user finishes viewing a video, he immediately

requests to see it again since they only use one movie to do the admission experiments which makes their admission proposal lack flexibility and feasibility. Moreover, they assume that time is divided into periods where the length of each period is equal to the length of the grid interval which introduces a large initial delay because the server must wait until the beginning of a period before it can begin to transmit. Therefore, the introduction of the grid in their scheme is coupled with a tradeoff between the simplification of the calculation of the K-sequence and the initial delay. In our admission strategy, we assume there are a bunch of movies whose requests possibility follow the Zip-f distribution and we also present an optimization criteria by shifting the newly-arriving movie stream in a given range to minimize the gap between the maximum aggregate bitrate and the minimum aggregate bitrate after admitting it. During the shifting process, we also introduce the batching technique to satisfy a large number of customers who request the same movie through one single transmission of that movie. In their admission scheme, they do not take use of batching technology. These important properties in our admission strategy make it more feasible and practical.

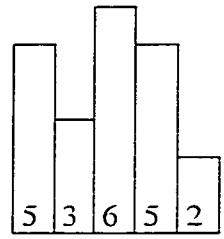
2.3.3 Comparison with Optimal Smoothing

Reducing rate variability of variable-bit-rate compressed video as smooth as possible for a given client buffer, i.e., optimal smoothing , has been proposed by Salehi et al in [37]. One of the hypotheses of their scheme is that each client must provide a buffer with a certain capacity, which, however, is not a must in our approach since we assume zero buffer is needed in the client side, even in the whole network.

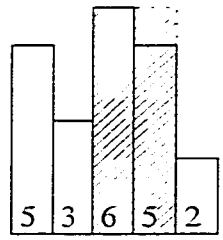
In [37], the authors assume data is consumed from the client buffer after any arrivals at time t. They define $d(t)$ as the amount of data consumed by the client at time t (i.e., the size of the t^{th} frame), and $a(t)$ as the amount of data sent by the server at time t. According to their definition, $D(t)$ and $A(t)$ represent the cumulative amounts of data consumed by the client and sent by the server, respectively, over (1, t); $B(t)$ denotes the maximum cumulative amount of data that can be received by the client over (1, t), without buffer overflow. Their algorithm is to find a feasible optimally-smoothed server transmission schedule if the client buffer neither starves nor overflows during stream playback, i.e., if $D(t) \leq A(t) \leq B(t)$ for all t.

However, when the capacity of the buffer that the client can provide is relatively small, i.e., several KB, the number of CBR segments generated by optimal smoothing algorithm which is the same notion as the number of steps in our k-ENV approach could be very large in which case the performance of optimal smoothing scheme is not efficient.

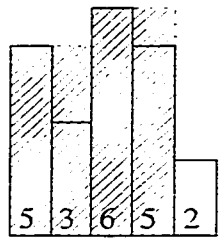
In summary, our proposed bandwidth allocation scheme for VBR video traffic demonstrates the great feasibility and high flexibility in the real implemenation for VoD systems.



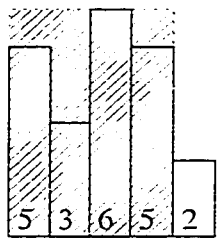
$k = 5 \quad e_1 = 1 \quad c_1 = 0 \quad C = 0$
$e_2 = 2 \quad c_2 = 0$
$e_3 = 3 \quad c_3 = 0$
$e_4 = 4 \quad c_4 = 0$
$e_5 = 5 \quad c_5 = 0$



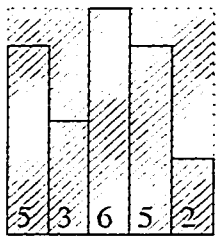
$k = 4 \quad e_1 = 1 \quad c_1 = 0 \quad C = 1$
$e_2 = 2 \quad c_2 = 0$
$e_3 = 4 \quad c_3 = 1$
$e_4 = 5 \quad c_4 = 0$



$k = 3 \quad e_1 = 2 \quad c_1 = 2 \quad C = 3$
$e_2 = 4 \quad c_2 = 1$
$e_3 = 5 \quad c_3 = 0$



$k = 2 \quad e_1 = 4 \quad c_1 = 5 \quad C = 5$
$e_2 = 5 \quad c_2 = 0$



$k = 1 \quad e_1 = 5 \quad c_1 = 9 \quad C = 9$

Figure 2.4: Envelope generation using the bottom–up heuristic for decreasing values of k from top to bottom, $N = 5$.

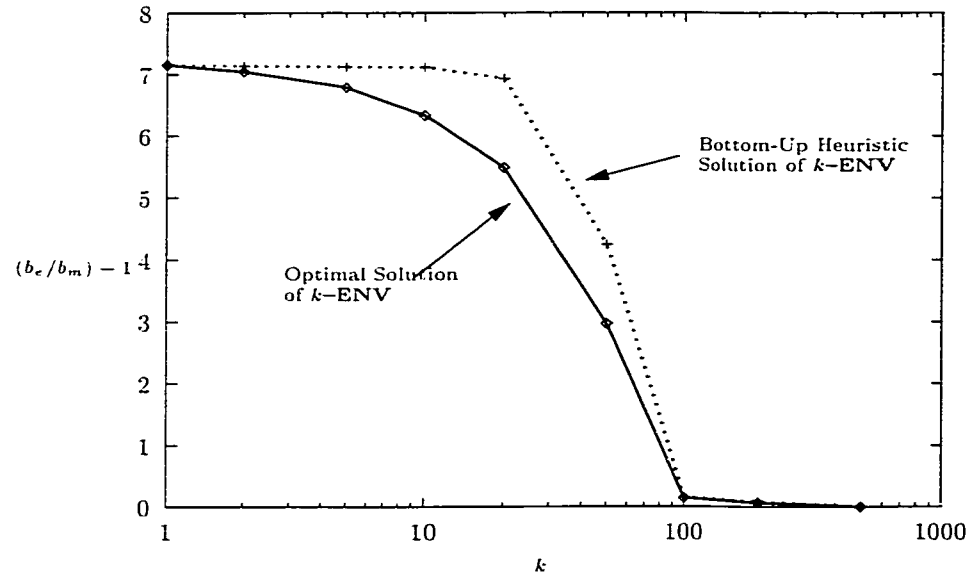


Figure 2.5: Comparison between optimal and the bottom-up heuristic solution. The quantity $(b_e/b_m) - 1$ on the y–axis represents how much extra bandwidth compared to the average bandwidth is used by the step–wise envelope of the connection, for a specific k.

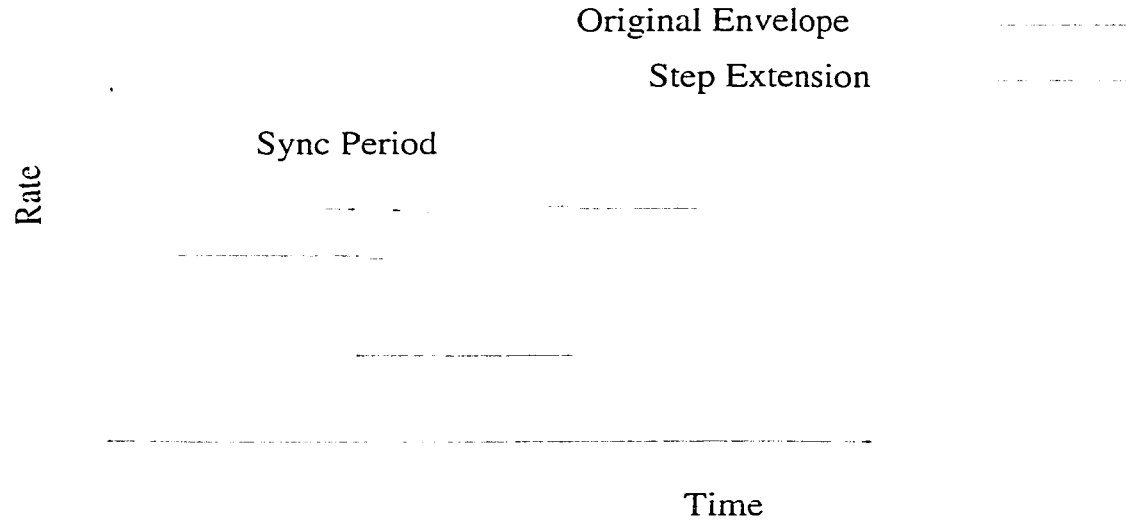


Figure 2.6: System synchronization with a certain synchronization period which is a multiple of frame time by extending a step on the side of which the rate is higher than that of the adjacent step in the envelope.

Preloads	k	k-ENV w/out smth.	k-ENV with smth.
1Mbytes	7	5.27	0.58
512kbytes	21	4.27	0.66
256kbytes	49	3.61	0.61
128kbytes	109	2.92	0.46
64kbytes	178	2.57	0.38
0.5kbytes	24056	0.04	0.00

Figure 2.7: Comparison with PCBR [8] regarding the delay introduced by its preloading in Bytes for a given buffer space vs. the bandwidth efficiency in k–ENV ($b_e/b_w - 1$) for the same number of steps (k) and the same video trace (Fuss) with smoothing by delaying the same time as preloads do or without smoothing.

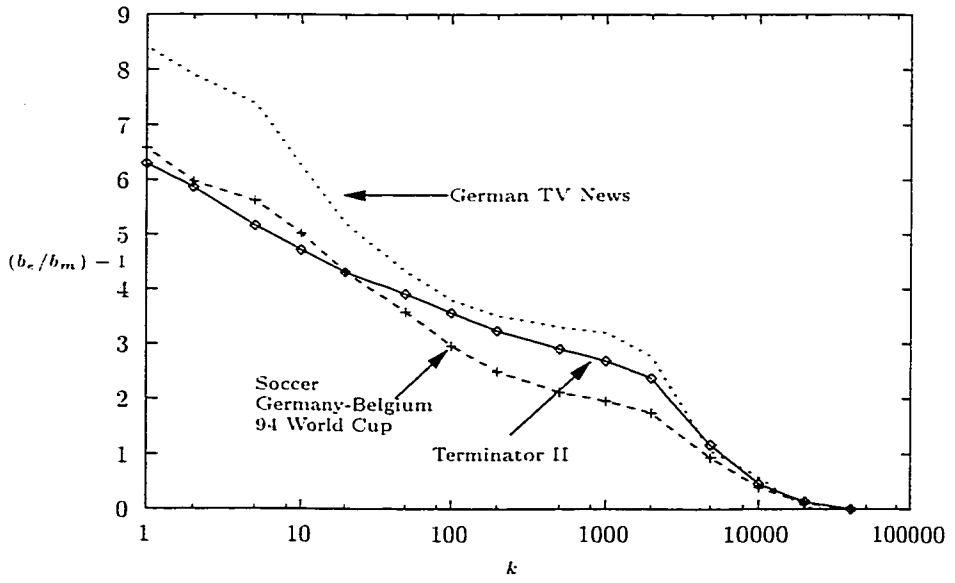


Figure 2.8: The additional bandwidth requirement, expressed as $(b_e/b_m) - 1$, for three different MPEG traces with $N = 40000$ for variable number of envelope steps, k.

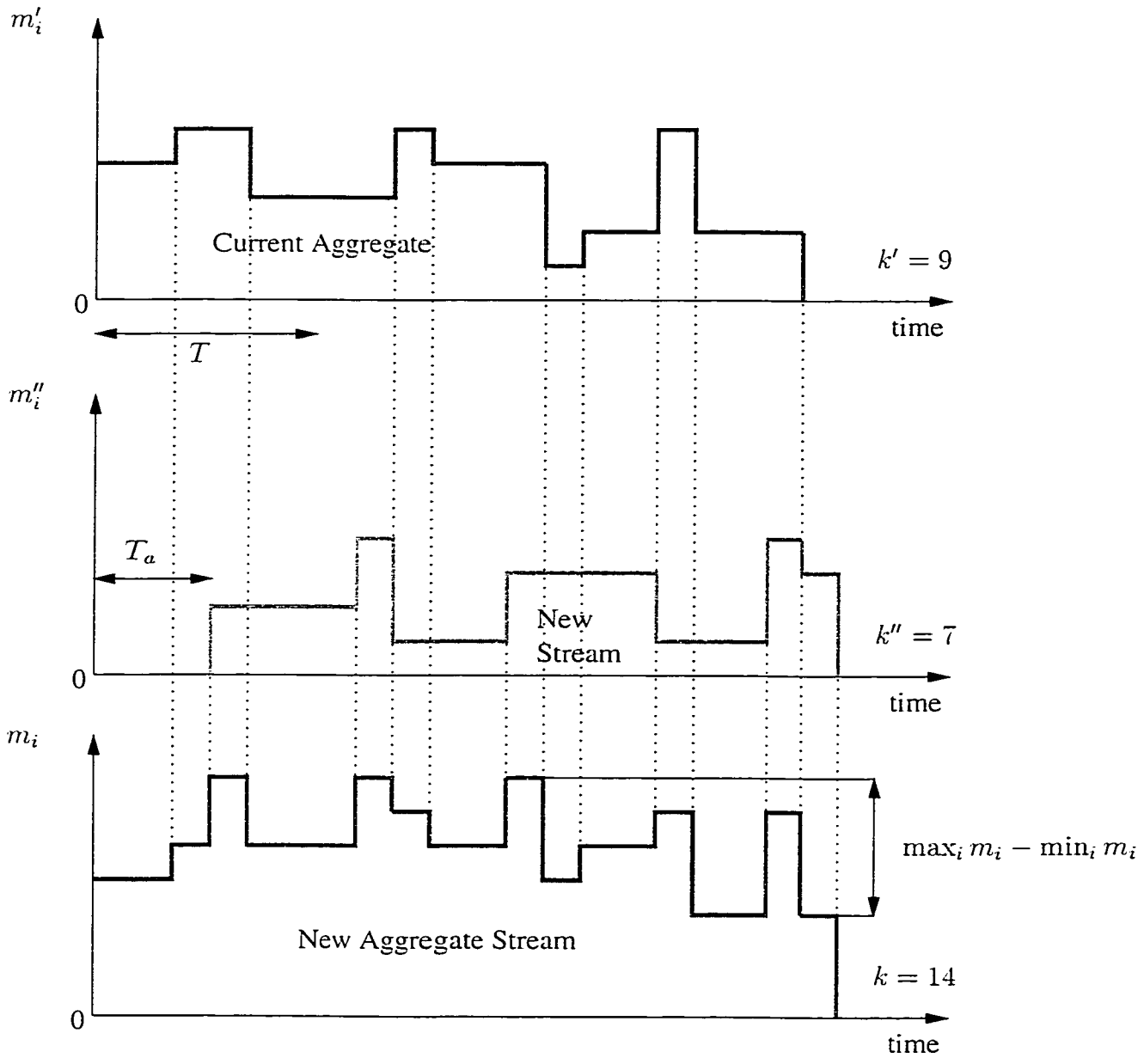


Figure 2.9: Example of the the merging operation of two traffic step–wise envelopes and the resulting step–wise envelope.

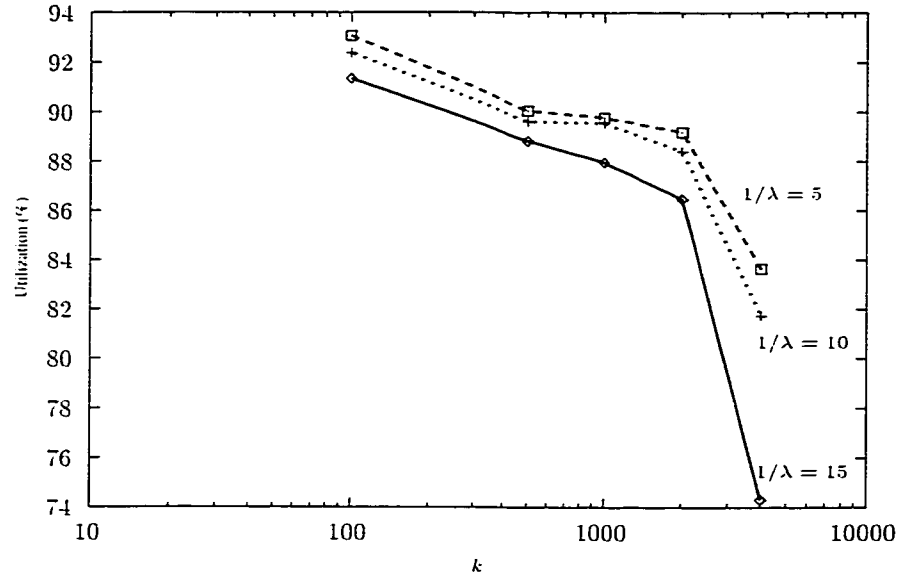


Figure 2.10: Utilization versus envelope steps for three different request interarrival times. ($T = 2000$)

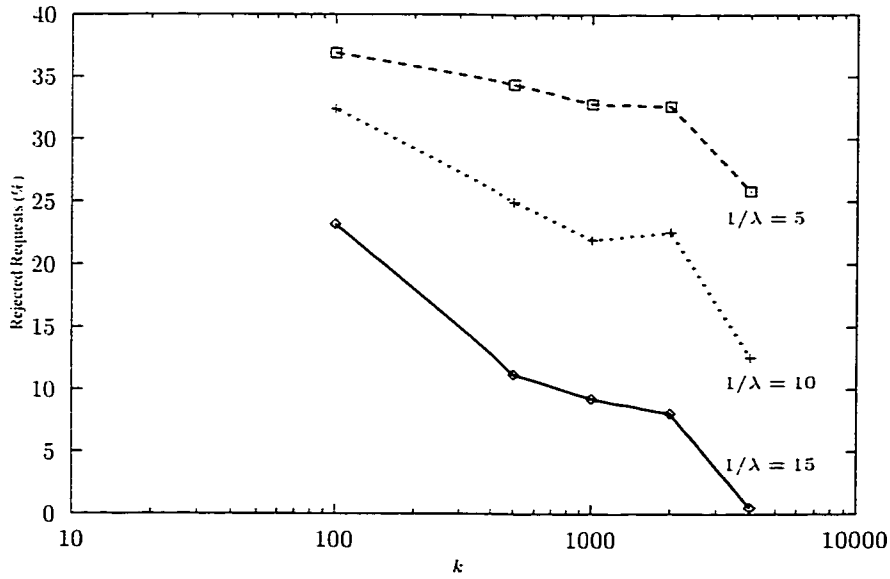


Figure 2.11: Rejected requests versus envelope steps for three different request interarrival times. ($T = 2000$)

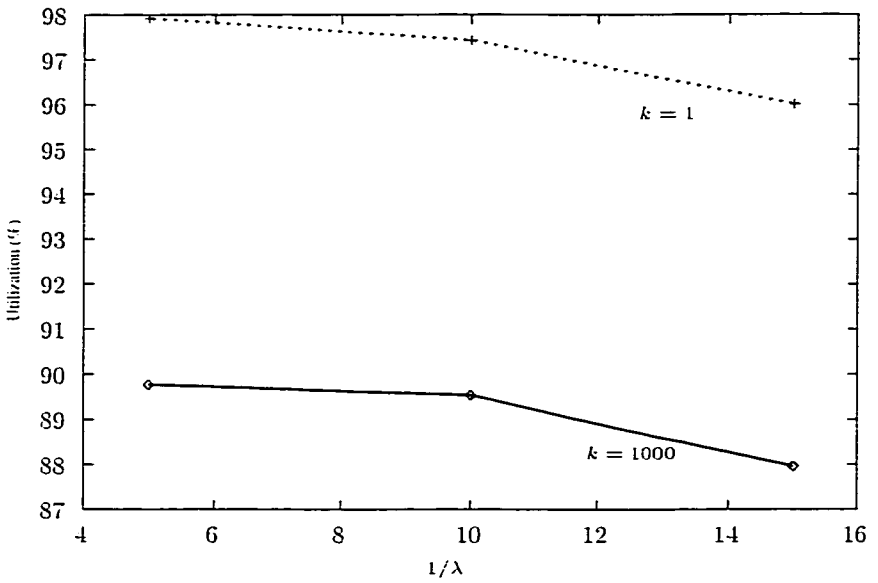


Figure 2.12: Utilization versus request interarrival times for single step ($k = 1$) and 1000-step ($k = 1000$) stream envelopes. ($T = 2000$)

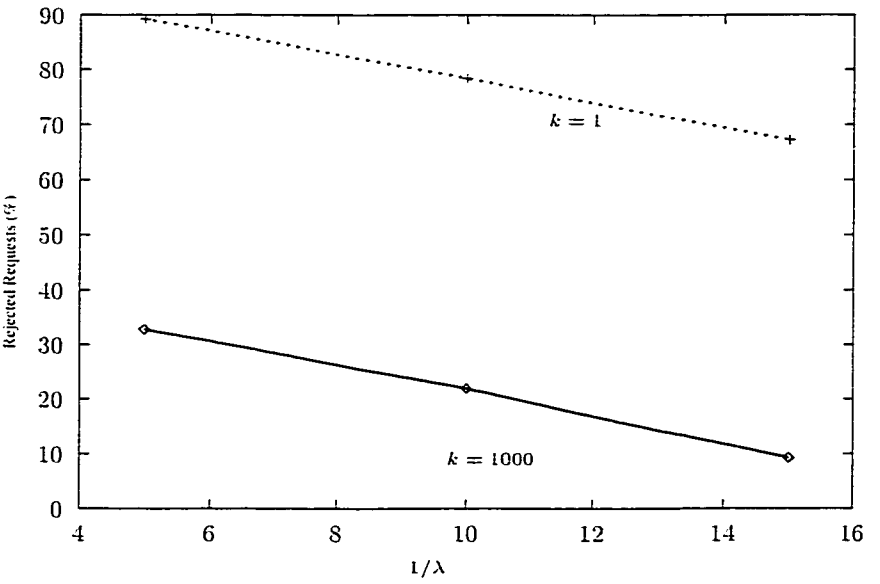


Figure 2.13: Rejected requests versus request interarrival times for single step ($k = 1$) and 1000-step ($k = 1000$) stream envelopes. ($T = 2000$)

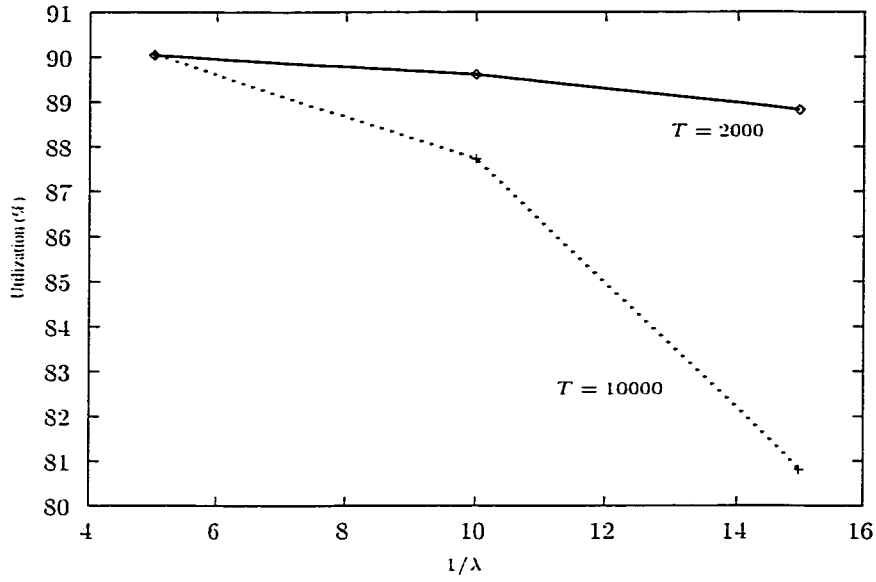


Figure 2.14: Utilization versus request interarrival times for long ($T = 10000$) and short ($T = 2000$) admission interval. ($k = 500$)

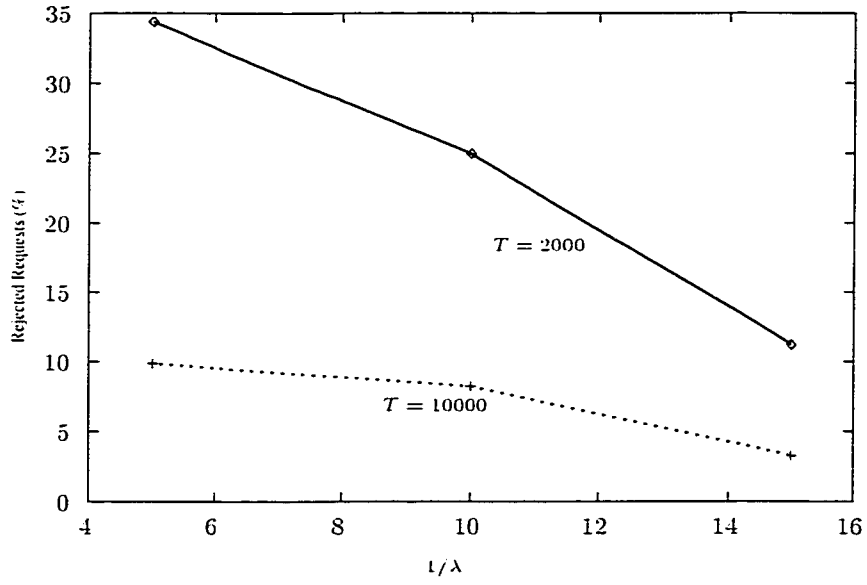


Figure 2.15: Rejected requests versus request interarrival times for long ($T = 10000$) and short ($T = 2000$) admission interval. ($k = 500$)

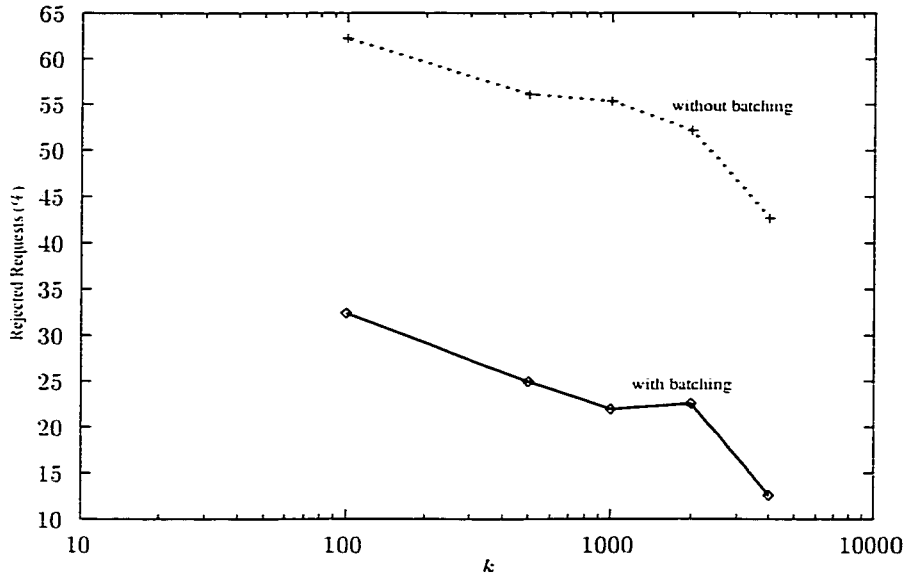


Figure 2.16: Rejected requests versus envelope steps for a system with batching (using $T = 2000$) and one without batching. $(1/\lambda = 10)$.

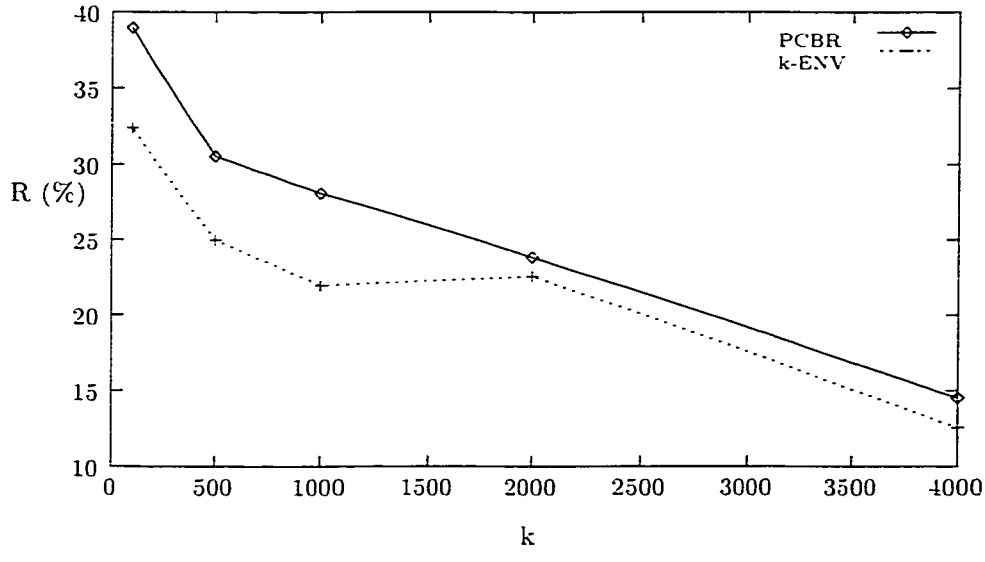


Figure 2.17: Comparison with PCBR for the call admission scheme regarding the requests rejection rate (R) with batching (using $T = 2000$) for variable number of envelope steps (k). The request interarrival time is 10 seconds.

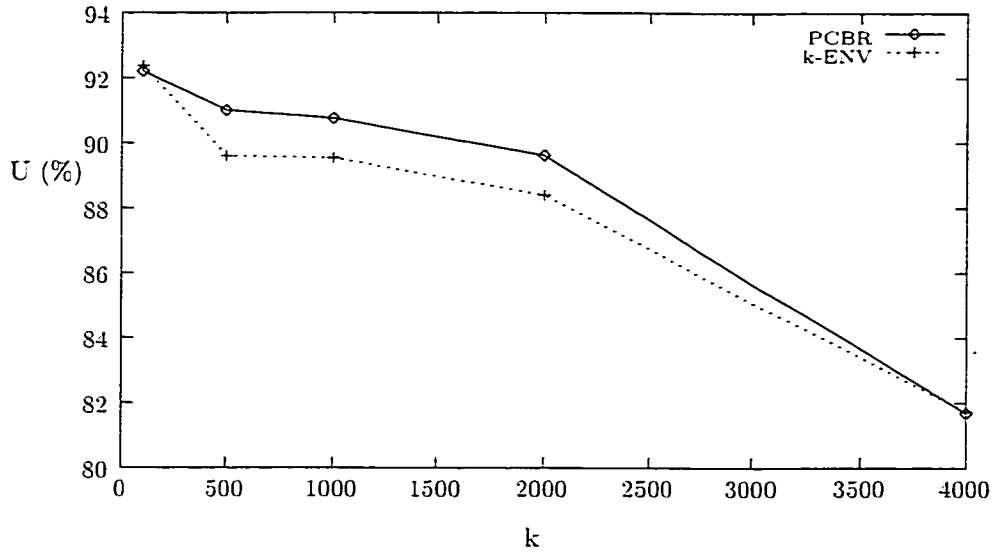


Figure 2.18: Comparison with PCBR for the call admission scheme regarding the bandwidth utilization (U) with batching (using $T = 2000$) for variable number of envelope steps (k). The request interarrival time is 10 seconds.

Chapter 3

Trace-Adaptive Fragmentation Scheme for Periodic Broadcast of VBR Video

The scheduled multicast scheme we have seen can increase the efficiency of VoD systems, however, it is still demanding in terms of bandwidth for a popular movie. Periodic broadcast schemes operate by periodically broadcasting the same video, which can significantly reduce the bandwidth requirement compared with the scheduled multicast schemes for popular videos. One cental point for any periodic broadcast scheme is the way in which the video is fragmented before broadcasting. Due to the high bit rate variability of VBR video, different fragmentations may have very different aggregate traffic when multiplexing the simultaneously transmitted VBR streams, which could result in very different packet loss rate for a limited link capacity. Notably, all of the existing periodic broadcast schemes use rigid fragmentation without regard to packet loss. We are exploring the design of trace-adaptive fragmentation scheme which takes the particular video traffic into account and derives the broadcast schedules to achieve the minimal packet loss rate.

3.1 Basic Concepts and Motivation

In periodic broadcast schemes, the entire video is fragmented into smaller successive and non-overlapping segments and each segment is transmitted on a separate channel with a certain amount of bandwidth. The lengths of the segments, normalized relative to the first segment, is also called the *broadcast series* of the periodic broadcast scheme.

The periodic broadcast schemes require large bandwidth, especially if short startup latency is required. The bandwidth allocated for the transmission of all the segments comprising a video is several times the bandwidth necessary to transmit a single instance of the video from beginning to end at its nominal frame rate (also called *consumption bandwidth* of a video). In addition, the broadcast schemes impose particular synchronization relations between the different segments of the same video. Finally, the bandwidth requirements are further increased by the fact that most periodic broadcast schemes assume Constant Bit Rate (CBR) coded video. It has been observed [9] that, for the same video quality, the average CBR bandwidth demands far exceed the average bandwidth demands of Variable Bit Rate (VBR) coded video. In addition, VBR coding schemes are becoming commonplace (MPEG-1 and MPEG-2) and extensive libraries of video material are already available in VBR form.

The technique presented in this work is suitable for VBR as well as for CBR coded videos. The work is influenced by the Client–Centric Approach (CCA) [18] periodic broadcast schemes and by recent advances towards the periodic broadcast support of VBR encoded videos (VBR-B) [36]. We expand the class of CCA schemes in a way that allows us to derive a set of alternative fragmentations while still satisfying the desired startup latency constraint. The wealth of fragmentation choices provides direct benefits for VBR coded videos. Due to the significant rate variability of VBR-encoded video, different fragmentation schemes could lead to different aggregate traffic shape when multiplexing the periodically broadcast segments together, resulting in different data loss rates. Figure 3.1 illustrates how two alternative 2-segment broadcast schedules of the same VBR video, result in substantially different aggregate traffic patterns, depending on the particular lengths of the segments. Based on this observation, we will focus on a way to generate multiple candidate fragmentations for

the same startup latency requirement. Subsequently, we will select the fragmentation that minimizes the data loss that occurs due to the limited broadcast link capacity. The resulting scheme, called Trace-Adaptive Fragmentation (TAF), combines and improves the properties of [18] and [36].

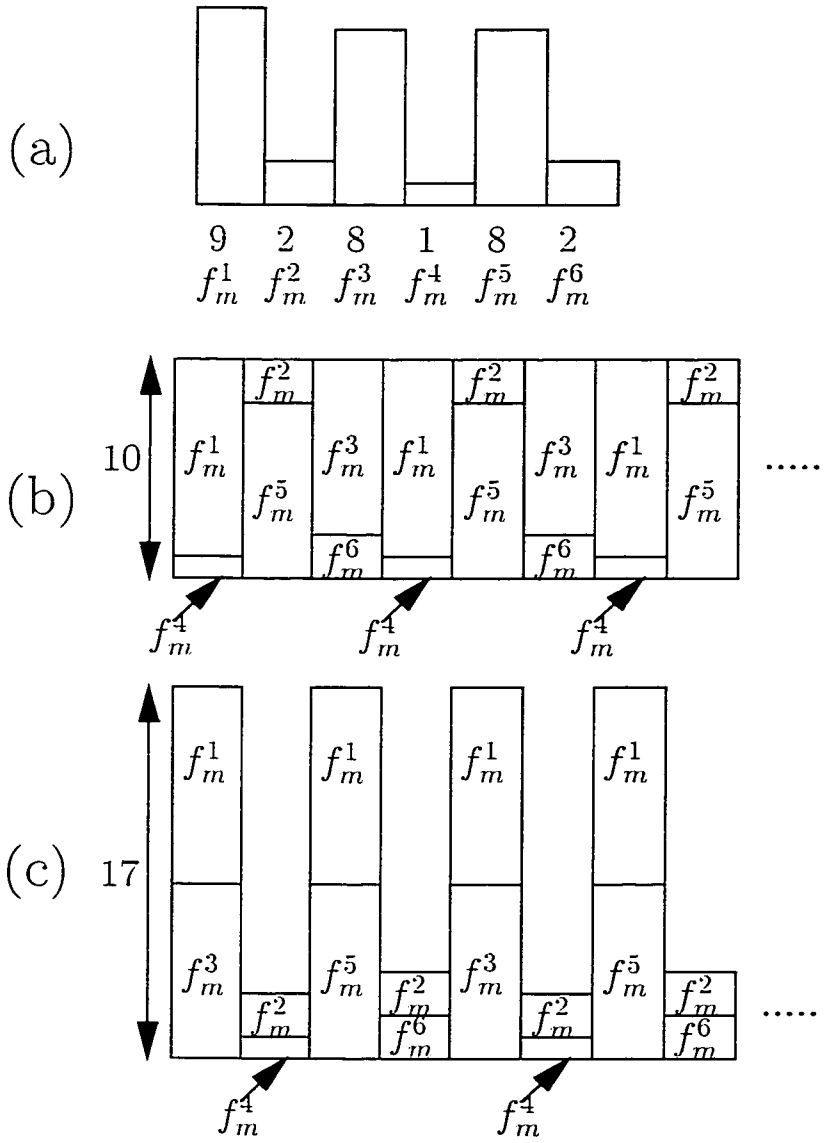


Figure 3.1: An example of a video frame sequence, (a), (the numbers are the frame sizes) and two alternative fragmentation options for the same sequence. In (b) the broadcast series is $\{1, 1\}$ and the aggregate peak is 10 units. In (c) the broadcast series is $\{1, 2\}$ and the aggregate peak is 17 units.

3.2 Scope and Assumptions

In this work we consider only simple VoD service. The key requirement of VoD service is the ability for uninterrupted playout once playout of a video starts. We will not investigate additional (VCR-like) operations such as fast-forward/backward and pause. We note that one can always trivially support VCR-like operations once the entire video is stored on secondary storage.

We will consider the combined broadcast of M videos, with M in the range of 10 to 20 in order to correspond to observed distributions of "hot set" videos [10]. All M videos are multiplexed on the same broadcast physical link of bandwidth B (in Mb/sec). All videos have the same frame rate F (in frames/sec). The frame sequence of each video is fully known a-priori. Let f_m^n, $n = 1, \ldots, N_m$, $m = 1, \ldots, M$ stand for the number of bits in the n^{th} frame of the m^{th} video, where N_m denotes the total number of frames of the m^{th} video. In its general form, the problem of the periodic broadcast of VBR videos can be stated as follows: Given M different videos, characterized by their individual number of frames, N_m, $m = 1, \ldots, M$ and the actual per-video frame sizes f_m^n, $n = 1, \ldots, N_m$ we wish to fragment them for periodic broadcast and transmit them over a link of capacity equal to B Mb/sec.

Several constraints need to be satisfied for the broadcast schedule construction: (a) the startup latency constraint for video m should be less than or equal to w_m seconds, (b) the number of segments video m is split into must be less than or equal to K_m, and (c) the number of concurrently downloaded channels by the receiver must be less than or equal to C_m. Note that K_m limits the bandwidth allocated by a VoD server for the m–th movie. Similarly, C_m limits the bandwidth necessary for the receiver.

We claim that the objective of minimizing the client secondary storage requirements is of minor importance and it will not be taken into consideration in this work. Improvements in storage capacities for magnetic hard disk drives as well as the availability of re-writable optical disks allow now the storage of an entire feature-length video. Instead, we believe that the important restriction is that of the I/O throughput of the secondary storage system. C_m can be used to capture such an I/O throughput constraint. We wish to derive a broadcast series for each of the M videos.The m–th

video broadcast series is represented by $\{s_m^1, s_m^2, \ldots, s_m^{K_m}\}$. Note that $s_m^1 = 1$ and all other s_m^i are normalized relative to s_m^1. A common theme in the current literature is to only consider integer values for s_m^i. One benefit of integer s_m^i appears to be the ability to express all time durations as multiples of a basic unit (typically s_m^1) and hence reduce the complexity of keeping all the segments synchronized with each other. We will thus maintain in this work the assumption of integer s_m^i.

Each segment is broadcast on a separate *logical channel*. The term *logical channel* is used in VoD literature to capture the use of a single physical broadcast link to convey multiple flows of data. Hence, the link capacity is split into several logical channels. The mechanisms through which the sharing of the physical link is achieved will not be detailed. However, note that in the case of CBR video and logical channels of equal capacity, the mechanism required can be straightforward Time Division Multiplexing (TDM). If the logical channels are of different but constant bandwidths, then a Fair Queueing scheduler could be used. In the case of variable bandwidth per logical channel (as in the case of VBR video), the sharing is achieved through statistical multiplexing. Depending on the scheme used, a corresponding jitter absorption scheme is necessary at the receiver. Given that the multiplexing we consider in this work is *bufferless*, no significant jitter absorption is necessary.

Finally, we note that in the construction of a periodic schedule, there exist several choices for optimization criteria, such as minimization of the server bandwidth, minimization of the client buffer size etc. The objective we will use is the minimization of overflow traffic beyond the link capacity, when VBR traffic segments are aggregated. However, by committing to the minimization of data loss, the schedule construction must still be able to capture all the other relevant constraints, i.e., the ones stemming from the synchronization between segments, and the hardware capabilities of the server and clients. How this is accomplished is explained in the next two sections.

3.3 Periodic Broadcast of VBR Video

We follow the definitions of [36] regarding the periodic broadcasting scheme for VBR traffic. Each video m is divided into K_m segments prior to broadcasting. The server broadcasts $\sum_{m=1}^{M} K_m$ video streams simultaneously, one per segment per video.

Frames from $\sum_{m=1}^{M} K_m$ logical channels are multiplexed into the single broadcast link without buffering. Bits are lost whenever the aggregate broadcast traffic rate exceeds the link capacity. The server broadcasts each video segment at a rate F frames per second, the consumption rate of the videos.

A user waits until the next starting point of the first segment. At the next broadcast of the first segment the user begins to receive and concurrently display frames from the beginning of the segment. As with the CBR schemes, the client downloads the remaining segments of the video according to a specific download strategy [19, 18]. The choice of the download strategy depends crucially on the client bandwidth, on the number of channels, C_m, that can be simultaneously received by the client. Part of the download strategy is that, if a segment size is larger than the segment following it, the next segment can always be downloaded to the disk in advance of being needed for playout, resulting in overhead in terms of disk space and disk bandwidth. In this sense, there exists an inherent conflict between start–up delay constraints and disk space or disk bandwidth constraints. If these constraints are tight, there may exist no schedule that simultaneously satisfies all of them.

3.3.1 Startup Latency

Next, we indicate how the startup latency depends on the selected broadcast series. Let $\{ s_m^1, s_m^2, ..., s_m^{K_m} \}$ be a broadcast series for the m^{th} video. Without any loss of generality, set $s_m^1 = 1$. Let N_m^i indicate the number of frames in the i^{th} segment of the m^{th} video. The broadcast series implies that the segment sizes are:

$$N_m^i = s_m^i \times N_m^1, \quad i = 2, \ldots, K_m, m = 1, \ldots, M. \tag{3.1}$$

The size of the first video segment is therefore determined by:

$$N_m^1 = \frac{N_m}{\sum_{i=1}^{K_m} s_m^i}. \tag{3.2}$$

Given a particular fragmentation of the video, the startup latency, is bounded by the time it takes to start receiving the first segment of the video, which in turn is equal to the broadcast duration of the first segment. Let D_m denote the service latency for the m^{th} video. We have:

$$D_m = \frac{N_m^1}{F} \tag{3.3}$$

In general, for a broadcast series $\{s_m^1, s_m^2, \ldots, s_m^{K_m}\}$ where $s_m^1 = 1$ the service latency is given by:

$$D_m = \frac{N_m}{(F \sum_{i=1}^{K_m} s_m^i)} \tag{3.4}$$

According to our assumption, the fragmentation strategy must guarantee a startup latency of w_m seconds maximum for the users before they can watch a video of their choice. From equation (3.4), we have:

$$D_m = \frac{N_m^1}{F} = \frac{N_m}{F \sum_{i=1}^{K_m} s_m^i} \leq w_m \tag{3.5}$$

thus:

$$\sum_{i=1}^{K_m} s_m^i \geq \frac{N_m}{F w_m} \tag{3.6}$$

Clearly, equation (3.6) provides a simple benchmark as to whether a broadcast series satisfies the startup latency constraint.

3.3.2 Bufferless Multiplexing Data Loss

In the bufferless multiplexer model, bits are lost from the video streams if the aggregate amount of traffic that arrives at the link during frame time t exceeds the link's capacity. Let $A_{i,m}^t$ indicate the actual arrival bits sent by the stream of the i^{th} segment of the m^{th} video during frame time t. Let A_t denote the total arrival bits sent by all of the $\sum_{m=1}^{M} K_m$ streams. Then, we have the following:

$$A_t = \sum_{m=1}^{M} \sum_{i=1}^{K_m} A_{i,m}^t \tag{3.7}$$

$$A_{i,m}^t = f_m^j \tag{3.8}$$

where j is given by:

$$j = \sum_{l=1}^{i-1} N_m^l + (t \bmod N_m^i) \tag{3.9}$$

Notably, j stands for the index for the frame of the m^{th} video that is sent during frame time t. Thus, loss occurs in frame time t if:

$$A_t > B/F \tag{3.10}$$

Let P_{loss} denote the long-run fraction of traffic loss, we have:

$$P_{loss} = \lim_{T\to\infty} \frac{\sum_{t=1}^{T}(A_t - B/F)^+}{\sum_{t=1}^{T} A_t} \tag{3.11}$$

Equation (3.11) provides the definition of the data loss which is the quantity we wish to minimize. Note that P_{loss} is defined in terms of data loss in units of bits. However, we consider the bit loss ratio as a sufficiently accurate approximation for the packet loss ratio when the packets are small, e.g. if ATM cells were to be used as the underlying means of conveying the segments.

3.4 The Trace–Adaptive Fragmentation (TAF)

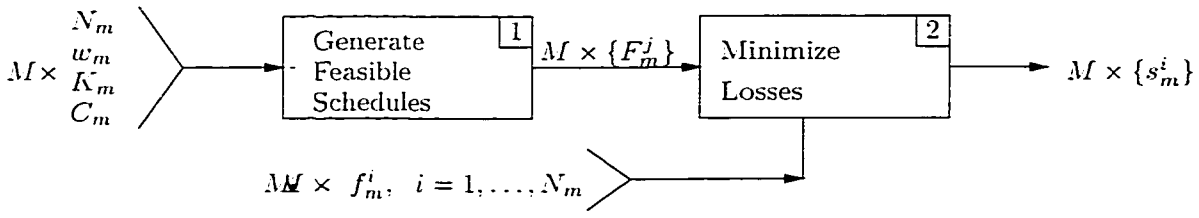


Figure 3.2: Block diagram of TAF.

The Trace–Adaptive Fragmentation is summarized in the block diagram of Figure 3.2. In the first stage, a number of alternate fragmentations are found that satisfy the per-video delay constraints. In the second stage, one fragmentation is selected for each video in such a way that combined traffic of all the videos under the selected fragmentations minimizes the lost data. Note that in Figure 3.2 the input consists of the C_m, K_m, w_m and N_m parameters for each one of the M videos. N_m characterizes the corresponding video frame sequence but the rest (C_m, K_m, w_m) are all constraints of the VoD system. That is, at the first stage, the actual frame sequence plays no role, except through its total number of frames.

3.4.1 Continuity Constraint and Segment Sizes

The segments must be transmitted in such a relation to each other that the *continuity condition* holds. The continuity condition ensures that no starvation will occur at the client once it starts to consume the video stream and until the end of the video.

We know that the client can download the video data from $C_m (2 \leq C_m \leq K_m)$ streams simultaneously. If C_m is equal to 1, then all of the segments have to be equally-sized in order to guarantee the continuity condition. Immediately after the client process begins to download the video segments, the user can start playing back the video at its normal consumption rate of F frames per second in the order [1] $S_m^1 \bullet S_m^2 \bullet S_m^3 \ldots S_m^{K_m}$.

We follow a similar approach taken by CCA [18], that is, to receive and play-back the data fragments, a client uses $C_m + 1$ service routines: C_m data loaders, $L_1, L_2, \ldots L_{C_m}$, and one video player.A multi-threaded client multiplexes itself among these routines. Each data loader can download data at the consumption rate. The data segments are downloaded in G_m rounds. During each round, each of the C_m loaders is responsible for downloading its respective data segment in a certain transmission group, say the r^{th} group, at its *earliest* occurrence. When the download of the current group has been completed, the loaders proceed to download the next transmission group, i.e., $(r+1)^{th}$ group, in the same manner.

The solution we adopt operates by fragmenting each video file into K_m segments and partitioning the K_m segments further into G_m transmission groups, where G_m is given by $\lceil K_m/C_m \rceil$. Each group contains C_m segments except for the last group that contains $K_m - (\lceil K_m/C_m \rceil - 1) \times C_m$ elements. Specifically, the i–th transmission group of the m^{th} video contains the segments from $S_m^{(i-1)C_m+1}$ up to $S_m^{iC_m}$ (inclusive).

Given the above definitions, the segment sizes in each group possess the following properties:

1. a segment is greater than or equal to the previous segment;

2. the size of the last segment of a group must be equal to the size of the first segment of the next group;

[1] Note that S_m^i is notation representing a set: the frames of the i–th segment of the m–th video, while s_m^i is an integer: the length of the i–th segment of the m–th video relative to segment $s_m^1 = 1$.

3. the size of the segment in each group is an integer multiple of the size of the first segment of the group.

We now introduce X_m^i to indicate the maximum segment size for the i^{th} segment of the m^{th} video. X_m^i represents the maximum allowable length for a segment in order to guarantee, that regardless when this segment starts relative to all the other segments of the same group, it can provide uninterrupted playout. The definition of X_m^i relies on the knowledge of how much time is necessary for consuming all the segments the precede it in the same group:

$$X_m^i = \begin{cases} 1 & i = 1, \\ s_m^{i-1} & i \bmod C_m = 1, \\ s_m^{C_m\lfloor i/C_m \rfloor + 1} + \sum_{j=C_m\lfloor i/C_m \rfloor + 1}^{i-1} s_m^j & i \bmod C_m \neq 1. \end{cases} \tag{3.12}$$

for $2 \leq i \leq K_m$. (Note that, $s_m^{C_m\lfloor i/C_m \rfloor + 1}$, represents the first segment of the group in which segment i belongs.) In fact, the definition of X_m^i restates the fragmentation principle of CCA [18] and its correctness can be proven using exactly the same arguments as for CCA. The difference is that the X_m^i is only the upper bound on the length of a segment. The correctness (continuity) holds even if the segments are chosen to have length less than their corresponding X_m^i. In such case, starting to store the earliest occurrence of a segment may not be the most buffer-efficient approach, but we are not concerned with the size of secondary storage at the client. Overall, if $s_m^i \leq X_m^i$, each segment is guaranteed to have become available (started transmission) once or more times while the preceding segments were being consumed.

For $s_m^i = X_m^i$, the scheme specializes to CCA. If in addition $K_m = C_m$, then the scheme specializes to the the geometric broadcast series $\{1, 2, 4, \ldots\}$ which has been used in the VBR-B scheme [36].Furthermore, as in SB, we can use a constant W to restrict the size of the largest segment from becoming too large. W solves the so-called *big-segment problem.* If a segment is larger than W times the size of the first segment, we force it to be exactly W units.

3.4.2 An Enumeration Algorithm

We present an enumeration algorithm which forms the first component of the TAF scheme (Figure 3.2). The input of the algorithm are parameters K_m, C_m, N_m and w_m

and its output are all feasible broadcast schedules that conform to the startup latency and the continuity constraints. The enumeration starts with the base case where all segments are of equal length. Very likely, such a configuration will fail with respect to the delay constraints. The length of the segments is increased gradually starting from the last segment. If the last segment, even after increased to its maximum, cannot satisfy the delay constraint, an increase of the second segment from the end can start. The process continues until it becomes necessary to increase s_m^2 to more than 2 (since, for $s_m^2 = 3 \geq X_m^2$ the continuity cannot hold).

The algorithm is presented in Figure 3.3. An example run of the algorithm is shown in Figure 3.4 where only five feasible solutions are found including the one that CCA constructs (F_m^5). We denote the set of feasible fragmentations of the m–th video as $\{F_m^j\}$. The generation of the feasible schedules (Figure 3.2) is performed independently for each one of the M videos. Note that the set of feasible schedules is potentially large. From Figure 3.4 we also extract the instance (which fails the startup latency constraint) with broadcast series $\{1, 2, 3, 3, 6, 6\}$ and present it in Figure 3.5 to illustrate the temporal relation between the segments as constructed by the first stage of the TAF scheme.

3.4.3 Loss Minimization

The next step is the selection of the optimal broadcast schedule for each video, given the feasible schedules for M videos (stage 2 of Figure 3.2). At this point, we can follow a computationally expensive process of determining the aggregate traffic of all possible combinations of feasible video schedules to determine the one that minimizes the loss rate. For M videos, the combinations that need to be examined are:

$$\prod_{m=1}^{M} |\{F_m^i\}| \tag{3.13}$$

This approach requires extensive computation (depending on the set of feasible solutions), especially if the computation of the data loss for each combination is computationally expensive as well. Since all computation is performed off–line, one can claim that processing power is not going to be an issue. However, we can approximate the optimal selection by picking for each video the feasible schedule which results

in the minimum peak rate. Thus, the aggregation of all the individual schedules is likely to produce traffic which has a small peak rate as well, and hence small loss ratio when multiplexed. In the presented experiments we use this particular approximation instead of the optimal solution.

At this point we must note that the computation of the data loss ratio of equation (3.11) can be simplified. We refer to the *least common multiple* as LCM for brevity. Let:

$$\tilde{s_m} = LCM\{s_m^1, s_m^2, \ldots, s_m^{K_m}\} \tag{3.14}$$

$$\tilde{s} = LCM\{\tilde{s_1}, \tilde{s_2}, \ldots, \tilde{s_M}\} \tag{3.15}$$

$$\tilde{N} = LCM\{N_1^1, N_2^1, \ldots, N_M^1\} \tag{3.16}$$

$$\tilde{T} = \tilde{s}\tilde{N}. \tag{3.17}$$

Due to the periodic nature of the broadcast strategy, the aggregate traffic of the $\sum_{m=1}^{M} K_m$ segments is also periodic with a period of $\tilde{T}$ frame times. To determine the data loss rate, we only need to observe the traffic during the first $\tilde{T}$ frame times. According to the relationship among successive segment sizes expressed in equation (3.1), we observe that:

$$\begin{aligned}
LCM \quad & \{N_1^1, N_1^2, \ldots, N_1^{K_1}, \\
& N_2^1, N_2^2, \ldots, N_2^{K_2}, \ldots, \\
& N_M^1, N_M^2, \ldots, N_M^{K_M}\} = \\
LCM \quad & \{s_1^1 \times N_1^1, s_1^2 \times N_1^1, \ldots, s_1^{K_1} \times N_1^1, \\
& s_2^1 \times N_2^1, s_2^2 \times N_2^1, \ldots, s_2^{K_2} \times N_2^1, \ldots, \\
& s_M^1 \times N_M^1, s_M^2 \times N_M^1, \ldots, s_M^{K_M} \times N_M^1\} = \\
LCM \quad & \{\tilde{s_1} \times N_1^1, \tilde{s_2} \times N_2^1, \ldots, \tilde{s_M} \times N_M^1\} = \\
LCM \quad & \{\tilde{s_1}, \tilde{s_2}, \ldots, \tilde{s_M}\} \times \\
LCM \quad & \{N_1^1, N_2^1, \ldots, N_M^1\} = \\
\tilde{s}\tilde{N} \quad & = \tilde{T}
\end{aligned} \tag{3.18}$$

Thus, to compare the packet loss rates of different fragmentations of the video files, we only need to calculate the packet loss rate in the first $\tilde{T}$ frame times. Hence, given a particular selection of broadcast series for each one of the M videos we can produce an estimate of the data loss probability in time $O(\tilde{T})$.

In the calculation of the loss probability, the frame sizes of the video traces are necessary, thus the need to include the information about f_m^i in the second stage of TAF (Figure 3.2). Therefore, the total space requirements for keeping track of the

frame sizes is $O(\sum_{m=1}^{M} N_m)$. Note that CBR coded videos can be included trivially ($f_m^i = constant.$) allowing TAF to produce schedules for a mixture of VBR and CBR coded videos.

3.5 Experimental Results

The approximation introduced in the previous section selects for each video the feasible schedule that minimizes the peak bitrate of the aggregated bandwidth (over all the segments) for the specific video trace. The computational complexity of identifying the per–video peak bitrate for a given broadcast series is $O(LCM(N_m^1, N_m^2, \ldots, N_m^{K_m}))$, which using the notation of the previous section, becomes $O(N_m^1 \tilde{s_m})$.

Figure 3.6 provides the peak bitrate found for each one of 10 example video traces of 40.000 frames each (see [35] for the origin of the traces) depicting diverse material, including feature movies, TV news, sporting events etc. The set of feasible schedules explored were the ones produced in Figure 3.4. Observe that the peak bitrate depends drastically on the selection of the feasible broadcast schedule. For example, trace `talk_2` exhibits peak bitrates from a maximum of 12.26 Mb/s (for F_m^4) down to almost half the maximum at 6.19 Mb/s (for F_m^3). Notably, the broadcast series used by CCA, F_m^5, provides the minimum peak bitrate for only two of the ten examined traces. The numbers in parentheses in Figure 3.6 are the client peak bandwidth requirements. That is, the peak of the aggregate traffic received by downloading simultaneously $C_m = 3$ segments of the same group. It should be pointed out that for VBR traffic, the peak experienced by a set-top box depends on the random instant at which it started downloading. In other words, a set-top box does not necessarily remain active downloading C_m channels for $N_m^1 \tilde{s_m}$ frame times. In fact, the entire video can be completely downloaded in much lesser time. Thus, the reported peak bitrate for a client is a worst–case unrealistic scenario. Nevertheless, Figure 3.6 clearly illustrates that reduced client bandwidth is achievable by limiting the number of channels that can be simultaneously downloaded. It must be emphasized that in the presentation of the experimental results we have taken a server–centric view. Hence the selection criterion we apply for a broadcast schedule is to exhibit the lowest peak bitrate. One can use a client-oriented view which encourages the selection of schedules that

minimize the client download bandwidth. We can see from Figure 3.6 that doing so (thus selecting the schedules with the boldface parenthesized values) does not always result in low bandwidth demands for the server. What is a good for the client set-top box is frequently not good for the server.

Another angle to view the same results is to consider the fact that given several feasible schedules, external factors can be incorporated in the selection of the best schedule. For example, in systems with heterogeneous set-top boxes, it is possible to determine which fragmentation is better used for which set-top boxes, possibly transmitting the same video in two or more different fragmentation formats, one for each set-top box technology.

3.5.1 Comparison of VBR-B and TAF

We conduct a comparison between TAF and VBR-B using the same set of 10 selected MPEG traces. However, to provide a basis for the comparison, no limit on the number of downloading channels is imposed ($C_m = K_m$). We are already aware that the search for feasible schedules of TAF subsumes the schedules produced for VBR-B (geometric series). At the same time, TAF uses an approximate search (instead of the exhaustive search implied by equation (3.13)). Therefore, it is not at all clear whether the approximate TAF performs consistently better than VBR-B. For this reason, we perform a set of additional experiments comparing directly TAF and VBR-B.

The lower peak bitrate for TAF compared to that of VBR-B is readily confirmed in Figure 3.7. The differences are not always spectacular and they are sensitive to the particular video trace. However, once multiplexed on the same link, the modest differences in peak bitrate on a per-video basis compound resulting in distinctly different loss behavior as Figure 3.8 reveals. From the results in Figure 3.8, we can see that for the same packet loss rate, TAF can save as much as 30% server link bandwidth compared to VBR-B; for the same server link capacity, the packet loss rate can be reduced by a factor of near 100 with TAF compared to VBR-B in this example. Clearly TAF provides a performance advantage in this environment.

To determine the effect of server bandwidth on performance (Figure 3.8), we varied the link capacity B between 50 Mb/sec and 85 Mb/sec while the maximum startup latency, w_m, was set to a borderline small value of 16.5 seconds. The startup latency

pushed the scheduling to its limits forcing particularly small first segments and much lengthier subsequent ones. Because the number of segments was small, $K_m = 7$, the limit value $W = 32$ was used to force VBR-B to not create too large segments and to match exactly the startup latency objective. A similar intervention on TAF did not prove necessary since it identified several feasible schedules satisfying (or even exceeding) the requirements of the startup latency constraint.

Similar to bufferless multiplexing, we can compare VBR-B to TAF assuming buffered multiplexing at the server. That is, all streams sent to the clients are fed to a common buffer before transmission by the server. The results for this case are shown in 3.9. It is clear that for large enough buffer size no appreciable loss is present, but at the same time the arrival jitter at the clients can be increased. Thus, we provide this example only to point out the flexibility of TAF which is in all aspects similar to the flexibility of VBR-B while providing better loss rate results. Note that Figures 3.8 to 3.11 are plotted using log–scale for the y–axis to capture the wide range of values over which the loss probability spans for even small changes in system parameters (e.g. link capacity B). Hence, seemingly small differences in Figure 3.9 between TAF and VBR-B are quite substantial in terms of absolute numbers.

The Join-the-Shortest Queue prefetching scheme by Reisslein and Ross in [34] can also be used by the server to force the clients from any of the ongoing video streams to prefetch video frames and to send the prefetched frames to the buffers in the appropriate clients to fully utilize the shared links' bandwidth (when the link is idling due to the VBR nature of the multiplexed video segments). It is assumed that each stream has a virtual buffer and the one with the shortest queue has the highest priority to prefetch one more frame if the aggregated bandwidth is not currently over the shared links' capacity. We refer the readers to [34] for the details of the JSQ prefetching policy. Figure 3.10 presents the results produced by TAF and VBR-B with the addition of the Join the Shortest Queue (JSQ) policy. Even in this case TAF has an advantage although its potential for improvement is reduced w/ increasing virtual buffer size. This is partly because of the more "compact" aggregate traffic in the case of TAF compared to VBR-B. That is, by avoiding extreme values in the required aggregate bandwidth, TAF presents less opportunities to prefetch using JSQ.

A final extension of the basic scheme is the inclusion of Group of Picture (GOP)

smoothing to the broadcast schedule computation. Smoothing drastically reduces the variance of the aggregate bandwidth, and this in turn reduces the differences of the peak bitrate of the different candidate fragmentations for TAF. Thus the selection of the minimum peak bitrate becomes less consequential to the loss rate performance (Figure 3.11). Remarkably, even under settings that smooth and "reshape" the traffic (such as buffered multiplexing, JSQ–prefetch and GOP–smoothing) TAF still provides a consistently better performance than VBR-B.

```
for i = 1, ..., K_m do
    s_m^i = 1;
    calculate X_m^i per equation (3.12);
endfor
while s_m^2 ≤ X_m^2 do
        if equation (3.6) satisfied then
          output s_m^i ∀i  // Feasible fragmentation.
        endif
        s_m^{K_m} = s_m^{K_m} + s_m^{C_m ⌊K_m/C_m⌋+1};
        for i = K_m, ..., 2 do
            if s_m^i ≥ X_m^i then
              if (i − 1) mod C_m = 1 then
                 s_m^{i−2} = s_m^{i−2} + s_m^{C_m ⌊(i−2)/C_m⌋+1};
                 s_m^{i−1} = s_m^{i−2};
              else
                 s_m^{i−1} = s_m^{i−1} + s_m^{C_m ⌊(i−1)/C_m⌋+1};
                 for j = i, ..., K_m do
                     calculate X_m^i per equation (3.12);
                     s_m^j = s_m^{i−1};
                 endfor
              endif
            else
              break
            endif
        endfor
endwhile
```

Figure 3.3: Pseudocode for the generation of all possible feasible broadcast series that guarantee the delay and continuity condition for video m.

s_m^1	s_m^2	s_m^3	s_m^4	s_m^5	s_m^6	Feasible?
1	1	1	1	1	1	No
1	1	1	1	1	2	No
1	1	1	1	1	3	No
1	1	1	1	2	2	No
1	1	1	1	2	3	No
1	1	1	1	2	4	No
1	1	2	2	2	2	No
1	1	2	2	2	4	No
1	1	2	2	2	6	No
1	1	2	2	4	4	No
1	1	2	2	4	6	No
1	1	2	2	4	8	No
1	1	3	3	3	3	No
1	1	3	3	3	6	No
1	1	3	3	3	9	No
1	1	3	3	6	6	No
1	1	3	3	6	9	No
1	1	3	3	6	12	No
1	2	2	2	2	2	No
1	2	2	2	2	4	No
1	2	2	2	2	6	No
1	2	2	2	4	4	No
1	2	2	2	4	6	No
1	2	2	2	4	8	No
1	2	3	3	3	3	No
1	2	3	3	3	6	No
1	2	3	3	3	9	No
1	2	3	3	6	6	No
1	2	3	3	6	9	No
1	2	3	3	6	12	Yes (F_m^1)
1	2	4	4	4	4	No
1	2	4	4	4	8	No
1	2	4	4	4	12	Yes (F_m^2)
1	2	4	4	8	8	Yes (F_m^3)
1	2	4	4	8	12	Yes (F_m^4)
1	2	4	4	8	16	Yes (F_m^5)

Figure 3.4: Example run of the enumeration algorithm, for $K_m = 6, C_m = 3, F = 25$ frames/s, $N_m = 40,000$ frames. F_m^1 to F_m^5 are the five feasible fragmentations that satisfy $w_m \leq 60$ sec.

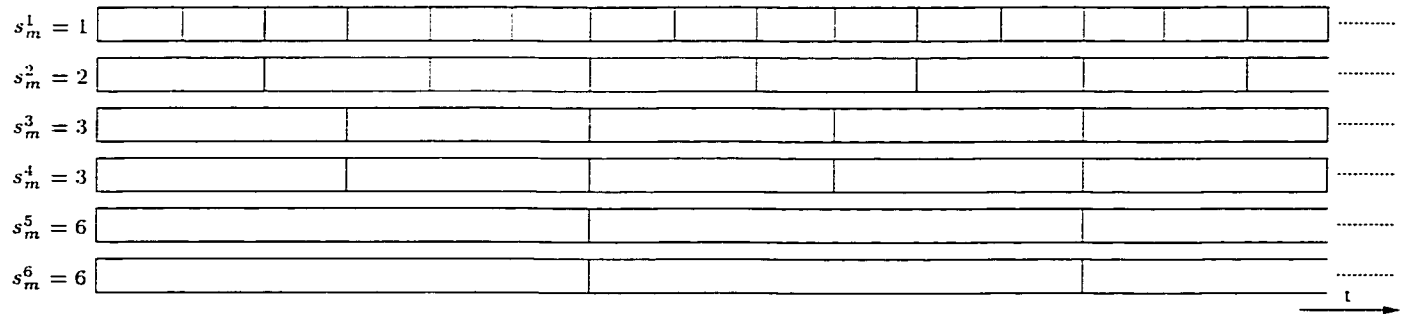


Figure 3.5: Example timing of the segments of the broadcast series $\{1, 2, 3, 3, 6, 6\}$ generated for $C_m = 3$, $K_m = 6$.

Video	F_m^1		F_m^2		F_m^3		F_m^4		F_m^5	
mtv_1	14.8	(11.9)	10.1	(8.1)	10.2	(8.1)	16.5	(10.1)	**10.0**	**(7.9)**
mtv_2	11.3	(9.5)	**9.9**	(8.5)	**9.9**	(8.5)	14.3	(9.8)	10.3	**(7.1)**
race	12.5	(8.4)	11.1	**(7.1)**	**10.5**	**(7.1)**	16.7	(9.1)	11.0	(7.8)
talk_1	8.5	(6.0)	7.9	(5.4)	7.4	**(4.9)**	13.7	(7.1)	**7.4**	(5.4)
talk_2	7.8	(5.0)	7.2	(4.5)	**6.2**	**(4.4)**	12.3	(6.4)	6.9	(4.5)
simpsons	10.1	**(6.9)**	**9.6**	(6.9)	**9.6**	(6.9)	14.6	(8.1)	10.1	(7.3)
terminator	5.4	(3.5)	4.5	**(3.0)**	**4.1**	**(3.0)**	7.9	(4.6)	5.0	(3.2)
soccer_1	12.0	(8.9)	11.0	(7.1)	**10.1**	(7.1)	17.8	(9.8)	10.6	**(6.7)**
soccer_2	10.0	(8.4)	10.3	**(7.4)**	**10.0**	(7.4)	15.2	(9.2)	13.7	(8.8)
news_2	8.6	(7.0)	7.9	**(5.7)**	**7.7**	**(5.7)**	14.2	(8.4)	9.2	(6.8)

Figure 3.6: Peak aggregate transmission (reception) bitrate in Mb/sec of the multiplexed periodic broadcast traffic for each feasible schedule of Figure 3.4 for 10 video traces taken from [35]. The numbers in boldface correspond to the minimum peak aggregate bitrate for each video over all the five feasible fragmentations. ($w_m \leq 60$ sec, $C_m = 3$, $K_m = 6$, $N_m = 40,000$ frames, $F = 25$ frames/sec).

Video	VBR-B	TAF
mtv_1	11.7	9.9
mtv_2	11.8	9.2
race	12.9	11.2
talk_1	9.5	7.3
talk_2	9.1	6.7
simpsons	11.4	9.1
terminator	5.9	4.6
soccer_1	13.5	10.3
soccer_2	12.8	10.4
news_2	10.5	7.9

Figure 3.7: Peak aggregate bitrate (in Mb/sec) of the multiplexed periodic broadcast traffic for VBR-B and TAF for 10 video traces taken from [35] ($w_m \leq 16.5$ sec, $C_m = K_m = 7$, $N_m = 40,000$ frames, $F = 25$ frames/sec).

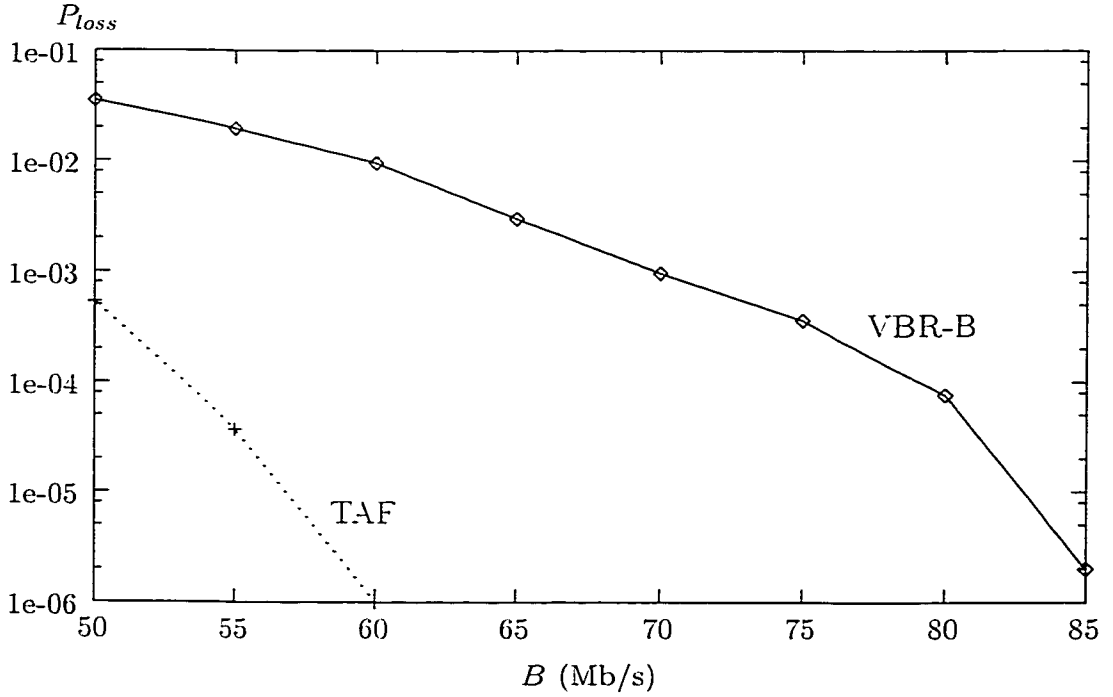


Figure 3.8: Comparison of the loss rate for variable B between VBR-B and TAF.The 10 videos of Figure 3.7 were used. ($w_m \leq 16.5$ sec, $C_m = K_m = 7$, $N_m = 40,000$ frames, $F = 25$ frames/sec).

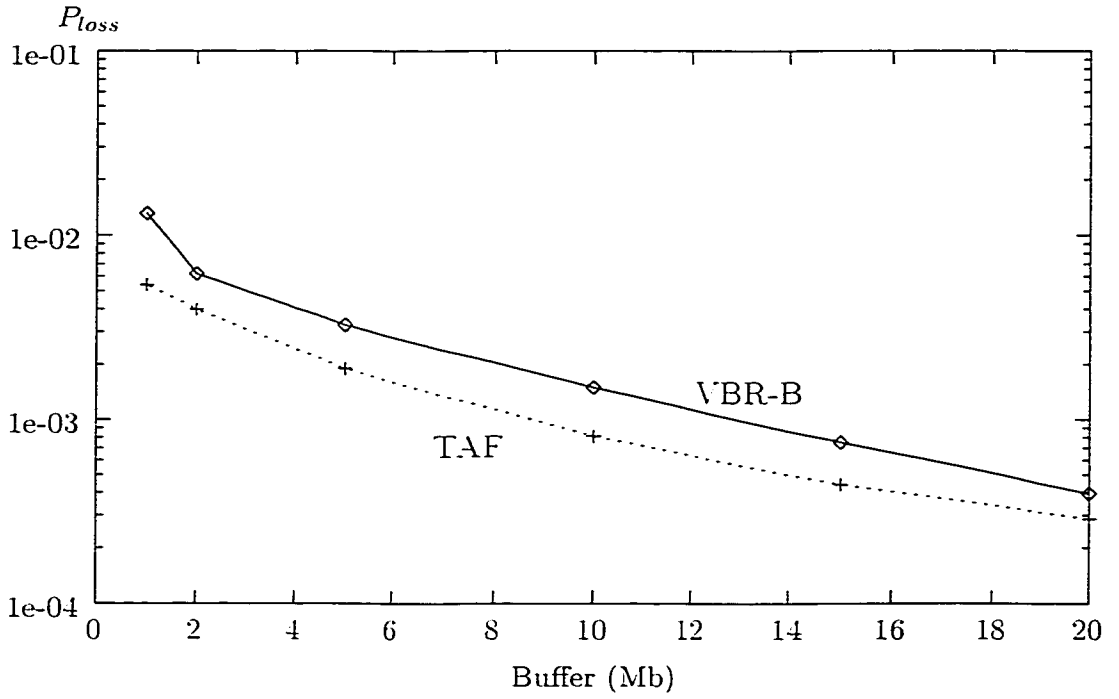


Figure 3.9: Comparison of loss rate between VBR-B and TAF using buffered multiplexing for variable shared buffer size at the server. The 10 videos of Figure 3.7 were used. ($w_m \leq 16.5$ sec, $C_m = K_m = 7$, $N_m = 40,000$ frames, $F = 25$ frames/sec, $B = 38$ Mb/sec).

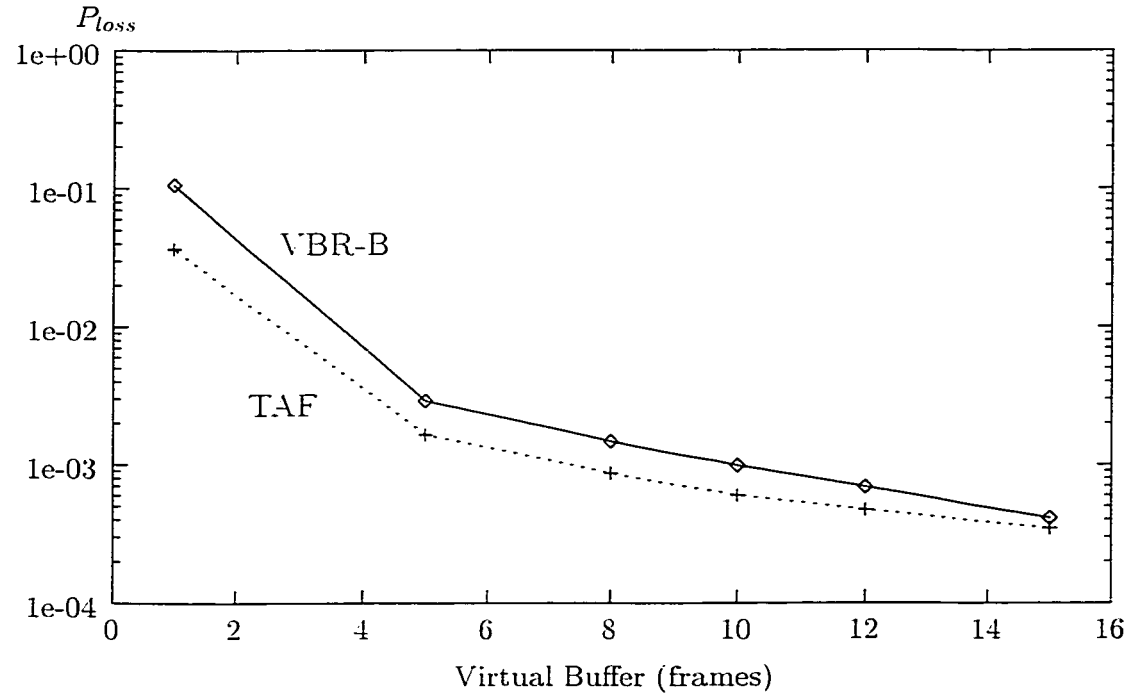


Figure 3.10: Comparison of loss rate for variable virtual buffer size between VBR-B and TAF using JSQ prefetching for the allocation of the spare bandwidth after multiplexing. The 10 videos of Figure 3.7 were used. ($w_m \leq 16.5$ sec, $C_m = K_m = 7$, $N_m = 40,000$ frames, $F = 25$ frames/sec, $B = 38$ Mb/sec).

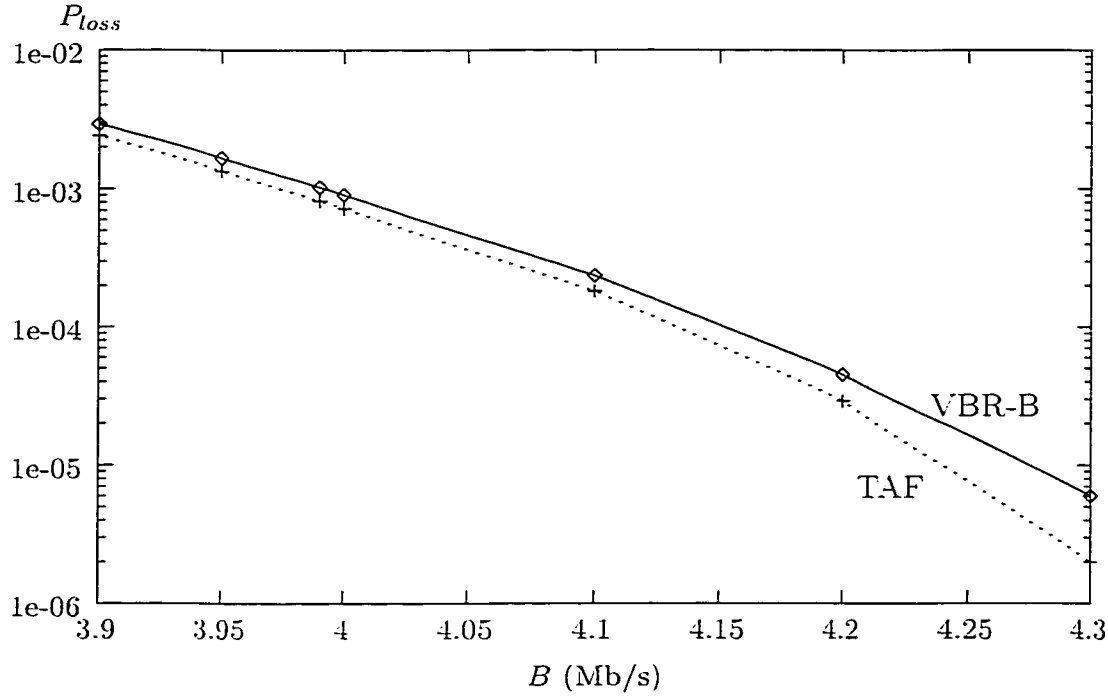


Figure 3.11: Comparison of loss rate for variable B between VBR-B and TAF using GOP–smoothing over periods equal to a single GOP (12 frames). The 10 videos of Figure 3.7 were used.($w_m \leq 16.5$ sec, $C_m = K_m = 7$, $N_m = 40,000$ frames, $F =$ 25 frames/sec).

Chapter 4

A Loss–Less Bandwidth–Efficient Protocol for Periodic Broadcast of VBR Video

So far, we have seen the batched multicast schemes and "lossy" periodic broadcast schemes for VBR video distribution. Notably, all of the existing schemes for VBR broadcast assume packet loss due to multiplexing of VBR sources. We are exploring a loss-less VBR broadcast scheme while requring the least amount of bandwidth. This can be possible if we equate the server and client bandwidth and then minimize the server/client bandwidth. In addition, we assume the client can download the video data immediately after he/she tunes in to request a particular video. Once the first segment is fully downloaded by the client, he/she can start the playout uninterruptly. In other words, we impose a fixed delay for every users, which is the time to download the first segment of the video. We can take advantage of this to achieve more bandwidth efficiency.

4.1 Design Assumption

The proposed scheme in the context of periodic broadcast differs from previous work in two crucial assumptions:

1. *The client-side set-top box secondary storage capacity is not considered a constraint.* Recent price drops for secondary storage devices, such as magnetic disks, as well as the gradual availability of re-writable DVD media, indicates that commodity priced set-top boxes will likely contain storage in the range of several Gigabytes. We thus believe that in the near future any set-top box with a basic set of capabilities will be sufficiently equipped for the off-line storage of complete feature-length (approximately 2 hours) videos. Subsequently, we will assume that the client set-top box is capable to store a large percentage, up to 100 %, of a video.

2. *Reception of a broadcast segment does not delay until the start of the segment transmission, but can start as soon as the client set-top box "tunes-in".* Previous schemes insist on forcing the reception of a segment to start at the exact beginning of a segment, and never *during* its transmission. The segment lengths are possibly representing several Gigabytes of information. Hence, it makes little sense to force the start-of-segment restriction, especially if the following

factors are accounted for:

(a) As a matter of design, VoD systems are to predominantly operate over uni–directional broadcast/multicast channels, over cable or satellite infrastructure. Forward Error Correction (FEC) is routinely employed in such systems for increased open–loop data integrity. FEC schemes group information in units (blocks) such that error propagation is limited and re-synchronization of sender/receiver is still possible in the presence of errors.

(b) The information conveyed in MPEG-1 and other VBR video encodings observes its own frame–oriented or block–oriented structure. Parsing an incoming stream of MPEG-1 data allows the receiver to synchronize by identifying the boundaries between successive frames.

(c) A possible multiplexing technology used for transporting the several different video segments is packet–(cell–)based statistical multiplexing. Thus the entire segments are split into packets and the inclusion of timing information in the header of the packets is subsequently trivial to accomplish.

The above three factors indicate that it is, in principle, possible to start reception of a segment without waiting for its beginning, since the fragmentation of the entire segment into smaller units is a matter of error resilience, or of the nature of video traffic, or of the underlying multiplexing technique. For the sake of brevity we will assume that the reception starts at frame boundaries. For example, a segment consisting of 10000 frames, can be completely received if 10000 consecutive frames are received from the channel on which the corresponding segment is cyclically broadcast. The time interval between the tune–in and the identification of the first frame of a segment is considered negligible, at least compared to w. Note that reasonable values of w for realistic VoD systems are in the order of a few minutes.

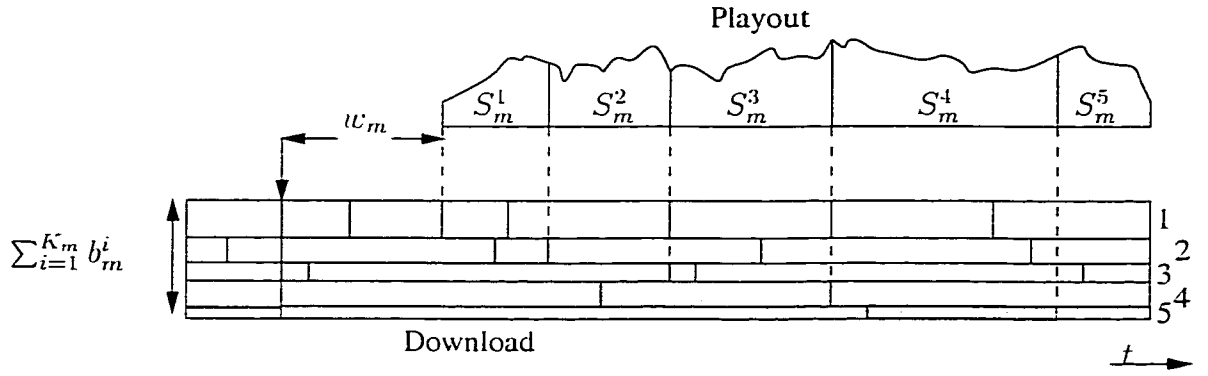


Figure 4.1: An example of the operation of LLBE.

4.2 Definition of the LLBE Protocol

Figure 4.1 illustrates the architecture of the LLBE protocol. The x–axis is time, the y–axis is bandwidth. The playout part describes the consumption of the video at the client. The download part describes the continuous concurrent periodic broadcast of the five example segments. Each of the five separate segments is periodically transmitted on a channel of bandwidth b_m^i. The shaded area in the download part corresponds to the information that is actually download by the client. Each segment is completely stored at the client just in time for its playout.

Let M denote the number of VBR videos to be broadcast. The bandwidth of the broadcast link from server to clients is B Mbits/sec. All video streams sent by the server share the B Mbits/sec. The consumption rate of each video is F frames per second. The trace sequence of each video is fully known *a priori*. Let f_m^i, $i = 1, ..., N_m$, $m = 1, ..., M$ stand for the number of bits in the i–th frame of the m–th video, where N_m denotes the total number of frames of the m–th video.

The m–th video is divided into K_m segments prior to broadcasting. The server broadcasts $\sum_{m=1}^{M} K_m$ video streams simultaneously, each stream periodically broadcasts the same segment of the same video. A central point of a periodic broadcast scheme is the way in which the videos are fragmented. The m–th video is fragmented into K_m segments of different sizes. Let S_m^i denote the set of successive frames of the i–th segment of the m–th video, we have:

$$N_m = \sum_{i=1}^{K_m} |S_m^i| \tag{4.1}$$

In LLBE, each of the K_m segments is transmitted on a separate channel of bandwidth b_m^i, $i = 1, \ldots, K_m$. Thus, the total bandwidth necessary to transmit the m–th video given a fragmentation $S_m = \{S_m^i, \ldots, S_m^{K_m}\}$ is

$$B_m(S_m) = \sum_{i=1}^{K_m} b_m^i(S_m) \tag{4.2}$$

In LLBE the client behaves in a greedy fashion, that is, it starts downloading all K_m segments that correspond to the desired video. Moreover, the download of all the segments starts at the same time point: the tune–in instant. Thus, equation (4.2) is both the required server and required client bandwidth. As segments get gradually consumed, the necessary client bandwidth decreases. That is, the maximum client bandwidth is necessary in the beginning of the video reception. Essentially, the required per–video server and client bandwidth are coupled. In the next section, a minimization process is employed to reduce the required server and client bandwidth on a per–video fashion.

Following the construction illustrated on Figure 4.1, and in order to guarantee the startup latency w_m, we note that the first segment must be broadcast at a bandwidth:

$$b_m^1(S_m) = \frac{\sum_{i \in S_m^1} f_m^i}{w_m} \tag{4.3}$$

Equation (4.3) establishes that the surface of the first horizontal rectangle of the step–function for the client download in Figure 4.1 is the same as the surface of the area denoted by the S_m^i in the playout graph of the same figure. A similar relation is true for all subsequent segments. That is, in order for the continuous and timely playout, the next segment, S_m^i, is fully downloaded and available at the client just before the end of the playout of segment S_m^{i-1}. Thus, the bandwidth at which the i–th stream of the m–th video must broadcast segment S_m^i is determined by:

$$b_m^i(S_m) = \frac{\sum_{i \in S_m^i} f_m^i}{w_m + \frac{\sum_{j=1}^{i-1} |S_m^j|}{F}} \tag{4.4}$$

Overall, after the startup latency w_m seconds, the uninterrupted playout of the video is possible. Each segment is available on local storage at the client just–in–time

for its playout. The data are played out at their nominal frame rate of F frames per second in the order of $S_m^1 \bullet S_m^2 \bullet S_m^3 \ldots S_m^{K_m}$. Note that because the frames have different sizes, the playout curve of Figure 4.1 which represents the per–frame consumed data is also variable. From equations (4.3) and (4.4), we conclude that the server bandwidth requirement to broadcast the m–th video using LLBE for any valid transmission schedule, S_m, can be expressed by:

$$B_m(S_m) = \sum_{i=1}^{K_m} \frac{\sum_{j \in S_m^i} f_m^j}{w_m + \frac{\sum_{j=1}^{i-1} |S_m^j|}{F}} \tag{4.5}$$

The loss–less nature of LLBE is evident since the aggregation of the bandwidth required for all the K_m segments of the m–th video, is constant as described by equation (4.5).

One can observe that several feasible schedules exist, since the construction of S_m is based on fairly mild constraints. Namely, the only constraint necessary for S_m to be a feasible schedule is that all segments together cover the entire video and that they do not overlap. The bandwidth–efficiency component of LLBE comes from the selection of the feasible schedule that minimizes the required server (client) bandwidth (equation (4.5)). Our objective is thus to determine S_m^i's such that a broadcast schedule can be constructed which results in the least amount of bandwidth requirement B_m^*. $B_m^* = \inf\{B_m(S_m) | S_m \text{ is a feasible schedule}\}$.Essentially, we wish to determine an ordered sequence of $K_m - 1$ integers (the "boundaries" between the segments) in the range 1 to $N_m - 1$ that minimizes the value of equation (4.5).

4.3 Calculation of the Optimal S_m and B_m^*

In formulating the optimal selection for S_m to achieve the minimum required server bandwidth, we consider a discrete–time model at the frame level. That is, the segments are to consist of an integer number of frames. Let us consider the collection from the i–th to the $(j-1)$–th frame of a video. Let us define $C_m^{i,j}$ as the bandwidth necessary if the group of frames from i to $j-1$, inclusive, were to form a segment:

$$C_m^{i,j} = \frac{\sum_{t=i}^{j-1} f_m^t}{w + \frac{(i-1)}{F}} \tag{4.6}$$

Using the definition of cost from equation (4.6) and given a sequence of video frame sizes f_m^i, we can construct a Directed Acyclic Graph (DAG) with vertices labelled from 1 to $N_m + 1$ and with directed edges from i to j where $N_m + 1 \geq j > i \geq 1$. The edge weights are as defined by the cost equation (4.6). A fragmentation to K_m segments (or less) amounts to finding a path from 1 to $N_m + 1$ consisting of K_m edges (or less). Since the minimization we wish to apply corresponds to the minimization of the cost of the edges participating in the path, the optimization problem is in fact a shortest-path problem. Figure 4.2 illustrates the relation of the shortest path and the optimal fragmentation of a video for $K_m = 3$.

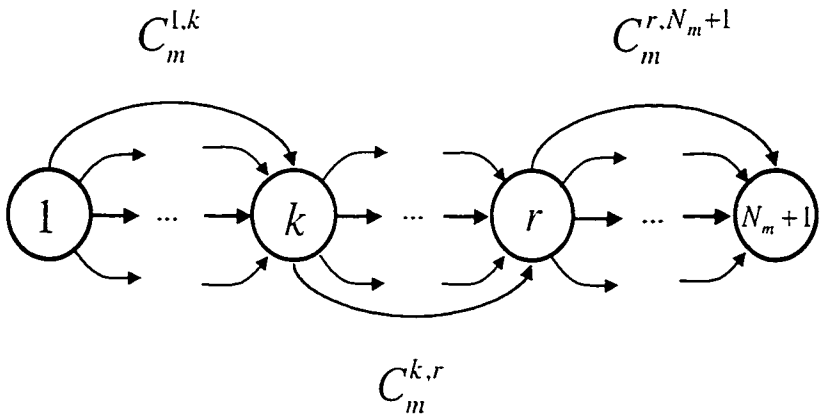


Figure 4.2: Example shortest path and optimal fragmentation, S_m, for $K_m = 3$ given a certain w_m. Segment S_m^1 spans frames from 1 to $k-1$. Segment S_m^2 spans from k to $r-1$. Segment S_m^3 spans from r to N_m.

It appears appealing to apply the Bellman-Ford algorithm, specialized to single-source (from vertex 1) shortest path. By terminating the algorithm after its $(K_m - 1)$-th iteration, we guarantee that all paths found up to this point are the shortest paths with K_m *or less* edges. Thus, the resulting running time is $O(K_m(N_m)^2)$. However, a source of complications is not the running time but the need to preserve the original cost matrix $C_m = \{C_m^{i,j}\}$ in main memory. The space requirement for the C_m matrix is $O((N_m)^2)$ but because typical values of N_m are 160,000 or more (for a two-hour video file) it is not practical to assume that the cost matrix can be stored entirely in main memory in currently available systems. Resorting to virtual memory is one

option but results in several orders of magnitude slowdown in terms of the execution speed.

To solve the problem of the large memory requirements, we introduce a vector $A_m = \{A_m^i\}$ which contains the prefix sums of the frame size sequence, that is:

$$A_m^i = \sum_{j=1}^{i-1} f_m^j, \qquad i = 1, \ldots, N_m + 1 \tag{4.7}$$

From equations (4.6) and (4.7), $C_m^{i,j}$ can be restated as:

$$C_m^{i,j} = \frac{A_m^j - A_m^i}{w_m + \frac{i-1}{F}} \tag{4.8}$$

$sum = 0$;
for $i = 1, \ldots, N_m + 1$ **do**
  $A_m^i = sum$;
  $D_m^i = A_m^i$;
  $W_m^i = D_m^i$;
  $sum = sum + f_m^i$; // Note: $f_m^{N_m+1} = 0$
endfor
for $iter = 1, \ldots, (K_m - 1)$ **do**
  for $j = 2, \ldots, N_m + 1$ **do**
    for $i = 2, \ldots, (j - 1)$ **do**
      if $\left(W_m^j > D_m^i + \frac{A_m^j - A_m^i}{w_m + \frac{i-1}{F}} \right)$ **then**
        $W_m^j = D_m^i + \frac{A_m^j - A_m^i}{w_m + \frac{i-1}{F}}$;
        $r_m^{iter,j} = i$;
      endif
    endfor
  endfor
  $D_m = W_m$; // array-wide operation
endfor

Figure 4.3: Pseudocode of single-source Bellman-Ford version of the K_m-step shortest path algorithm using an $O(N_m)$ prefix sum array, A_m. W_m is a work array. $r_m^{iter,j}$ holds (if applicable) the intermediate vertex on the path to j at the *iter* iteration (if applicable). $r_m^{iter,j}$ is $(K_m - 1) \times N_m$ and is initialized to NULL.

By introducing equation (4.8) the memory requirements are decreased to $O(K_m N_m)$. The calculation of $C_m^{i,j}$ for any i, j pair is performed upon demand based on the A_m

vector which is used to look-up the prefix sums. A_m is calculated in $O(N_m)$ once at the beginning of the run. Figure 4.3 presents the pseudocode of the specialized shortest-path algorithm, and it can be clearly seen that it exhibits a time complexity of $O(K_m(N_m)^2)$ and space complexity of $O(K_m N_m)$.

We note that for LLBE, the scheduling process for each video is independent from all the other videos that share the same link. In other words, once the transmission schedules for a specific video is constructed, the required bandwidth for transmitting this video is finalized and it is only dedicated for this video.

4.4 Experimental Results

The presented experiments are based on the application of LLBE on a set of sample traffic traces coded according to the MPEG-1 standard[35]. All traces used herein are 40000 frames long, captured at 25 frames per second with a Group of Picture (GOP) of 12 frames. We selected a total of $M = 10$ videos from the set of available traces. The set contains both low-motion and high-motion content, namely it includes news broadcasts, feature movies, music videos, sporting events and animations. We do not present separate per-video results for each video since the performance results we derived (unless stated otherwise) lead to the same basic conclusions regardless of the video content.

First we look into the influence of K_m and w_m on the required server bandwidth for an arbitrary video. We anticipate that increasing K_m or w_m decreases the required bandwidth. Morever, a question that arises naturally by observing the slowly increasing bandwidth demand in HB and its variants is whether there exists an asymptotic bandwidth demand which, given a startup latency objective, is the limit of the required bandwidth as the number of segments K_m tends to infinity. The results presented in Figure 4.4 suggest that this asymptotic behavior is indeed the case. Hence, if we factor out the constraint that the number of segments is limited by technological constraints, we claim that there exists an inherent bandwidth limit for a given startup latency w_m. This is demonstrated for VBR video traffic in LLBE, and trivially, the same is true for CBR video as well. Consequently, we can claim that the experimental results suggest that for LLBE:

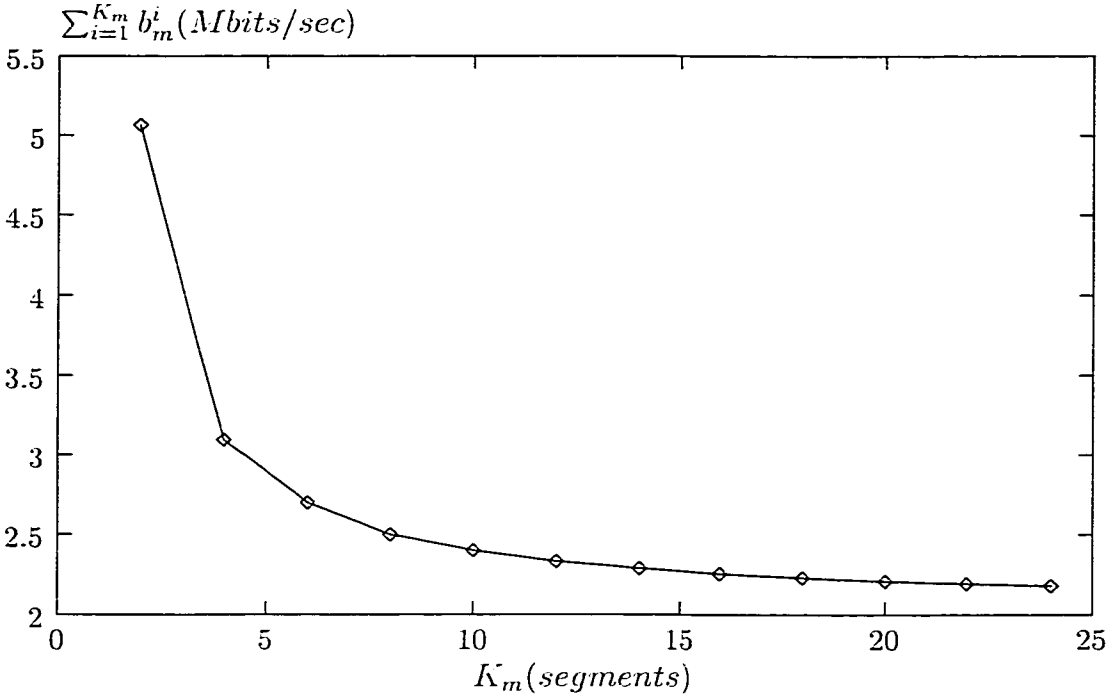


Figure 4.4: Optimal server bandwidth, B_m^*, demand for a sample video vs. variable number of segments K_m. ($w_m = 60$ *sec*, trace `MTV_I`)

$$\lim_{K_m \to \infty} B_m^* = constant < \infty \tag{4.9}$$

Figure 4.4 presents the results for a specific video trace for variable K_m from 2 to 24 and for a fixed startup latency $w = 60sec$. Similar curves were observed for all examined video traces. We note that the value at which B_m^* converges depends on the *exact* video trace under consideration, that is, different videos converge to different values. The convergence value does not depend only on the average frame size or any other trace–wide statistic, but instead on the *exact* sequence of frame sizes. We also caution the reader that the asymptotic behavior is a byproduct of the fact that the broadcast schedule construction uses a *maximum* of K_m channels. If we use a strict requirement of *exactly* K_m channels, then for increasing K_m, a larger bandwidth demand is possible. To illustrate the point, consider the extreme case $K_m = N_m$, where one channel is assigned to each separate frame. The resulting bandwidth demand is $\sum_{i=1}^{N_m} (f_m^i/(w_m + i - 1))$ (in bits per frame period) which can be much larger than the one for smaller K_m. For example, for the experiment presented in Figure 4.4, if $K_m = N_m$ (exactly) the required bandwidth is approximately 7 *Mbits/sec*.

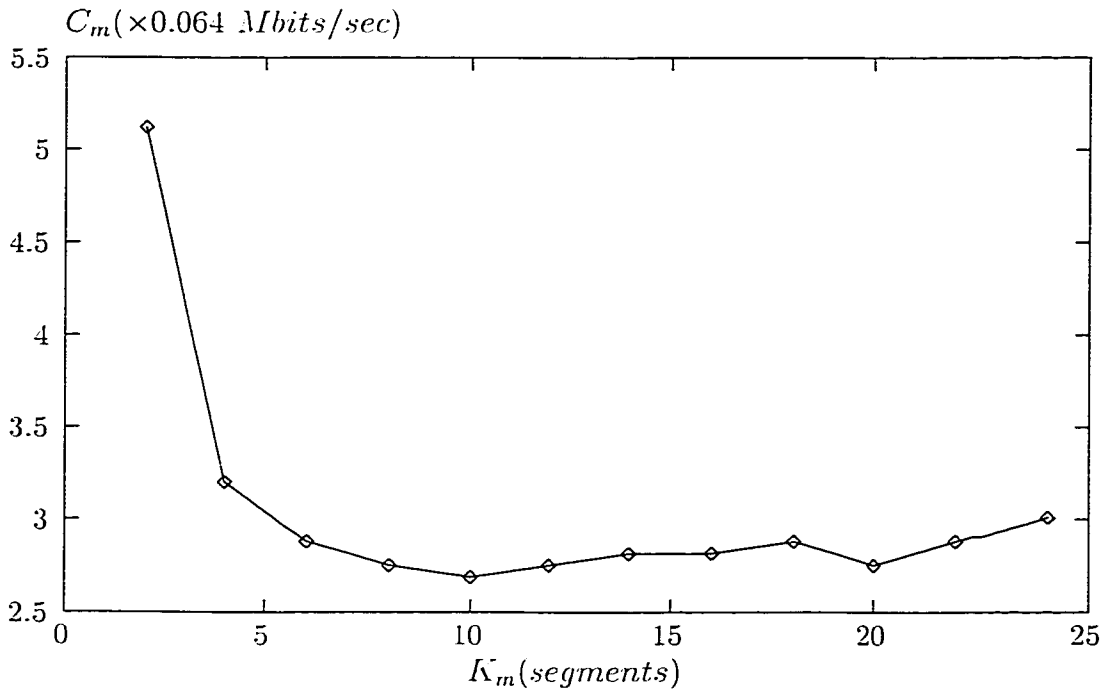


Figure 4.5: Optimal quantized server bandwidth, $C_m \times 0.064$ ($Mbits/sec$), demand for a sample video vs. variable number of segments K_m. ($w_m = 60$ sec, trace MTV_I)

LLBE, by its definition, produces segments that are broadcast on channels of different bandwidths. An implementation issue is how to schedule a given broadcast medium between several channels/streams such that each one is allocated a different portion of the available bandwidth. The actual complexity is in providing the mechanism for allowing the allocation of *arbitrary* bandwidth to each channel/stream. The current literature provides already a large assortment of scheduling mechanisms, e.g., Weighted Fair Queueing (WFQ) and its approximations, to allow implementations of arbitrary bandwidth allocations. Instead of the scheduling issue, we focus on the related issue of quantized bandwidth allocation, that is, on the inefficiency of using a scheduling scheme that allows only multiples of a basic rate to be allocated. Scheduling schemes that provide quantized bandwidth allocation are, for example, the Time Division Multiplexing (TDM) schemes. Their inefficiency in terms of quantized bandwidth allocation is compensated by the straightforward multiplexing and demultiplexing mechanism, hence they are serious candidates for low–cost network interface cards for set–top boxes.

We consider a TDM system with a basic rate of 64 $kbits/sec$ (ISDN rate). Thus, we consider this particular example as an illustration of what a realistic implementation using conventional voice channels would require in terms of bandwidth. Namely, the total number of ISDN channels necessary for video m are:

$$C_m = \sum_{i=1}^{K_m} \lceil b_m^i / (64\ kbits/sec) \rceil \tag{4.10}$$

The observation we derive from experimental results for increasing K_m in Figure 4.5 is the disappearance of the monotonic decrease of B_m^* for increasing K_m. The reason is the *internal fragmentation* within the quanta of allocated bandwidth. For example, if $b_m^i = 130\ kbits/sec$, then the corresponding channels for this video segment requires $3 \times 64 kbits/sec$, leaving $3 \times 64 - 130 = 62 kbits/sec$ unused. In order to provide straightforward demultiplexing, no sharing of a primary rate channel is allowed between two or more segments. Thus, as the number of segments increases, the per–channel bandwidth may decrease, leading to an increase of the level of internal fragmentation.

A comparison of Figures 4.4 and 4.5 indicates that because of the internal bandwidth fragmentation the overall bandwidth allocation is indeed larger in the quantized case. If, instead, WFQ was used to provide an arbitrary bandwidth allocation, then a jitter absorption scheme is necessary because the implementations of WFQ can only approximate the fluid model assumed for the traffic, and service is dispensed in packet or cell quanta. Jitter absorption is also necessary for TDM systems. In TDM, the necessary jitter absorption is defined by the length of the repeating TDM cycle (each channel is allocated one or more slots within the TDM cycle, where each slot in the current example corresponds to a $64 kbits/sec$ quantum).

Dealing with the internal fragmentation inefficiency in a TDM system requires the generation of segments that use bandwidth very close (but lower) to an integer multiple of the basic rate. A strategy that can be used for reducing the internal fragmentation is the following: at the DAG construction stage that we illustrated earlier, we omit all edges that correspond to bandwidths that lead to large internal fragmentation. Hence, the shortest path, once created, uses only edges(and corresponding channel bandwidths) that to not exacerbate the internal fragmentation. However,

such technique is not valid in general because it can lead to a disconnected graph, prohibiting the construction of a complete path. We thus recommend, instead, the exploration of the set of allowed values of K_m for each video (which can be accomplished in one single run of the shortest path algorithm for up to the maximum K_m) and the selection of the value that minimizes the quantized required bandwidth.

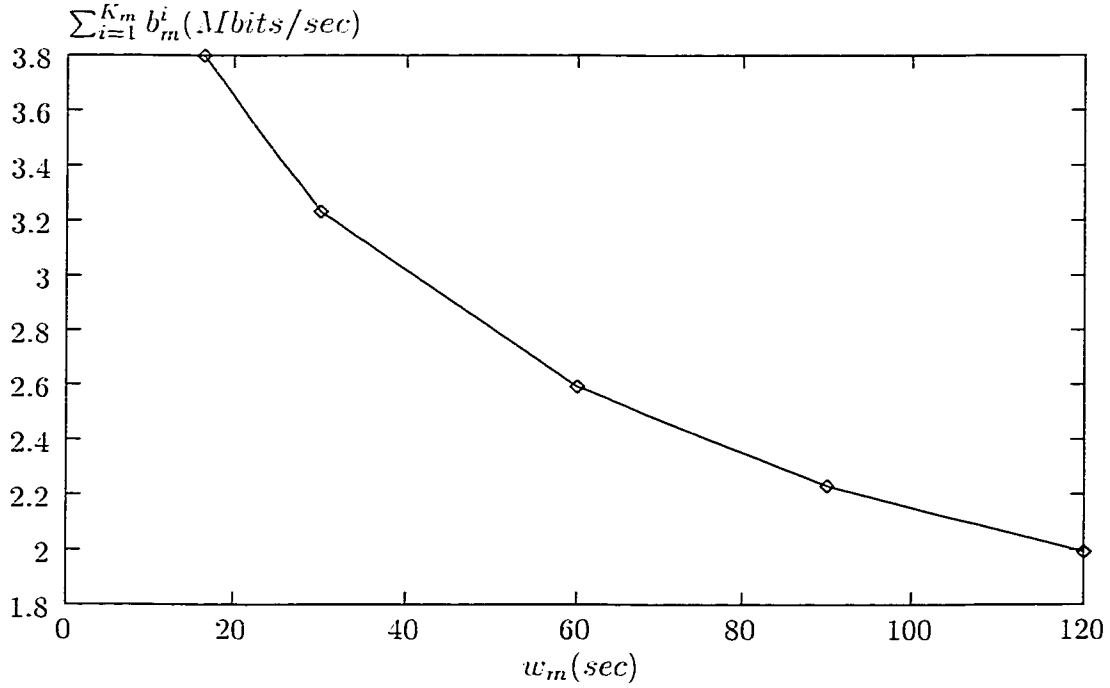


Figure 4.6: Optimal server bandwidth, B_m^*, demand for a sample video vs. startup latency w_m. ($K_m = 7$ *segments*, trace `MTV_I`)

Next we explore the performance of the required server bandwidth B_m^* as a function of w_m. Figure 4.6 presents the results that are representative of any of the examined video traces. Namely, the required bandwidth indeed decreases with increasing startup latency. The construction of LLBE allows the following very intuitive asymptotic behavior (for a given K_m):

$$\lim_{w_m \to \infty} B_m^* = 0 \tag{4.11}$$

and

$$\lim_{w_m \to 0} B_m^* = \infty \tag{4.12}$$

That is, if the clients can wait for a long period of time, the necessary bandwidth

is virtually zero (equation (4.11)). Inversely, instant playout startup requires essentially infinite bandwidth. What is important to observe, is the fast reduction of the necessary bandwidth for "reasonable" values of w_m. For example, for the MPEG-1 videos that we examined and $w_m = 60$ *sec* the necessary bandwidth is in the range of 2.5 to 3 $Mbits/sec$ and for a latency of a few minutes, the necessary bandwidth is the average bandwidth necessary of the corresponding MPEG-1 trace. Compared to the typical commodity disk drive I/O throughput of roughly 10 $Mbytes/sec$ the necessary bandwidth is indeed well below the technological limits of low–cost set–top boxes for "reasonable" values of w_m even if we assume $N_m = 160000$ or more.

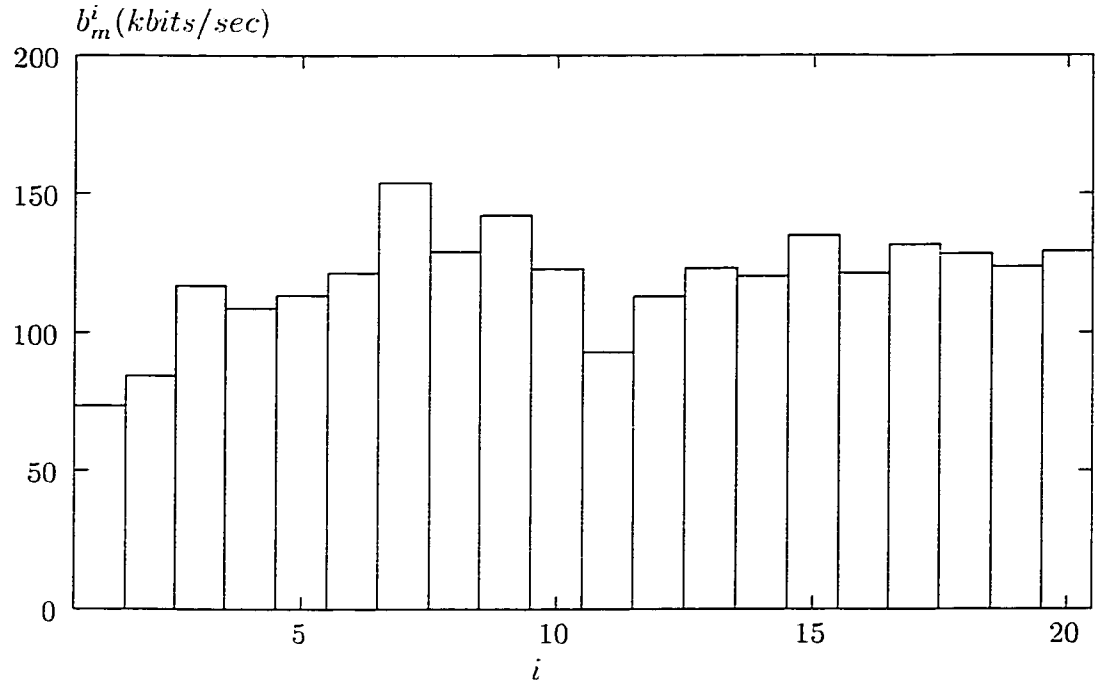


Figure 4.7: Per–segment server bandwidth, b_m^i, for each segment. (K_m = 20 *segments*, $w_m = 60$ *sec*, trace FUSS)

The next set of observations we make are related to the bandwidth necessary for each segment. Figure 4.7 illustrates the relation of the bandwidth of all the segments of the same video.For sufficiently large K_m, and for the segments in the later part of the trace the following is consistently observed:

$$b_m^i \approx b_m^{i+1} \tag{4.13}$$

In Figure 4.7 the above behavior is noticeable for $i > 12$. This observation is also

in agreement with the intuition behind the construction of the schedules with equal bandwidth segments (PB and variants).

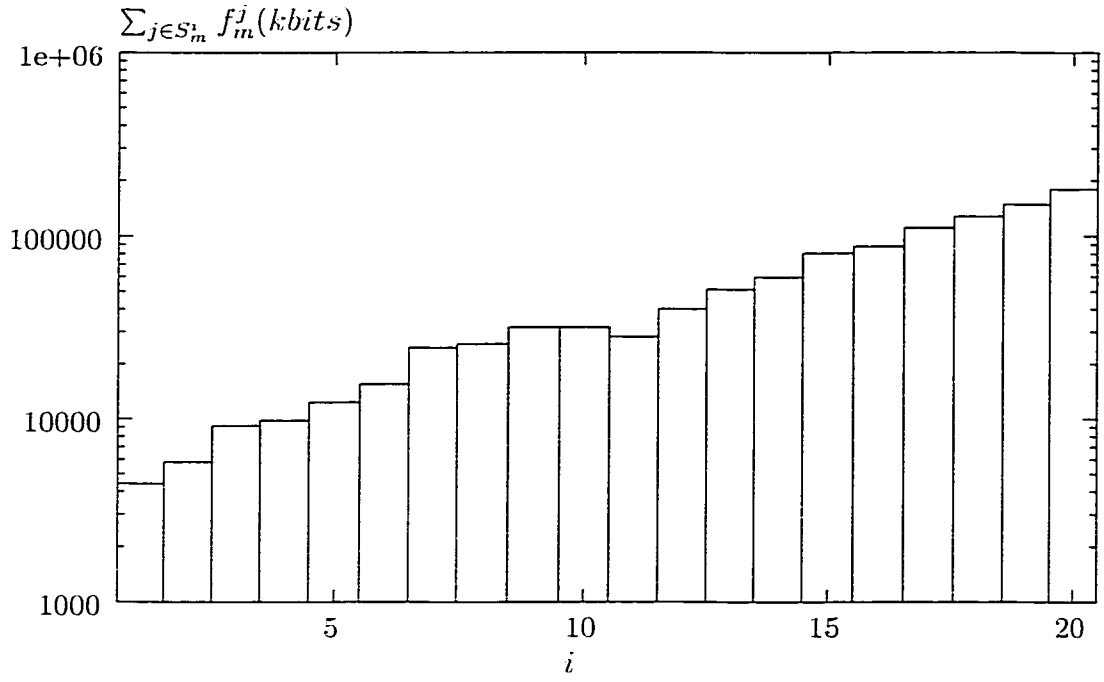


Figure 4.8: Segment sizes. $\sum_{j \in S_m^i} f_m^j$, for each segment. ($K_m = 20$ *segments*, $w_m =$ 60 *sec*, trace FUSS)

For the same video trace (FUSS), Figure 4.8 illustrates the relation of the segment sizes. Notice the logarithmic scale used for the y–axis on Figure 4.8. Hence, the almost linear increase in the size of the later segments corresponds to an exponential increase of successive segment sizes (for $i > 12$). Similar behavior was observed in all the other video traces as well. The observations can be summarized in the following relation:

$$\sum_{j \in S_m^{i+1}} f_m^j \approx \beta \sum_{j \in S_m^i} f_m^j \tag{4.14}$$

The above relation is also the intuition behind the CBR–based schemes, and the variants of PB in particular. We note however that the factor of the geometric series, β, depends on w_m, K_m and the exact frame sequence of the trace.

The later segment sizes, by accumulating a larger number of frames together, are almost equal to the product of the number of frames multiplied by the average frame

size. Essentially, for large i, the corresponding segments depend on the average frame size statistics. On the other hand, for small values of i, the segments consist of a small collection of frames and they exhibit higher variability which depends on the exact collection of frames. Hence, for large i, an exponential increase of the length of the segments (in terms of numbers of frames) is also observed (for the sake of brevity, we omit presenting a corresponding figure).

A final performance comparison can be performed against the two schemes that have been proposed in the past for VBR periodic broadcast, VBR–B and TAF [36]. We first note that the segments sizes in VBR–B follow an geometric sequence with a factor of 2. TAF generalizes the segment length selection. The results shown in Figures 4.7 and 4.8 indicate that the selection of the "best" segment lengths (sizes) may follow a sequence which is not necessarily increasing, in direct contrast to VBR–B and TAF. The ability of LLBE to guarantee the timing constraints given a non–increasing segment lengths is due to its particular "greedy" nature of the client download strategy. Further, the size of segments, in terms of bits, compared to their size in terms of frames are drastically different. For example, the number of frames corresponding to the 8–th segment of Figure 4.8 is approximately twice the number of frames of the 9–th segment, but the size of the 8–th segment in bits is less than that of the 9–th segment.

To compare the bandwidth efficiency of multiplexing 10 videos in the proposed LLBE with that of VBR-B [36] and TAF, we set the number of segments for each videos, K_m to 7. The startup latency w_m is set to 16.5 seconds. Figure 4.9 demonstrates the data loss rate for three VBR broadcasting schemes for a collection of 10 videos. The bandwidth from the aggregation of the 10 videos according to LLBE is used as bandwidth for VBR–B and TAF. The results clearly indicate that under VBR–B and TAF, the resulting data loss (approximately 15 and 10 % respectively) is high enough to render the video distribution useless in such limited bandwidth. The exception is LLBE. In order to achieve the same zero data loss rate, the required bandwidth for transmitting the 10 movies for LLBE is much lower than that for VBR-B and TAF (Figure 4.10). For LLBE, the required bandwidth is 33.59 Mbits/sec, while for VBR-B it is 86.96 Mbits/sec and 60.72 Mbits/sec for TAF.

Scheme	Loss Rate
LLBE	0.000
VBR-B	0.153
TAF	0.104

Figure 4.9: Data loss rate for three VBR broadcasting schemes for a collection of 10 videos. ($K_m = 7$ *segments*, $w_m = 16.5$ *sec*, $B = 33.6$ *Mbits/sec*)

Scheme	$\sum_{i=1}^{K_m} b_m^i$
LLBE	33.587
VBR-B	86.958
TAF	60.722

Figure 4.10: Required server bandwidth for zero data loss rate for three VBR broadcasting schemes for a collection of 10 videos. ($K_m = 7$ *segments*, $w_m = 16.5$ *sec*)

Chapter 5

Conclusions and Future Work

5.1 Concluded Remarks

We have investigated the techniques, in particular, the transmission scheduling schemes that address the scalability problems of VoD systems. Specifically, we have proposed (a) a technique that allows the construction of time–dependent envelopes of VBR video traffic, based on which a scheduled multicast scheme is employed in the call admission process of the user requests, (b) a Trace–Adaptive Fragmentation (TAF) scheme for periodic broadcast of VBR video, which derives a set of fragmentation choices and selects the one with the least packet loss rate for the aggregated segments based on the particular video traffic traces, and (c) a Loss–Less Bandwidth–Efficient (LLBE) protocol for periodic broadcast of VBR video that allows the user to receive video data immediately after tuning in and handles the distribution of VBR video as a collection of per–segment CBR transmissions in order to avoid data losses and to achieve the minimal required server bandwidth. In particular, we note that all the proposed techniques in this thesis are associated with the solution of a corresponding optimization problem.

For the scheduled multicast scheme presented in Chaper 2, we have essentially traded the buffers (inside the network and at the receivers) for more computation during the call admission process. We are exploring the impact of different envelope granularities for different types of content and its interaction with smoothing. Notably, the scheme presented herein does not preclude smoothing for several successive frames at the sender before transmission.

In Chapter 3, for the construction of broadcast schedules, given the a–priori knowledge of the entire traffic and given the system objectives we have approached the problem as essentially a *deterministic* problem, for which choice of an "optimal" broadcast schedule with the least data losses may be possible. In order to cope with the many and conflicting performance objectives we have focused on the construction of broadcast schedules that satisfy startup latency and continuity constraints and enforce a maximum number of server and client channels. An enumerative process identifies broadcast schedules that satisfy these constraints. Subsequently, the selection of a particular schedule, which is determined by the choosing of a particular fragmentation, is the result of an optimization process which accounts for the VBR nature of

the transmitted video segments. The optimization process , TAF, appears currently to be computationally expensive, and, as a result, we propose a fast approximation which has been shown to outperform existing rigid fragmentation approaches in a variety of settings.

In Chapter 4, for the design of LLBE protocol, we have explored the lossless VBR broadcast schedules with the least required bandwidth. Essentially, we show the equivalence of the optimization problem to a shortest path problem on a directed acyclic graph. The optimization algorithm proposed for LLBE allows us to exactly quantify the necessary parameters (the number of segments and the corresponding required bandwidth to transmit each segment under the continuity condition) based on the particular stored video content and startup latency objective. Moreover, because LLBE handles the distribution of VBR video as a collection of per–segment CBR streams, it inherently avoids data losses that plagued previous schemes (even when they include additional techniques to reduce the losses at significant implementation complexity).

5.2 Future Directions

We still need to investigate the issues of how to implement the interactivity performance such as pause, rewind and fast-forward in near–VoD systems using our proposed scheduled multicast techniques.

Currently, in LLBE, the server and client bandwidth demands for a video are identical. Future research in this direction will focus on the reformulation of the optimization problem in a manner that will allow a–priori constraints to be enforced on the client–side I/O bandwidth separately from constraints enforced on the server–side bandwidth. We are also exploring techniques to support potentially heterogeneous clients/set–top boxes. Another possible use is the application of the decreasing–only download bandwidth, as dictated by LLBE, to the call admission in a batched multicast or advance reservation system. One approach to this end is the generalization of LLBE, in a similar manner to the generalization of Skyscraper Broadcasting [19] towards the Dynamic Skyscraper Broadcasting of Eager and Vernon [12].

Bibliography

[1] E. Abram-Profeta. K. Shin. Scheduling Video Programs in Near Video-on-Demand Systems. In *Proc. of ACM Multimedia*, 1997.

[2] C. Aggarwal. J. Wolf. and P. Yu. On optimal batching policies for video-on-demand storage servers. In *Proc. of IEEE Intl. Conf. on Multimedia Systems '96*. 1996.

[3] C. Aggarwal, J. Wolf, and P. Yu. A permutation-based pyramid broadcasting scheme for video-on-demand systems.In *Proc. of IEEE Intl. Conf. on Multimedia Systems '96*. 1996.

[4] K. Almeroth and M. Ammar.The use of multicast delivery to provide a scalable and interactive video-on-demand service. *IEEE JSAC*, August 1996.

[5] M. Ammar and J. W. Wong. The design of teletext broadcast cycles. *Performance Evaluation*. 5(4):235–242. 1985.

[6] S. Chakrabarti and R. Wang. Adaptive Control for Packet Video. In *Proc. IEEE ICMCS '94*. Boston, USA, 1994.

[7] S. Chan, F. Tobagi and T. Ko. Providing On-Demand Video Services Using Request Batching. Technical Report, Computer System Laboratory, Stanford University, CA, USA.

[8] K. Chang, H. Kung. Efficient Time-Domain Bandwidth Allocation for Video-on-Demand Systems. In *Proc. ICCCN '96*, Oct. 1996, Washington D.C., USA.

[9] I. Dalgic and F. Tobagi. Characterization of quality and traffic for various video encoding schemes and various encoder control schemes. Technical Report CSL–

TR–96–701, Department of Electrical Engineering and Computer Science, Stanford University, August 1996.

[10] A. Dan, D. Sitaram, and P. Shahabuddin. Scheduling policies for an on-demand video server with batching. In *Proc. of ACM Multimedia '94*, pages 15–23, October 1994.

[11] A. Dan, D. Sitaram, and P. Shahabuddin. Dynamic Batching Policies for An On-Demand Video Server. In *Mulitmedia Systems*, pp. 112-121, June 1996.

[12] D. Eager, M. Vernon. Dynamic Skyscraper Broadcasts for Video-on-Demand. In *Proc. of MIS '98*, Sept. 1998.

[13] W. Feng and S. Sechrest. Smoothing and Buffering for Delivery of Prerecorded Compressed Video. In *ISET/SPIE Symp. on Multimedia Computing and Networking*, Feb. 1995.

[14] M. Garrett and W. Willinger. Analysis, Modeling and Generation of Self-Similar VBR Video Traffic. In *Proc. ACM SIGCOMM*, Sept. 1994.

[15] M. Grossglauser and D. Tse. A Time-Scale Decomposition Approach to Measurement-Based Admission Control. In *Proc. IEEE INFOCOM'99*, March 1999.

[16] M. Grossglauser, S. Keshav and D. Tse. RCBR: A Simple and Efficient Service for Multiple Time-Scale Traffic. In *IEEE/ACM Trans. Networking*, Dec. 1997.

[17] P. Hart, N. Nilsson, and B. Raphael, A formal basis for the heuristic determination of minimum cost paths. In *IEEE Trans. Syst. Sci. Cybern.*, SSC-4, 2 (July 1968), pp. 100-107.

[18] K. Hua, Y. Cai, and S. Sheu. Exploiting client bandwidth for more efficient video broadcast. In *Proc. of IEEE ICCCN '98*, 1998.

[19] K. Hua and S. Sheu. Skyscraper broadcasting: A new broadcasting scheme for metropolitan video-on-demand systems. In *Proc. of ACM SIGCOMM '97*, pages 89–100, 1997.

[20] C. Judice, E. Addeo, M. Eigner, H. Lemberg. Video on Demand: A Wideband Service or Myth? In *Proc. ICC '86*, pp. 1735–1739, June 1986.

[21] L. Juhn and L. Tseng. Harmonic broadcasting for video-on-demand service. *IEEE Trans. Broadcasting*, 43(3):268–271, September 1997.

[22] L. Juhn and L. Tseng. Adaptive Fast Data Broadcasting Scheme for Video-on-Demand Service. In *IEEE Trans. Broadcasting*, June 1998.

[23] E. Knightly, H. Zhang. D-BIND: An Accurate Traffic Model for Providing QoS Guarantees to VBR Traffic In *ACM/IEEE Trans. Networking*, May 1996.

[24] E. Lawler. *Combinatorial Optimization*. Holt,Rinehart and Winston, 1976.

[25] V. Li, W. Lian, X. Qiu and E. Wong. Performance Model of Interactive Video-on-Demand Systems. In *IEEE Journal On Selected Areas In Communications*, August 1996.

[26] J. Liebeherr and D. Wrege. An Efficient Solution to Traffic Characterization of VBR Video in Quality-of-Service Networks. In *Proc. IEEE INFOCOM'96*, Mar. 1996.

[27] S. Liew and D. Tse. A Control-Theoretic Approach to Adapting VBR Compressed Video for Transport Over a CBR Communications Channel. In *IEEE/ACM Trans. Networking*, Feb. 1998.

[28] S. Liew and C. Tse. Video Aggregation: Adapting Video Traffic for Transport Over Broadband Networks by Integrating Data Compression and Statistical Multimplexing. In *IEEE Journal on Selected Areas in Communications*, August, 1996.

[29] J. McManus, K. Ross. A Dynamic Programming Methodology for Managing Prerecorded VBR Sources in Packet-Switched Networks. *http://www.systems.seas.upenn.edu/ross/jean2.ps*, Jan. 1997.

[30] J. Paris, S. Carter, and D. Long. Efficient broadcasting protocols for video on demand. In *Proc. of MASCOTS '98*, pages 127–132, July 1998.

[31] J. Paris, S. Carter, and D. Long. A low bandwidth broadcasting protocol for video on demand. In *Proc. IEEE ICCCN '98*, 1998.

[32] A. Reibman and A. Berger. Traffic Descriptors for VBR Video Teleconferencing over ATM Networks. In *IEEE/ACM Trans. Networking*, June 1995.

[33] D. Reininger, D. Raychaudhuri and J. Hui. Bandwidth Renegotiation for VBR Video Over ATM Networks. In *IEEE Journal on Selected Areas In Communications*, August 1996.

[34] M. Reisslein and K. Ross. A join-the-shortest-queue prefetching protocol for vbr video on demand. In *Proc. IEEE ICNP '97*, October 1997.

[35] O. Rose. Statistical properties of mpeg video traffic and their impact on traffic modelling in atm systems. Technical Report Technical Report 101, University of Wuerzburg, Germany, February 1995.

[36] D. Saparilla, K. Ross, and M. Reisslein. Periodic broadcasing with vbr-encoded video. In *Proc. IEEE INFOCOM '99*, 1999.

[37] D. Salehi, Z. Zhang, J. Kurose, D. Towsley. Supporting Stored Video: Reducing Rate Variability and End-to-End Resource Requirements through Optimal Smoothing In *Proc. SIGMETRICS 1996*, pp. 222–231, May 1996.

[38] B. Vandalore, G. Babic and R. Jain. Analysis and Modeling of Traffic in Modern Data Communication Networks. Available through http://www.cis.ohio-state.edu/ jain/papers.html.

[39] S. Viswanathan and T. Imielinski. Metropolitan area video-on-demand service using pyramid broadcasting. *Multimedia Systems*, 4(4):197–208, August 1996.

[40] D. Wrege, E. Knightly, H. Zhang, J. Liebeherr. Deterministic Delay Bounds for VBR Video in Packet-Switching Networks: Fundamental Limits and Practical Trade-Offs. In *IEEE/ACM Trans. Networking*, vol. 4, no. 3, pp. 352–362, June 1996.

[41] E. Zegura, S. McFarland, O. Parekh. A Survey and New Results in Renegotiated Service. In *Journal of HighSpeed Networks*, Volume 6, No. 3, 1997.

[42] W. Zhao, M. Krunz, S. Tripathi. Efficient Transport of Stored Video Using Stream Scheduling and Window-Based Traffic Envelopes. In *Proc. ICC '97*, June 1997, Montreal, Canada.

[43] W. Zhao and S. Tripathi. Bandwidth-Efficient Continuous Media Streaming Through Optimal Multiplexing. In *Proc. SIGMETRICS '99*, May 1999.

[44] W. Zhao, T. Seth, M. Kim, M. Willebeek-LeMair. Optimal Bandwidth/Delay Tradeoff for Feasible-Region-Based Scalable Multimedia Scheduling. In *Proc. IEEE INFOCOM '98*, March 1998.