2004

Interactivity And User-heterogeneity In On Demand Broadcast Video

Mounir Tantaoui El Araki
University of Central Florida

INTERACTIVITY AND USER-HETEROGENEITY IN ON DEMAND BROADCAST VIDEO

by

MOUNIR TANTAOUI EL ARAKI
B.S. University of Mohammed V, 1996
M.S. University of Central Florida, 1999

A dissertation submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy
from the School of Computer Science
in the College of Engineering and Computer Science
at the University of Central Florida
Orlando, Florida

Summer Term
2004

ABSTRACT

Video-On-Demand (VOD) has appeared as an important technology for many multimedia applications such as news on demand, digital libraries, home entertainment, and distance learning. In its simplest form, delivery of a video stream requires a dedicated channel for each video session. This scheme is very expensive and non-scalable. To preserve server bandwidth, many users can share a channel using multicast. Two types of multicast have been considered. In a non-periodic multicast setting, users make video requests to the server; and it serves them according to some scheduling policy. In a periodic broadcast environment, the server does not wait for service requests. It broadcasts a video cyclically, e.g., a new stream of the same video is started every t seconds. Although, this type of approach does not guarantee true VOD, the worst service latency experienced by any client is less than t seconds. A distinct advantage of this approach is that it can serve a very large community of users using minimal server bandwidth.

In VOD System it is desirable to provide the user with the video-cassette-recorder-like (VCR) capabilities such as fast-forwarding a video or jumping to a specific frame. This issue in the broadcast framework is addressed, where each video and its interactive version are broadcast repeatedly on the network. Existing techniques rely on data prefetching as the mechanism to provide this functionality. This approach provides limited usability since the prefetching rate cannot keep up with typical fast-forward speeds. In the same environment, end users might have access to different bandwidth capabilities at different times. Current periodic broadcast schemes,

do not take advantage of high-bandwidth capabilities, nor do they adapt to the low-bandwidth limitation of the receivers. A heterogeneous technique is presented that can adapt to a range of receiving bandwidth capability. Given a server bandwidth and a range of different client bandwidths, users employing the proposed technique will choose either to use their full reception bandwidth capability and therefore accessing the video at a very short time, or using part or enough reception bandwidth at the expense of a longer access latency.

ACKNOWLEDGMENTS

The author would like to thank all those who have helped in giving advice, comments and suggestions, especially former and current Data System Group members. The author would also like to thank the department of computer science at the University of Central Florida and ImageSoft Technologies for the opportunity to work as a graduate teaching and research assistant.

Finally, the author is very much grateful to his advisor, Professor Kien A. Hua and would like to express his sincere gratitude for his encouragement, support, guidance, and patience. This work would not have been possible without his fervent guidance and constant support over the past years.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER ONE: INTRODUCTION

Next-generation networks will support transmission rates that are orders of magnitude higher than current rates and will provide advanced services not currently available. In particular, the explosive increase in commercial usage of the Internet has resulted in a rapid growth in demand for video delivery techniques. They are underlying technologies for many new, exciting multimedia applications. For examples, on-demand home entertainment gives customers the freedom to view movies from remote sites at any time in the comfort of their own home; distance learning provides opportunities for students to take courses taught at remote locations according to their individual needs and time constraints; digital video library lets remote users research and view videos from a large video library; just to name a few. *Video-On-Demand* (VOD) will therefore play an important role in providing these applications.

A VOD system consists of a video server with a video archive and a number of client machines connected via a local area network. Users use client software to request for their desired video. In response to a service request, the server delivers the requested video to the user in an isochronous data stream. The unit of server capacity required to support the playback of one video stream is referred to as a *channel*. The server network I/O bandwidth limits the number of such channels. Figure 1.1 depicts a simple architecture of a *Video-On-Demand* system.

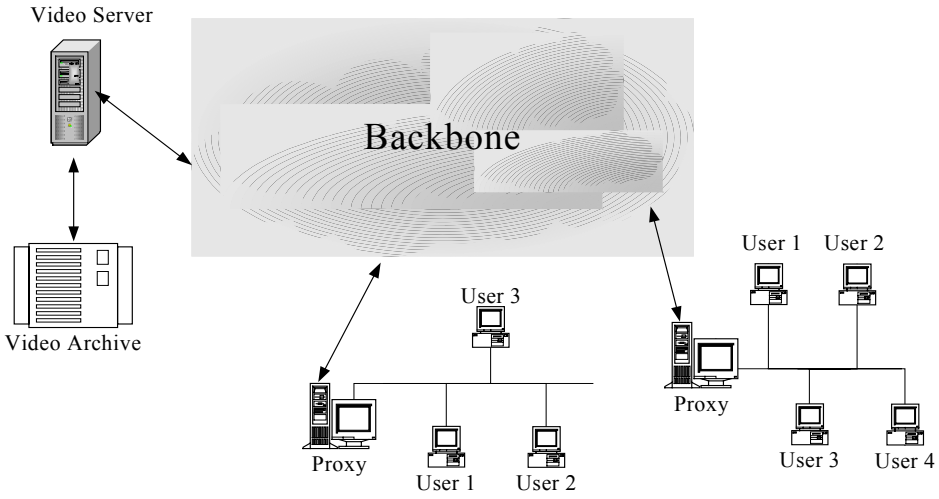


Figure 1. 1 Video-On-Demand Architecture

The simplest delivery technique requires a dedicated server channel to serve each video request. Obviously, this scheme is excessively expensive and non-scalable. To conserve server network I/O bandwidth and wide-area-network bandwidth and to reduce service delays experienced by users, three complementary approaches have been investigated.

- *Server transmission schemes using multicast:* These techniques can be categorized into *reactive transmission approaches*, *proactive transmission approaches,* and *hybrid approaches*. In the reactive transmission approach, the server uses a few server channels to serve several requests for the same video arriving closely in time. This strategy allows the users to share server and network bandwidth. In the proactive transmission approach, clients do not make requests to the server. Instead, the server broadcasts a video periodically, e.g., a new stream of the same video is started every *t* seconds. The worst

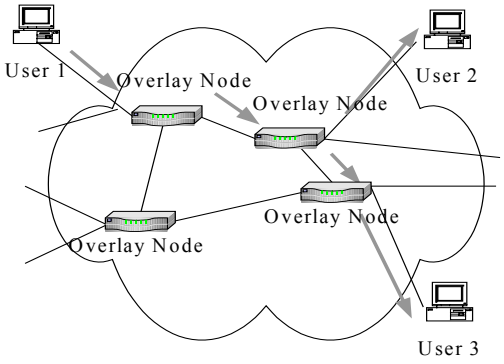
service latency experienced by any client is at most *t* seconds. A unique advantage of this approach is that it can serve a very large community of clients using minimal server bandwidth while guaranteeing a bounded service delay. In fact, the bandwidth requirement is independent of the number of concurrent clients using the system. The hybrid approach takes advantages of both the reactive and the proactive approaches.

- *Video streaming technologies with Application Layer Multicast***:** In theory**,** IP multicast can be employed with the server transmission schemes. In practice, IP Multicast has deployment difficulty beyond a local area network [DLLKB00]. As a result, several proposals for *Application Layer Multicast* (ALM) have recently been introduced along with new video streaming techniques using ALM for live broadcast and pre-recorded videos. The *infrastructure-based approach* and *the peer-to-peer* (P2P) approach are two different classes of ALM protocols. In the infrastructure-based approach, a set of dedicated machines called *overlay nodes* act as software routers with multicast functionalities [JGJKO00]. In these approaches a multicast tree is constructed where packets are replicated at overlay nodes. Video content is therefore transmitted from a source to a group of receivers on a multicast tree comprising of only the overlay nodes. A new receiver joins an existing multicast group by connecting to its nearest overlay node. *Range Multicast* [HTV00] is an infrastructure-based approach and utilizes an overlay structure consisting of overlay nodes placed at strategic locations on a wide-area network, and interconnected using unicast paths. As video packets pass through a sequence of overlay nodes on the delivery path, each node caches the video data into its

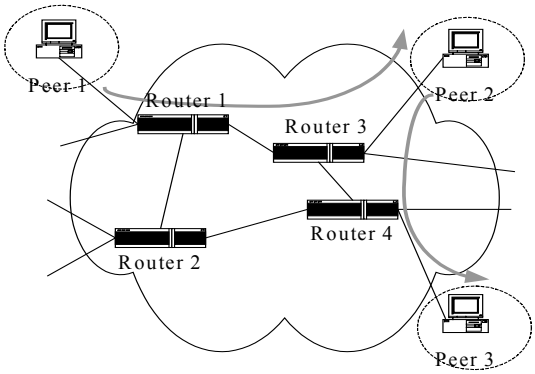
fixed-size FIFO buffer. Such buffers, if they still have the first video frame, can be used to relay the entire video stream to subsequent clients requesting the same video. Due to the high cost of deployment and maintenance of the infrastructure-based approach, the P2P approaches have been introduced and studied in Chaining [SHT97], DirectStream [GSKT03a], P2Cast [GSKT03a], and P^2VOD [DHT04] for pre-recorded video, and in CoopNet [PWC02], [XHB02] and ZigZag [THD02] for live video streaming. The communication paradigm lets users' end hosts forward video data to other users' end hosts in the downstream. Figure 1.2.a and 1.2b depict an example of the infrastructure-based approach and a peer-to-peer approach respectively.

- *Proxy caching technologies:* A video proxy (a computer system equipped with large storage space) can be used to store popular videos close to the requesting clients (refer to Figure 1.1). The proxy delivers the cached portion of the requested video to the client while the remote video server needs transmit only the uncached portion of the video. This scheme minimizes the load on the wide-area-network and the video server. A well designed proxy caching technique can also reduce service delays and improve playback quality. Proxy caching techniques can be divided into two major categories, namely *Standalone Proxy Caching* [SRT99, ZWDS00, KHRR01, RRG01, EFV99, GST03] and *Collaborative Proxy Caching* [AS00, GGUST03, PBC00]. *Standalone Proxy Caching* assumes a single V*ideo Proxy* and the allocation policy determines which portion of which video to cache at proxy storage. *Collaborative Proxy Caching* approaches take advantage of aggregate network bandwidth and storage space of several proxies in the

same Intranet to store more videos close to requesting clients. The number of participating proxies and their resources are known a priori.



a. Infrastructure-based approaches



b. P2P communication at application level

Figure 1. 2 Infrastructure-based vs. P2P

The reactive transmission schemes and proactive schemes facilitate large-scale deployment and content sharing of video services. However, clients of such applications can use multiple types of receiving devices. Therefore, the dissemination of a homogeneous description of a video does

not exploit high-bandwidth capabilities of some clients nor does it adapt to clients with low network bandwidth. A technique that can adjust to an array of heterogeneous receivers is required in practice. Furthermore, providing VCR-like capabilities (e.g., fast-forwarding or jumping to a specific frame in the video) is not only desirable but also required to quickly locate desired video content in some applications.

This dissertation introduces two new techniques, the *Broadcast-Based Interactive Technique (BIT)* for providing VCR-like functionalities, and *BroadCatch* a user-bandwidth heterogeneous approach to provide video content in a multicast environment and more specifically by employing proactive methods and approaches.

From this point, multicast techniques employing an active approach are referred to as *regular multicast techniques* or *multicast techniques*, and multicast techniques using a proactive approach are referred to as *periodic broadcast techniques*. From this terminology, a multicast setting is an environment where regular multicast techniques are used and a periodic broadcast setting where periodic broadcast techniques are employed.

Chapter 2 discusses several multicast, periodic broadcast, and hybrid approaches to disseminate video data to end-users. In this chapter, and since both proposed techniques, i.e. BIT and BroadCatch, employ periodic broadcast as way to deliver video, the main difference between multicast and periodic broadcast environments is indicated. Then several periodic broadcast techniques are described to point out various challenges.

Chapter 3 presents an overview of several schemes to provide VCR-like functionalities in

numerous related works, describes the Client-Centric Approach (CCA) [HCS98] from which the BIT technique is derived [THS02], presents and proves the correctness of the proposed Broadcast-based Interactive Technique (BIT), and shows simulation results to compare BIT to a recent method.

Chapter 4 discusses the user bandwidth heterogeneous problem related to proactive approaches. First, related works are presented. Second, the proposed BroadCatch [THD04] technique is described and analyzed. Third, several convenient characteristics of BroadCatch are explained. Finally, a performance evaluation section compares BroadCatch to existing techniques that provide user bandwidth heterogeneity and techniques that optimize user bandwidth without providing heterogeneous characteristics.

Chapter 5 is a study of two distinctive approaches [THJ03], one using a static multicast and the other a periodic broadcast delivery to disseminate dynamic terrain updates to simulators in a distributed interactive environment.

Chapter 6 concludes this dissertation with some remarks and future directions.

CHAPTER TWO: REACTIVE AND PROACTIVE APPROACHES

Video services can be classified into the following categories based on the scheduling policies of data delivery and on the degree of interactivity [LV96, MS02].

- **No-Video-on-Demand:** This service is similar to the broadcast TV service where a user is a passive participant in the system and has no control over the video session. In this case, users do not request videos.
- **Pay-Per-View:** This service is similar to the cable TV Pay-Per-View. Users sign up and pay for specific services scheduled at pre-determined times.
- **True Video-on-Demand (TVOD):** True VOD systems allow users to request and view any video at any time, with full VCR capabilities. The user has a complete control over the video session. The simplest way to achieve TVOD is to dedicate each channel to every user in the system. However, this simplest scheme is very expensive.
- **Near Video-on-Demand (NVOD):** Users requesting for the same video are served using one video stream to minimize the demand on server bandwidth. The server is, therefore, in control of when to serve the video. VCR capabilities can be provided using many channels delivering the different requested portions of the same video requested by the different users.

This chapter describes several techniques that utilize the reactive transmission approaches, the proactive transmission approaches, and a hybrid approach. Section 1 presents the difference between static multicast and dynamic multicast techniques. Section 2 describes the server-

oriented periodic broadcast techniques, client-oriented periodic broadcast schemes and compares the cost of deployment in both environments. Section 3 explains a hybrid solution of reactive and proactive transmissions. Section 4 describes some implementation issues related to periodic broadcast products and section 5 summarizes the chapter.

2.1 Multicast: Reactive Approaches

To conserve the server network I/O bandwidth requirement, two approaches, *static multicast* and *dynamic multicast*, have been studied. In the static multicast approach, a video server serves a batch of requests for the same video that arrive within a short period of time using one server channel. This approach is also known as *Batching*. All clients of the same batch receive the same data from the same multicast tree. The difference among the different schemes in this approach is the policy to select which batch to serve first when a server channel becomes available. The dynamic multicast approach extends the static multicast approach, allowing late coming requests to join a batch currently being served by extending the multicast tree to include the newly arriving client. The subsequent sections describe several Static and Dynamic Multicast techniques.

2.1.1 Static Multicast

In this environment, users requesting the same video, within a short period of time, are served together using the multicast facility. Since there could be several such batches of pending

requests, a scheduler selects one to receive service according to some queuing policy. Some scheduling techniques for the Batching approach are as follows:

- *First-Come-First-Serve (FCFS)* [AWY96, DSS96]: As soon as some server bandwidth becomes free, the batch containing the oldest request with the longest waiting time is served next. The advantage of this policy is its fairness. Each client is treated equally regardless of the popularity of the requested video. This technique, however, results in a lower system throughput because it may serve a batch with few requests, while another batch with many requests is pending.
- *Maximum Queue Length First [AWY96]:* This scheme maintains a separate waiting queue for each video. When server bandwidth becomes available, this policy selects the video with the most number of pending requests (i.e., longest queue) to serve first. This strategy maximizes server throughput. However, it is unfair to users of less popular videos.
- *Maximum Factored Queued Length First [AWY96]:* This scheme also maintains a waiting queue for each video. When server resource becomes available, the video v_i selected to receive service is the one with the longest queue weighted by a factor $1/\sqrt{f_i}$, where f_i denotes the access frequency or the popularity of v_i. This factor prevents the system from always favoring more popular videos. This scheme presents a reasonably fair policy without compromising system throughput.

Figure 2.1 depicts the static multicast schemes. Since users have to wait in a queue to get the video data, it is important to note that the aforementioned batching policies can only provide a

near-on-demand service.

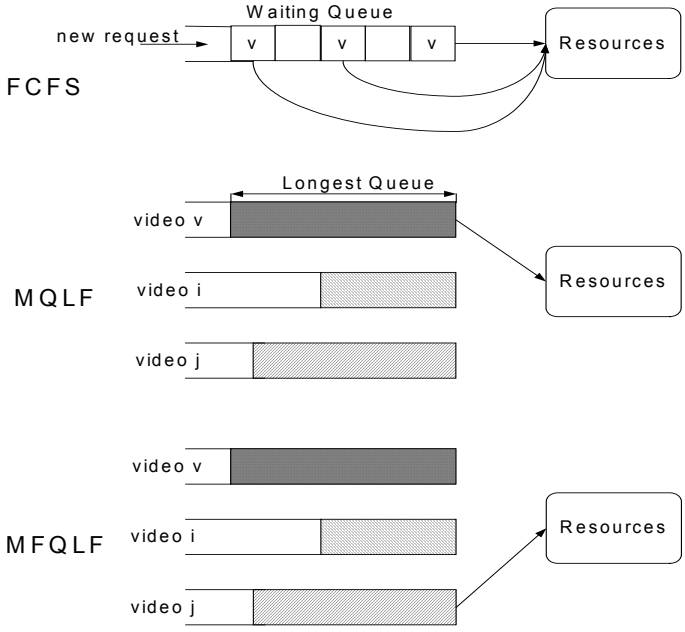


Figure 2. 1 FCFS, MQLF, and MFQLF

2.1.2 Dynamic Multicast

Unlike the static multicast approach, the dynamic multicast approach can offer true video-on-demand services while providing high throughput. This is achieved by letting late arriving requests for the same video to be serviced by dynamically expanding the already constructed multicast tree. In Adaptive Piggybacking [GM96], the server slows down the delivery rate of the

video stream to a previous client, and speeds up the delivery rate of the video stream to a new client until they share the same play point in the video. At this time, the server merges the two video streams, and uses only one channel to serve the two clients. *Chaining* [SHT97] introduced the peer-to-peer streaming paradigm. By caching portions of the video data, clients can forward the video to other downstream clients, lessening the burden on the video server. In fact, client nodes form a delivery chain, and the server delivers the video through that chain using a single data stream. A new client either gets the video from an existing chain or from a new chain if the current chain cannot serve the request. The main advantage of Chaining is that not all requests need be serviced from the video server and the video content is made available throughout the network at client nodes forming the chain. *Patching* schemes [HCS98b, CHV99, CL99, EVZ99, SRT99b, THT02] let a new client join an ongoing multicast and still receive the entire video data stream. For a new request for the same video, the server delivers only the missing portion of the requested video in a separate *patching stream*. The client downloads the data from the patching stream and immediately displays the data. Concurrently, the client downloads and caches the later portion of the video from the multicast stream. When finishing playing back the data in the patching stream, the client switches to play back the video data in its local buffer. Figure 2.2 illustrates an example of patching. It shows that users C_1 and C_2 are served together with a single multicast stream. At t time units later, user C_3 requests the same video. C_3 joins the multicast tree and starts buffering arriving video data, while at the same time C_3 downloads the missing portion via a *patching stream*. It is important to note that in patching, a user would get the video only if it is capable of simultaneously downloading from two streams (a regular

multicast and a patching), and it has enough buffer space to absorb the time skew *t*. If these conditions do not hold, a new multicast stream is needed.

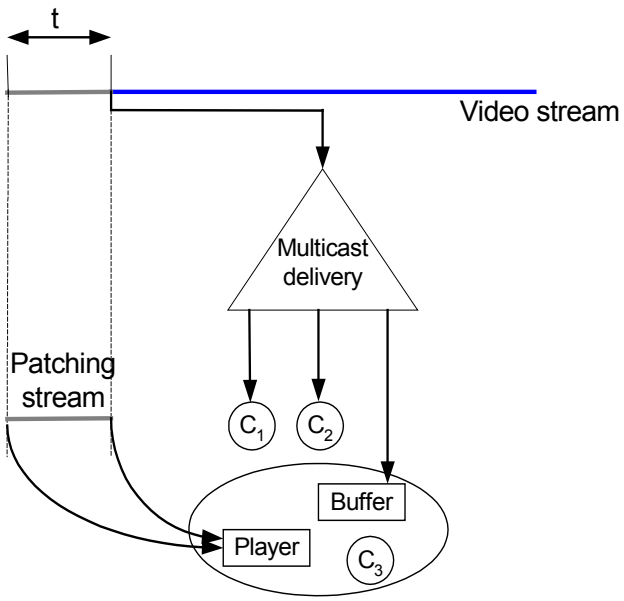


Figure 2. 2 Patching

2.2 Periodic Broadcast: Proactive Approaches

Periodic Broadcast is a highly scalable solution for streaming popular videos. Several periodic broadcast schemes have been proposed in recent years [DS94, AWY96b, VI96, HS97, JT97, JT97b, GKT98, HCS98, PCL98, PCL98b, PCL98c, JT98, PCL99, HU01, MEVS01, SGT01, GKT02, YK03, KS04]. In these approaches, a video is fragmented into a number of segments.

Each segment is periodically broadcast on a dedicated channel. A periodic broadcast system is highly scalable since it can serve a very large community of users requesting for the same video with minimal server bandwidth. The server bandwidth requirement is independent of the number of users the system is designed to support. The client tunes into one or more channels concurrently to download the broadcast segments into its buffer, while local playback software renders the video data in this buffer onto the screen. Periodic broadcast techniques guarantee that the client downloads a segment before its required playback time.

In this environment, if the client misses the beginning of the broadcast of the first segment, the client must wait until the next broadcast of the same segment. The worst service latency, therefore, is the time period between the consecutive broadcasts of the first video segments. Since the size of the first segment can be made very small, this approach can provide near-video-on-demand services. Many periodic broadcast techniques have been designed to keep the worst service delay small by making the first segment as small as possible while guaranteeing a jitter free playback at the client. Existing periodic broadcast schemes can be classified into two major categories, namely the *Server-Oriented Category* and the *Client-Oriented Category* [HTT04, HT03]. Techniques in the first category reduce service delays by increasing server bandwidth. On the contrary, techniques in the second category reduce the delays by requiring more client bandwidth.

2.2.1 Server-Oriented Category

Staggered Broadcasting [DS94] is the earliest and simplest video broadcasting technique. This scheme staggers the starting times for broadcasting a video evenly across available channels. The difference in the starting times is referred to as the *phase offset*. Because a new stream of the same video is started every phase offset, it is the longest time any client needs to wait for this video. The advantage of Staggered Broadcasting scheme is twofold. First, clients download data at the playback rate. Second, the clients do not need extra storage space to buffer the incoming data. This scheme, however, scales only linearly with the increase in the server bandwidth. Figure 2.3 shows two videos 'i' and 'j' periodically broadcast using 4 and 3 channels.

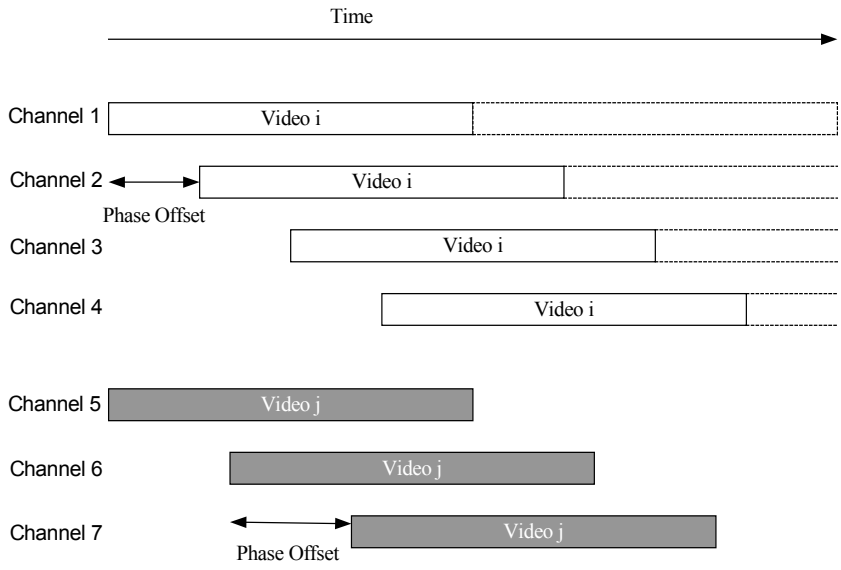


Figure 2. 3 Staggered Periodic Broadcast of two different videos

Pyramid Broadcasting [VI96] addresses this drawback by broadcasting the video segments at a very high data rate, and allowing the clients to prefetch data into a local buffer. In this scheme, video segments are of geometrically increasing sizes, and the server network bandwidth is evenly divided to periodically broadcast one segment in a separate channel. This solution requires expensive client machines with enough bandwidth to cope with the high data rate on each broadcast channel. *Permutation Based Broadcasting* [AWY96b] improves this condition by dividing each channel into s sub-channels that broadcast a replica of the video fragment with a uniform phase delay. This strategy reduces the requirement on client bandwidth by some factor s although the data rate remains very high, which can still flood the prefetch buffer with half of the total data [HS97].

In *Skyscraper Broadcasting* [HS97], the server bandwidth is divided into *several* logical channels of bandwidth equal to the playback rate of the video. Each video is fragmented into several segments, and the sizes of the segments are determined using the following series referred to as the *broadcast series*; [1, 2, 2, 5, 5, 12, 12, 25, 25, …]. In other words, if the size of the first data segment is x, the size of the second and third segments are $2 \cdot x$, the fourth and fifth are $5 \cdot x$, sixth and seventh are $12 \cdot x$, and so forth. This scheme limits the size of the biggest segments (W-segments) to W units or $W \cdot x$. These segments stack up to resemble a skyscraper of a width W, thus the name Skyscraper Broadcasting. The server repeatedly broadcasts each segment on its dedicated channel at the playback rate of the video. To download the video, each client employs two synchronized threads - an *Odd Loader* and *an Even Loader*. They download

the odd groups, each consisting of segments of odd size, and the even groups, each consisting of segments of even sizes, respectively. When a loader reaches the first *W*-segment, the client uses only this loader to download the remaining *W*-segments sequentially to minimize the requirement on the client buffer space. The main advantage of this technique is the fixed requirement on client bandwidth regardless of the desired service latency. To offer better service latency, one needs only add server bandwidth. Figure 2.4 shows the downloading scheme of clients 'X', 'Y', and 'Z' arriving at different times. Clients 'Y' and 'Z' will download the same segment at channel 6. Figure 2.4 also shows the play times for different segments relative to client 'X'.

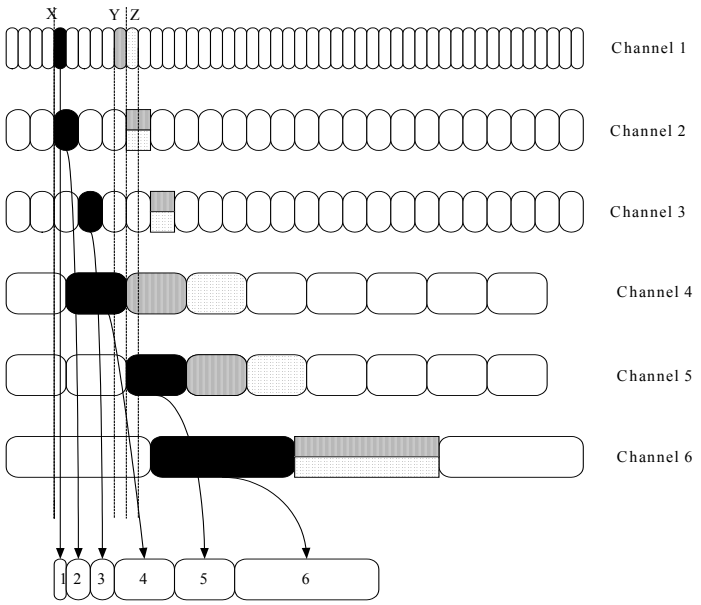


Figure 2. 4 Skyscraper Downloading Scheme

Client-Centric-Approach (CCA) [HCS98] is a periodic broadcast technique that reduces the service latency by adding only server resources once clients' resources have been determined. In fact, CCA can be considered as a generalization of Skyscraper Broadcasting in that each transmission group can have more than two segments, and the number of data loaders is not limited to two as in Skyscraper Broadcasting. In contrast to the Skyscraper scheme, CCA can leverage extra client bandwidth, if available, to further reduce access latency.

2.2.2 Client-Oriented Category

The techniques in this category increase the requirement of both server and client bandwidth in order to reduce the service delay. *Harmonic Broadcasting* (HB) [JT97] initiates the techniques in this category. HB fragments a video into segments of equal sizes, and periodically broadcasts each segment on a dedicate channel. The channels, however, have decreasing bandwidths following the Harmonic series. In other words, the first channel is allocated a bandwidth equal to the playback rate of the video; the second channel has the bandwidth of half of the playback rate; the third channel has one third, and so forth. The client downloads segments from all channels concurrently. The original HB, however, cannot deliver all the segments on time. A simple delay, before consumption of the first segment, equal to the size of one segment solves this problem [PCL98]. *Caution Harmonic Broadcasting* [PCL98], *Quasi-Harmonic* and *Poly-Harmonic Broadcasting* [PCL98b] also address this problem. Although theses schemes use many

channels to broadcast a video, the total bandwidth grows slowly following the Harmonic series, typically adding up to only *five* or *six* times the playback rate of the video. However, these schemes present another problem that involves the use of numerous channels (e.g., 240 channels are required for a 2-hour video if the latency is kept under 30 seconds). Since the client must concurrently obtain video data segments from many channels, a storage subsystem with the capability to move their read heads fast enough to multiplex among so many concurrent streams would be very expensive. To solve this problem, *Pagoda Broadcasting* has been proposed [PCL98c]. This scheme also divides each video into segments of equal sizes. However, it addresses the problems of too many channels by broadcasting more than one segment on some channels. Pagoda Broadcasting does not require much more bandwidth compared with Harmonic Broadcasting and its variants and at the same time does not use as many channels.

2.2.3 Server-Oriented vs. Client-Oriented

The techniques in the Client-Oriented category have many drawbacks compared to those in the Server-Oriented category. First, the client must have network bandwidth equal to the server bandwidth allocated to the longest video. The requirement on the client bandwidth is therefore very high, making the overall system very expensive. Second, to improve access latency, it will require adding bandwidth to both server and client, which makes the system enhancement very costly. The justification for the Server-Oriented approach is that server bandwidth, shared by a large community of users, contributes little to the overall cost of the VOD environment. As a

result, these techniques are less expensive than the Client-Oriented approaches, which require a client to be equipped with substantially more client bandwidth. Nevertheless, if the bandwidth is readily available, these schemes or the CCA technique can be used. It is worth noting that CCA is not classified as a client-oriented approach because CCA allows the system to improve service latency by adding only server bandwidth as in Skyscraper Broadcasting.

2.3 Hybrid Schemes

Periodic Broadcast techniques are most beneficial for popular videos. However, in general, these techniques are not applicable to less popular videos or a varying video access pattern. It was shown that a hybrid solution that combines both on-demand multicast and periodic broadcast offer the best performance [HO02]. *Adaptive Hybrid Approach* periodically measures the popularity of each video based on the distribution of recent service requests [HO02]. Popular videos are periodically broadcast using Skyscraper Broadcasting, and less requested videos are serviced using Batching. The number of channels used for periodic broadcast depends on the combination of popular videos currently in the system. The remaining channels are allocated to batching.

2.4 Periodic Broadcast: Implementation Issues

With the many designs of periodic broadcast schemes exist, a few software prototypes have been developed and experimented with. For specific periodic broadcast techniques, *Greedy-Disk*

Broadcasting [BWGST01], *Striping-Broadcasting* [JB02], and a variant of Pagoda Broadcasting [TPL03] have been implemented. A generalized periodic broadcast server (GPBS) software that supports many periodic broadcast techniques has also been developed [TPTW04]. Greedy-Disk Broadcasting, Striping-Broadcast, and GPBS implementation is based on IP-multicast. The common observation made from all the prototypes is that caching portions of broadcast videos in server memory improves the number of concurrent videos that can be supported by a broadcasting server. GPBS indicates that to ensure a jitter-free broadcast from the server viewpoint, the Client-Oriented broadcast schemes such as Harmonic Broadcasting and its variants require much more server memory than those in the Server-Oriented category given the same server disk and network bandwidth. Given sufficient network bandwidth, the increase in server disk bandwidth has more impact than the increase in server memory in terms of supporting more broadcast videos for the broadcast schemes in the Server-Oriented category. Although these prototypes present a clear feasibility of the periodic broadcast approaches, they are not well suited for an unreliable environment such as the Internet where communication is very lossy. In fact, it was shown in the Greedy-Disk-Broadcasting [BWGST01, BWGST01b] implementation that delivering a video from east to west of the United States can cause a packet loss factor up to 20%.

One of the practical considerations that arise on the use of IP multicast is the time division multiplexing of a finite number of multicast channels among users. The work presented in [BWGST01] separates control and data operations by incorporating distinct modules for server and client functionalities. Figure 2.5 illustrates this architecture. The *Server Control Engine* is

responsible for handling interactions between the server and the client. It also computes a *transmission schedule* that identifies video segments that should be retrieved from disk and transmitted over the network and a *reception schedule* that specifies how the client should receive video data. These interactions are based on the Real Time Streaming Protocol (RTSP) and the Session Description Protocol (SDP). The *Server Data Engine* coordinates with the *Server Control Engine* to retrieve video data from disk and transmit the data to the client using the *Real-time Transport Protocol (RTP)* and communicates with The *Server Control Engine* through a *TCP* connection. The *Server Control Engine* is implemented as a multithreaded single process system, where a single thread places all incoming requests into a specific queue. For each request, there exist a scheduler thread that constructs an abstract *transmission schedule* and a *reception schedule*. Because in [BWGST01] a patching technique and a periodic broadcast technique are implemented, the transmission schedule is made general and relies on a linked list of work requests for each multicast channel. A work request is a data structure that contains the time to begin the transmission of a specific segment, the time to finish the transmission, a pause time, and a count number to illustrate the number of times the segment is sent. The latter parameter is used for the periodic broadcast technique. In this way, the scheduler thread determines whether the client can be satisfied using an already scheduled transmission, or an additional schedule of part of the video is needed. Based on the server transmission scheme the scheduler thread computes a reception schedule for the specific client and transmit it. Because the work requests linked list is scheduled ahead of time, it can be used to search for free intervals to allow a channel/IP-address reuse. The *Server Data Engine* is also a multithreaded single

process entity. For each video that is being transmitted two separate thread are created, a disk thread that retrieves the video data from disk and network thread responsible to transmit data from buffers to the network according to the transmission schedule. Both threads are synchronized in separate rounds in order to reduce the impact of disk overheads and avoid injecting burst of traffic into the network.

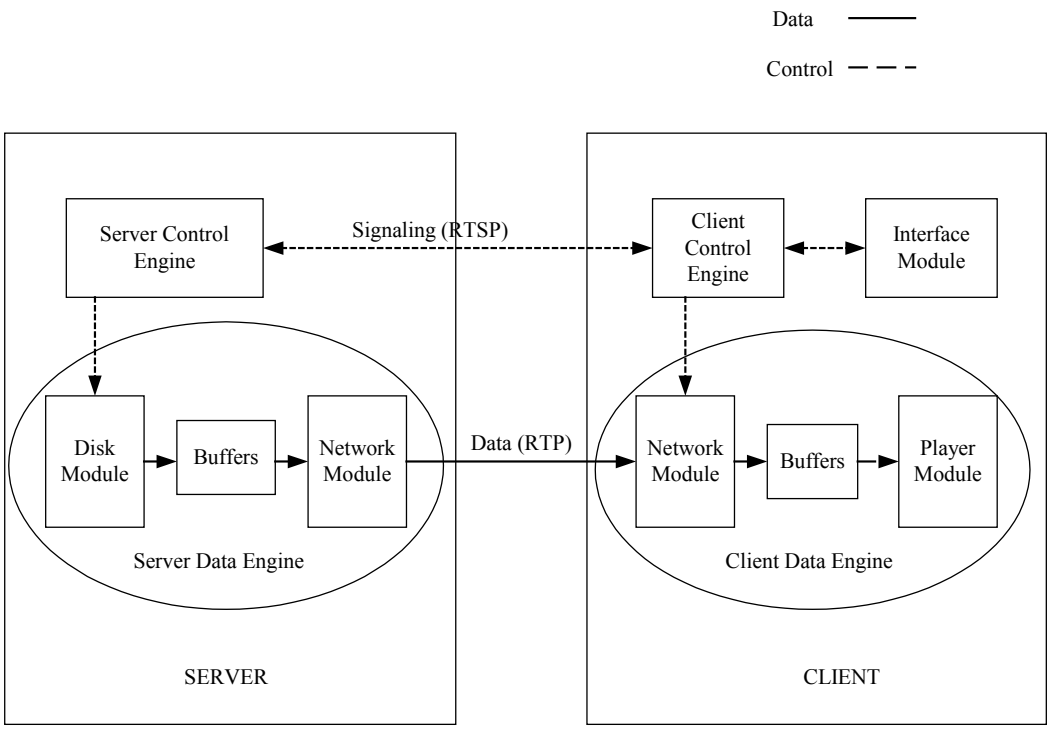


Figure 2. 5 Streaming Server and Client Architecture

In the *Striping Broadcasting* [JB02], the server software is composed of three modules, namely a

Data Retrieval Handler, a *Video Delivery Handler* and *Director Manager*. The Data Retrieval Handler and Video Delivery Handler are two separate threads that are created by a *Coordinator* once a new video is requested. The retrieval handler brings in multiple blocks of each segment from the disk to streaming buffers in rounds. Each retrieval buffer operates its own buffers, this is due to the fact that different videos can have different playback rate. Once the buffer is full, the retrieval thread goes into a sleep state until the delivery thread wakes it that is after purging some video blocks from the buffer. The delivery thread is responsible to periodically broadcast video data stored in the buffers on the server channels using their associated multicast address. The video thread broadcast one block[1] of each segment in a round-robin fashion for each playback duration of a block. The delivery thread arranges blocks into UDP packets. Each packet header will therefore contain the location of the block in the video file and will also serve as a timestamp or sequence number to reorder video data at the client site. The *Directory Manager* maintains all information regarding the video being broadcast and also relevant broadcast parameters such as the multicast address, port number of the first channel and playback rate of the video. It is important to notice that once an end client receives the first segment, the client software will allocate specific loaders to download later segments in a straightforward manner. Therefore, the video delivery thread attach to each first segment a *Broadcast Tag* that will be used by the client to compute its reception schedule.

Although both implementations discussed above use a *constant-bit-rate* (CBR) that equals to the playback rate of the video to deliver data, all periodic broadcast protocols can support variable-bit-rates (VBR) encoded video by mapping it to a constant-bit-rate stream using the peak

[1] Each video segment consists of several blocks, blocks are striped using the striping algorithm in [BJ02]

bandwidth of the video as the bit rate of the stream or by using better mapping approaches [HU01, SDKST97].

IP-multicast was originally designed for broadcasting situations where recovery is not needed for lost data. NACK-based and Tree-based protocols have been proposed to allow the receivers to acknowledge lost packets. In the Tree-based protocol, ACKs and NACKs are managed by the receiver's parent in a tree structure. The problem with these solutions is that the multicast efficiency decreases with the increasing number of receivers. This challenge has led many researchers to consider applying *forward-error correction* (FEC) to multicast. The main idea behind the use of FEC codes is to transmit the original source video data, in the form of sequence of some packets, as well as additional redundant packets, where the redundant information can be used to recover lost data packets at the receivers. Digital Fountain takes this approach in their periodic broadcast product [HKLLR01]. It allows a server to periodically multicast streams of FEC encoded data packets. Clients tune into one or more multicast groups to receive the packets, and are able to reconstruct the original data using the FEC codes in the event of packet loss.

2.5 Summary

A key aspect of designing a scalable VOD system is to leverage multicast and broadcast to facilitate bandwidth sharing. For popular videos or if the video database contains only few videos, a periodic broadcast technique achieves the best cost/performance. This approach, however, cannot deliver videos without some delay. Two alternatives have been considered for

addressing this limitation: the *Server-Oriented Approach* (e.g., Skyscraper Broadcast [HS97], Client-Centric Approach [HCS98]) reduces service delay by increasing server bandwidth; whereas the *Client-Oriented Approach* (e.g., *Cautious Harmonic Broadcast* [PCL98]*, Pagoda Broadcast* [PCL98c]) requires client to equip with significantly more download bandwidth. For wide-area deployment, the *Server-Oriented Approach* is preferred because the cost of server bandwidth can be shared by a very large community of users, and therefore contributes little to the cost of the overall system. *Client-Oriented Approach* is limited to local deployment on an intranet where client bandwidth is abundant (e.g., receiving the video at five times the playback rate). Applications such as corporate training, campus information systems can benefit from these techniques.

For less popular videos, multicast on demand is a better solution then repeatedly broadcasting the videos. Standard multicast, however, makes users wait for the batching period. *Patching* techniques resolve this problem by allowing the clients to join an ongoing multicast while playing back the missing part of the video arriving in a patching stream. For a better performance, a system should use both periodic broadcast and multicast. Hybrid techniques, such as in [HO02], can be used to monitor the popularity of the videos, and apply the best mechanism to deliver them.

CHAPTER THREE: INTERACTION WITH BROADCAST VIDEO

In video-on-demand applications, it is desirable to provide the user with the *video-cassette-recorder*-like (VCR) capabilities such as fast-forwarding a video or jumping to a specific frame. In a periodic broadcast environment where each video is broadcast repeatedly on the network, existing techniques rely on data prefetching as the mechanism to provide this functionality. This approach provides limited usability since the prefetching rate cannot keep up with typical fast-forward speeds. Fast-forwarding a video for several seconds would inevitably exhaust the prefetch buffer. In this chapter, this practical problem is addressed by repeatedly broadcasting the interactive versions of the videos. For instance, an interactive version might contain only every fifth frame in the original video. The client software leverages these "interactive" broadcasts to provide better VCR service. Section 1 discusses some related work for providing VCR functionalities in a VOD environment. Section 2 briefly explains the Client-Centric Approach technique (CCA) [HCS98] a proactive scheme that belongs to the server-oriented family, and describes a new Broadcast-based Interactive Technique (BIT) [THS02, THS02b] that extends the CCA method to provide VCR functionalities. Section 3 formally proves the correctness of the BIT approach, and section 4 compare its performance to a prefetch method, called *active buffer management.* Finally, a summary is given in Section 5.

3.1 VCR Interactions And Related Work

A video is a sequence of continuous frames. The frame being rendered by the user is the current

frame that is referred to as the *play point*. A user watching the video has the ability to change the position of the play point to another point (frame) that is called the *destination point*. One can distinguish between two different types of interactivity:

Continuous interactive functions: These are the interactions where the user continuously advances the play point one frame at a time at a high speed. The user has full control of the duration of the interaction.

Discontinuous interactive functions: These are the interactions where the user instantaneously changes the play point to a future frame (i.e., jump-forward) or a frame in the past (i.e., jump-backward).

One can further divide all interactions into *forward interactions* and *backward interactions*. If Δt is the time taken by the interaction, Δl the video length of the interaction in time unit, i.e., the difference between the play point and the destination point, and b the playback rate of the video, the parameter x, given in Equation 3.1, can determine all types of interactions. Table 3.1 gives the possible forward interactions, and table 3.2 gives the backward interactions with the possible values of the parameter x.

$$x = \frac{\Delta l / \Delta t}{b} \tag{3.1}$$

Table 3. 1 Forward Interactions

Action	Play	Fast Forward	Jump Forward
x	1	$(1, x_1)$	$+\infty$

Table 3. 2 Backward Interactions

Action	Pause	Play Backward	Fast Reverse	Jump Back.
x	0	-1	$[-x_1,-1)$	$-\infty$

A *play* action is when the length of the action over the time taken by the action is equal to the playback rate. For fast-forward, $\frac{\Delta l}{\Delta t} > b$ and the value of x_1 will primarily depend on Δl and Δt. In the case of a jump forward, since the action is instantaneous, $\Delta t = 0$ and therefore x is infinite. A pause action is when $\Delta l = 0$. Since Δl is negative in a play-backward the value of x is –1. Fast Reverse and jump-backward are the opposite of fast forward and jump-forward, respectively.

It is worth noting that the pause action belongs to the backward interactions. This is due to the fact that incoming frames make the destination point moves backward of the current play point. To complete the canonical world of interactions, the slow forward, $x \in (0,1)$ and the slow backward, $x \in (-1,0)$ can be added. Both interactions are backward interactions for the same reason as the pause interaction.

An early work on providing interactive service in a multicast environment is presented in [AA94] and [AA96]. In this scheme, if the data in the prefetch buffer cannot accommodate a jump request, the client requesting the interaction is moved to an existing stream whose play point matches the client's destination point. If such a stream does not exist, the server issues an emergency stream to provide the service. Emergency streams are expensive since they serve

only one client. To reduce this cost, the techniques, introduced in [LL97] and [APS98], try to move the client of an emergency stream to an existing multicast whose play point is ahead of the client's play point, but not further than some amount. In any case, using emergency streams defeats the benefits of the multicast paradigm. This approach is too expensive to provide VCR-like service to a large user community.

In the broadcast environment, it was observed in [FKMA99a] [FKMA99b] that the play point can be maintained at the middle of the video segment currently in the prefetch buffer in order to accommodate interactive actions in either forward or reverse direction equally well. This is accomplished in [FKMA99b] by selectively prefetching the segments to be loaded depending on the current position of the play point. This scheme is referred to as the *Active Buffer Management* (ABM) scheme. In general, ABM can be set to take advantage of the user behavior. If the user shows more forward actions than backward actions, the play point can be kept near the beginning of the video segment in the buffer, and vice versa. This scheme has been shown to offer better performance than conventional buffer management techniques.

3.2 Broadcast-Based Interaction Technique (BIT)

The proposed technique is an extension of the Client Centric Approach. It is assumed that two versions of the same video exist: a normal version and a version compressed by a factor *f* that we call the *compression factor*. An example of compression could be selecting each *f* frame of the original video. The user watching the compressed version of the video at the playback rate will

have the impression of fast playing the normal video.

Because the proposed technique is based on the Client-Centric Approach (CCA) [HCS98], CCA is briefly described in this section. The channel design and data fragmentation related to BIT are explained, then section 2.3 shows how BIT receive and play back the video, and explains how VCR functionalities can be achieved. Since broadcast is used as the mechanism to provide VCR service thus the name Broadcast-Based Interaction Technique (BIT). The motivation for this approach (instead of using emergency channels) is to retain the most desirable property of the broadcast approach, namely unlimited scalability - the bandwidth requirement is independent of the number of users the system is designed to support.

3.2.1 Client-Centric Approach (CCA) Broadcasting

A periodic broadcast design consists of a channel design, a data segmentation technique, a broadcast schedule, and a playback strategy. The CCA is primarily designed to take full advantage of the client's capability, i.e., the client buffer space and its downloading bandwidth.

CCA divides the communication bandwidth of the server into K logical channels, where every channel has the bandwidth of the playback rate. Each channel i periodically broadcasts one segment S_i of the video. The segmentation technique uses the following recursive equation to fragment a video V into segments S_i:

$$f(n) = \begin{cases} 1, & \text{if } n = 1, \\ 2 \cdot f(n-1) & \text{if } n \bmod (c) \neq 1, \end{cases} \qquad (3.2)$$

$f(n-1)$ if $n \bmod (c) = 1$.

The closed form of this segmentation function can be presented as:

$$f(n) = 2^{\,n - \lceil n/c \rceil}, \text{ where } 1 \le n \le K. \qquad (3.3)$$

The parameter c denotes the maximum number of fragments a client can download at one time. To conserve buffer space, the sizes of the largest segment are fixed to $W \cdot D_1$, where D_1 denotes the playback duration of the first segment. In the remainder of this section those segments are referred to as *W-segments*. CCA organizes the video fragments into $g = \lceil K/c \rceil$ transmission groups, each with c items except the last one. The size of the last segment of a group is equal to the size of the first segment in the next group. Figure 3.1 illustrates the data fragmentation, channel design, and transmission groups for the CCA technique, where each group contains 3 fragments.

To play back a video, the client starts using c loaders and a video player. Each loader downloads its predetermined segments sequentially at the playback rate. When a loader finishes downloading a segment S_j from a group i, the loader is allocated to the next group $i+1$ to download segment S_{j+c} at its next broadcast. Once the first *W-segment* is encountered, only one loader is used to download all the *W-segments* sequentially.

The motivation behind the CCA segmentation technique is to wisely utilize the full capacity of the client resources, while guaranteeing the continuous playback of the video.

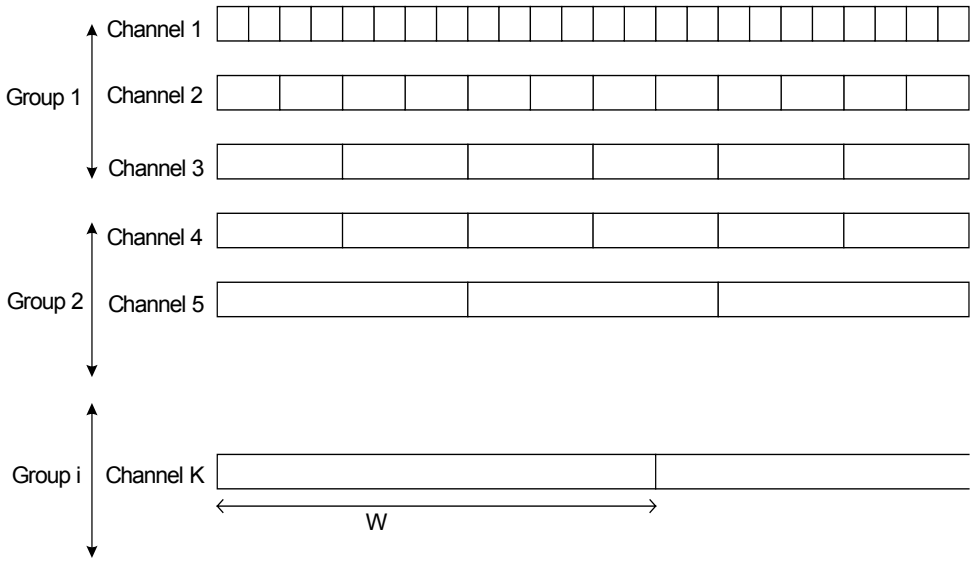


Figure 3. 1 CCA with *K* channels and 3 channels per group

Since two consecutive segments in the same group either start at the same time or finish at the same time (Figure 3.1), the continuity in playing back segments from the same group is guaranteed. Since the first loader always downloads the first and smallest segment of a group, this loader is available before the consumption of the last segment of the same group. This loader, therefore, can be used to download the first segment of the next group. Segments across a group boundary are always of the same size, so the continuity across group boundaries is satisfied.

3.2.2 Channel Design and Data Segmentation

As in the CCA technique, the server bandwidth is divided into K logical channels. Each channel periodically broadcasts a different segment of the video. If we refer to K as the total number of channels, K can be divided into $K = K_r + K_i$, where K_r is the number of *regular channels* used to broadcast the regular version of the video, and K_i the number of *interactive channels* used to broadcast the compressed version of the same video.

For the normal version the same fragmentation technique designed for CCA is utilized (refer to Section 2.1 equation (3.2)). The segment broadcast by a *regular channel i* are denoted as S_i , and its compressed version as S_i'. The segments of the compressed version are concatenated into group of f as follows:

Group $1, V_1$: S_1' S_2' ... S_f'

Group $2, V_2$: S_{f+1}' S_{f+2}' ... S_{2f}'

.

Group i, V_i: $S_{(i-1)f+1}' S_{(i-1)f+2}' \ldots$ S_{if}'

.

Group K_i, V_{Ki}: $S_{(ki-1)f+1}'$ $S_{(ki-1)f+2}'$ ... S_{Kr}'

For ease, each of these groups are referred as a *compressed segment* hereafter. Each compressed segment i, denoted by V_i, is broadcast on an interactive channel c_i. Thus, the number of interactive channels is determined by the number of compressed segments, or $K_i = K_r / f$. Figure 3.2 shows the channel design for the BIT technique with a compression factor f of 4. *Cr*'s

represent regular channels and *Ci*'s interactive channels. One can note that there is one interactive channel for every four regular channels. As an example, Table 3.3 shows the size in minutes of the compressed segments when *K=30*, *c =3*, *f* =4, the maximum segment size is 7 minutes, and the length of the video is 120 mn.

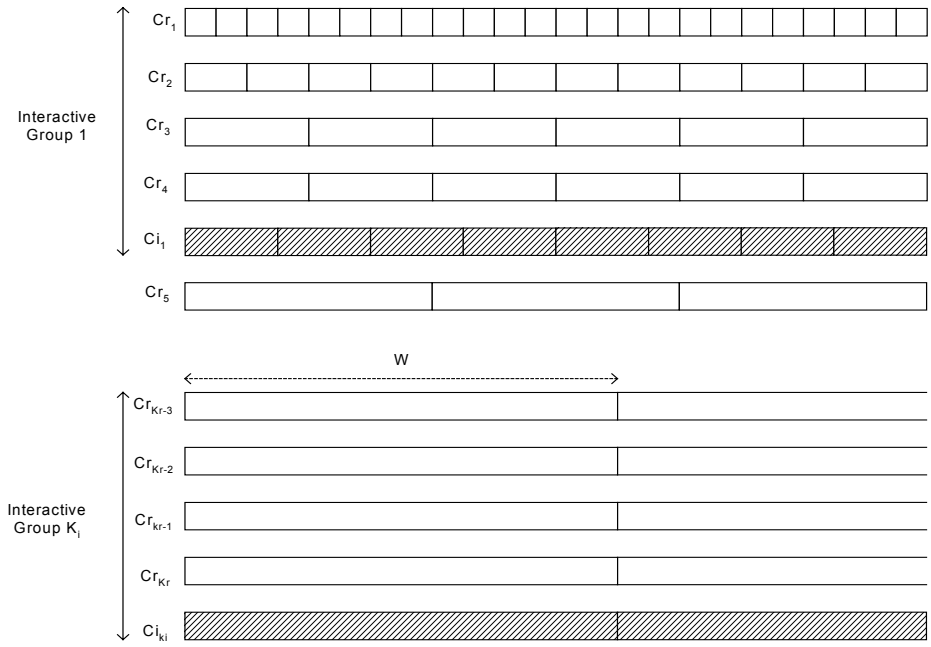


Figure 3. 2 Broadcasting scheme for BIT (K_r +K_i channels, *f* =4)

In Figure 3.2 a compressed segment is different from a compressed version of a regular segment. In this example, a compressed segment is the concatenation of four compressed versions of regular segments.

Table 3. 3 Size in minutes of the compressed segments

Segments	V_1	V_2	V_3	V_4	V_5	V_6
Size	0.29	1.87	6.84	7	7	7

3.2.3 Implementing VCR functions with BIT

The client storage space is divided into two parts: a *normal buffer* holds the normal video, and an *interactive buffer* caches the compressed version of the same video. To achieve interactions using BIT, client nodes are required to have *c+2* loaders where *c* is the CCA parameter that refers to the *normal loaders* L_1,L_2, …, L_c. The two extra loaders L_{i1} and L_{i2}, called *interactive loaders*, are used to download the compressed segments.

A Client issuing an interactive function switches from normal mode to interactive mode. During the *normal mode*, the client renders the content of the normal buffer; and during the *interactive mode* the client renders the content of the interactive buffer. The size of the normal buffer should be large enough to store a W-segment in order to guarantee continuous normal playback [HCS98]. The size of the interactive buffer is set twice the size of the normal buffer to ensure good interactive service. To play back and provide VCR functions, one can identify two components that work together: (1) the *Player* responsible for rendering the frames and accepting interactions, (2) the *Loader* responsible for tuning to the appropriate channels and

downloading the segments. These two components are discussed in the next sub-sections.

3.2.3.1 BIT Player

The player keeps playing the content of the normal buffer until it accepts a request for a VCR operation. At this time, the behavior of the player is determined by the content of the buffer, the closest point being broadcast with respect to the current play point, and the type of the interaction. The player behavior is explained in the following:

- *Play Action:* During the playback of the video, the player renders the current play point of the normal buffer, and the play point of the interactive buffer moves accordingly.
- *Fast-Forward/Fast-Reverse/Pause actions:* When a continuous action is initiated, the player switches to the interactive mode to render the frames in the interactive buffer. Two scenarios can follows: (1) If the user resumes the normal play before this buffer is exhausted, the player switches to the normal mode, and allocates the loaders to download the appropriate segments such that the normal playback is resumed at a frame closest possible to the destination point. Such a point is referred to as *closest point* hereafter. (2) If the user continues the interactive action beyond the frames in the interactive buffer, the player forces the user to resume the normal play by setting the destination point to the newest frame of the interactive buffer in case of a Fast-Forward action, or the oldest frame in case of a Fast-reverse or pause actions. The player then allocates the loaders, and resumes the normal play at the closest point as in the previous case. One notes that jumping to the closest point will show some discontinuity that is

referred to as a *displacement*. In practice, it can be avoided by allocating sufficient buffer space for a given application. It is also interesting to note that the parameter *x* (refer to section 1) has a value 1 for both Fast-forward and Fast-Reverse actions, which is due to the use of the same play back rate in the normal mode as well as in the interactive mode.

- *Jump Forward/Jump Backward actions***:** If the destination point is in the normal buffer, the player simply moves the play point to the destination point, and continues the normal play. Otherwise, the player allocates the loaders to start downloading the appropriate segments, and continues the normal play from the closest point. During these types of interactions there is no switch of modes.

The algorithm for the *player* is described formally in Figure 3.3.

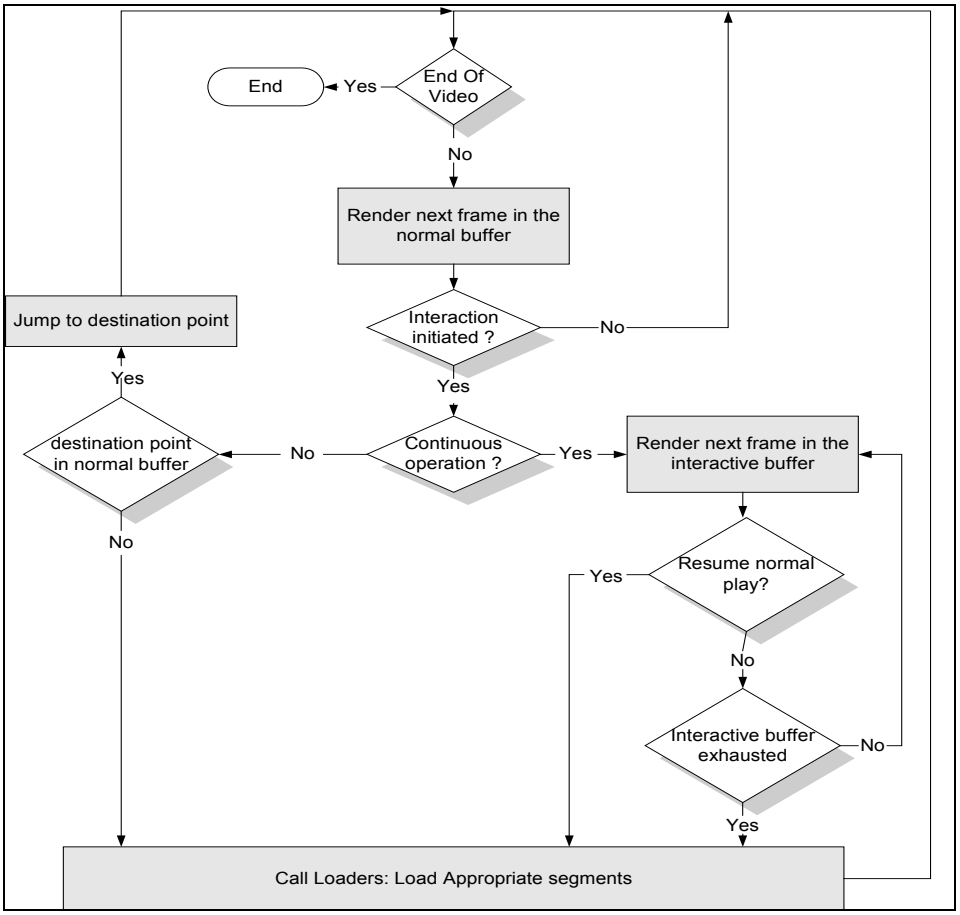


Figure 3. 3 BIT Player

3.2.3.2 BIT Loader

The *Loader* is responsible for tuning to the appropriate channels and loading the segments. When no interaction is taken place, the behavior of the loader is similar to the CCA technique. Playback of a video consists of two phases. First, the client downloads and plays back the unequal-sized fragments during the *unequal phase*, and then continues with the equal-sized

fragments during *equal phase*.

A client node uses c+2 loaders if the current segment, i.e. the segment viewed by the user, is in the unequal phase. Only three loaders are needed if the current segment is in the equal phase. Either way, if the current segment is in the first half of its interactive group $group_j$, the two interactive loaders are allocated to the previous and to the current interactive groups, i.e. $group_{j-1}$ and $group_j$. If, however, the current segment is in the second half of its interactive group, the interactive loaders are allocated to the current and future interactive groups, i.e. $group_j$ and $group_{j+1}$. By prefetching the compressed segments, one can make sure that the play point of the interactive buffer is always in the middle. Such choice will depend on the behavior of the user. For instance, users initiating more forward actions than backward actions can set the loader to always prefetch $group_j$ and $group_{j+1}$.

When an interactive operation is completed, if the VCR action was continuous or if the destination point exhausted the buffer, then the loader reallocates the *normal* and *interactive* loaders to the appropriate segments. If the interactive segments are not being downloaded or are not in the interactive buffer, the segments are downloaded immediately; otherwise no action is taking place. Once the current segment is being downloaded and viewed, the play point of the interactive buffer is logically shifted to match the play point of the normal buffer.

Figure 3.5 gives the algorithm of the loader. Figure 3.4 shows the client buffer before and after a continuous operation with $f=4$. The play points of both the normal buffer and the interactive buffer are shown as arrows. Before the interaction begins, the normal play point is in segment S_{i+1}. Similarly the interactive play point is adjusted to the same frame. When the fast Reverse

interaction is completed, the loader starts downloading the destination point in segment S_{i-2}. Since the destination segment is in the second half of its group, interactive loaders are allocated to the current group and to the next group, i.e. V_{i-1} and V_I respectively. The interactive loader does not download the latter segments since they are already stored in the interactive buffer.

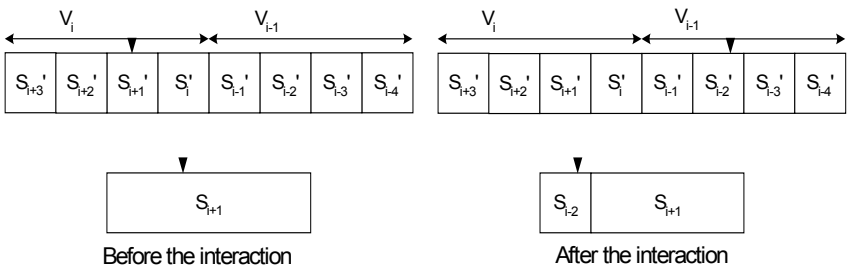


Figure 3. 4 Buffer Content Before and After a Fast Reverse interaction

It is assumed that the current play point is in the normal segment S_i

And S_i is in the interactive group j.

if (S_i is in the unequal phase)

Allocate the loaders L1,L2, …, Lc as in CCA;

Download the appropriate segments;

Call to allocate interactive loaders i.e. L_{i1} and L_{i2};

if (S_i is in the equal phase)

Allocate one loader L_k;

Download the segment

Call to allocate interactive loaders

// **Allocation of interactive loaders**

if(S_i is in the first half of its interactive group j)

Allocate L_{i1} to the interactive group j-1;

Allocate L_{i2} to the interactive group j;

Download the compressed segments if not in the buffer;

if(S_i is in the second half of its interactive group j)

Allocate L_{i1} to the interactive group j;

Allocate L_{i2} to the interactive group j+1;

Download the compressed segments if not in the buffer;

Figure 3. 5 Loader Algorithm

3.3 Analysis Of The Closest Broadcast Point

This section discusses the various cases that guarantee a normal play back after an interactive action is completed. The worst displacement that the BIT technique can encounter is given. Then, one can show that this worst case can be avoided by allowing the user in the interactive mode an extra small amount of time.

Figure 3.6 shows different cases of the closest broadcast point after the completion of an interactive operation. The parameter c is 3, however similar logic is applied for greater values. We refer to $D.p$ as the destination point, the points $B.p$'s are the broadcast points. The closest broadcast point relative to a destination point is differently studied in the unequal phase and in the equal phase as follows:

Unequal phase: (Figure 3. 6) during this phase, one can differentiate the following cases: (1) All segments are of unequal sizes, (2) The first two segments are of equal size but not the last one, and (3) the last two segments are of the same size but not the first one.

One can further distinguish the subsequent cases when all segments are of unequal size.

- *Case 1.1*: The segments are all left aligned. If segment i was loaded at its broadcast point, the user will consume it before the broadcast of the next cycle of segment $i+1$, similarly for segment $i+1$. Hence the closest broadcast point in this case is the broadcast point of segment $i+2$. When the loading of segment $i+2$ starts, the two other available

loaders are reallocated to the next round.

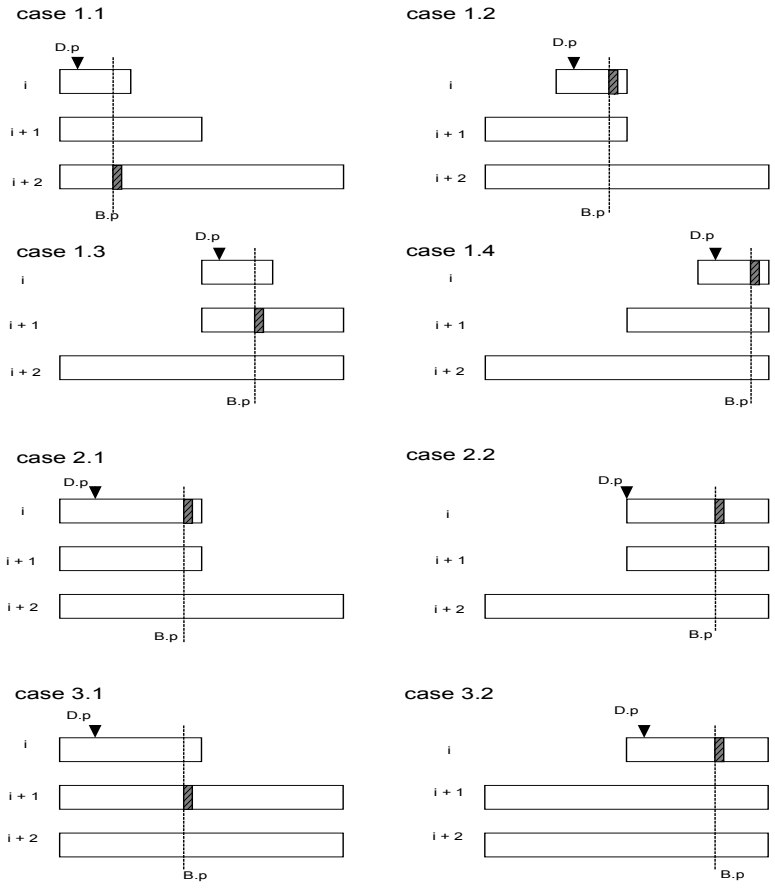


Figure 3. 6 Closest Broadcast Point during the Unequal phase

- *Case 1.2*: Since segments *i* and *i+1* are right aligned. The closest broadcast point is at segment i. After the consumption of segment *i*, the beginning of segment *i+1* is available and the beginning of segment *i+2* is available after the consumption of

segment *i+1*.

- *Case 1.3*: Segments *i* and *i+1* are not right aligned, but segments *i+1* and *i+2* are. Thus the closest broadcast point is at segment *i+1*. When the user consumes segment *i+1*, segment *i+2* is available.
- *Case 1.4*: Since all segments are left aligned, the closest broadcast point is at segment *i*.

When the first two segments are of equal size and the last segment is of a different size, two more cases are differentiated (*case 2* Figure 3.6). In both cases the closest broadcast point is at segment *i*. The difference between these cases is the time of loading segment *i+2* and *i+1*. For *case 2.1*, segment *i+2* is loaded after the consumption of segment *i+1*. While in *case 2.2*, segment *i+2* and *i+1* are loaded at the same time. Similar logic is applied for the third case where *case 3.1* is similar to *case 1.3* and *case 3.2* is similar to *case 2.2*.

Equal Phase: During the equal phase, the closest broadcast point is always at the same segment as the destination point. Since we use only one regular loader in the equal phase, once the user consumes segment *i* segment *i+1* is always available.

A minimal displacement occurs when the destination point is the closest broadcast point. However, there are situations where the displacement is maximal. In Figure 3.6, the worst displacement is shown in *case 1.1*. If the destination point is the first frame of segment *i* and the broadcast points are at the last frame of the same segment, because for this specific case the closest broadcast point is at segment *i+2,* then the worst displacement is twice the size of segment *i+1*.

If we denote by *k*, *k+1*, *k+2*, the segments of unequal sizes such that the *k+2* segment is the first

W-segment. Therefore, using formulae (3.3) (Section 2.1), the worst displacement is given by the following:

$$Worst\_diplacement = 2 \times 2^{k+1-\lceil k+1/c \rceil} \qquad (3.4)$$

Considering the *case 1.1* in Figure 3.6, if one can afford the user in the interactive mode an extra small amount of time, this specific case can be avoided. For example, when performing a Fast Forward action, by forcing the user to render the content of the interactive buffer, one can move from *case 1.1* to the case *1.2* that shows a smaller displacement.

3.4 Performance Study

To evaluate the performance of the BIT technique, detailed simulators to compare it with the *Active Buffer Management* (ABM) scheme are implemented. This section discusses the simulation study. Section 4.1 presents the user behavior model, section 4.2 discusses the performance metrics, and finally the simulation results are presented in section 4.3.

3.4.1 User Behavior Model

Users can initiate an interaction following the model given in Figure 3.7. Similar user interaction model have been introduced in [AS98] and also used in [FKMA99b]. This model gives the probabilities of issuing an interaction and the duration of each interaction. The probabilities define the frequency of interactions; and the m_i's define the average amount of video story, in time unit, fast forwarded or fast reversed in a continuous interaction. This

amount of continuous interaction is in terms of the original uncompressed version of the video. For instance, $m_{fr} = 50$ seconds indicating that the users, on the average, fast reverse 50 seconds of the video story; this is not the same as pressing the fast-reverse button for 50 seconds. Since *Jump Forward* and *Jump Backward* are instantaneous, their duration refers to the length of video skipped. A user starts playing the video with duration m_p. After this duration, the user may issue an interaction with the probability $P_i = 1\text{-}P_p$ or returns to play the video with the probability P_p. Once the VCR action is finished, the user always returns to play a portion of the video. For the sake of experimentation, the following assumptions to conduct our simulations are made.

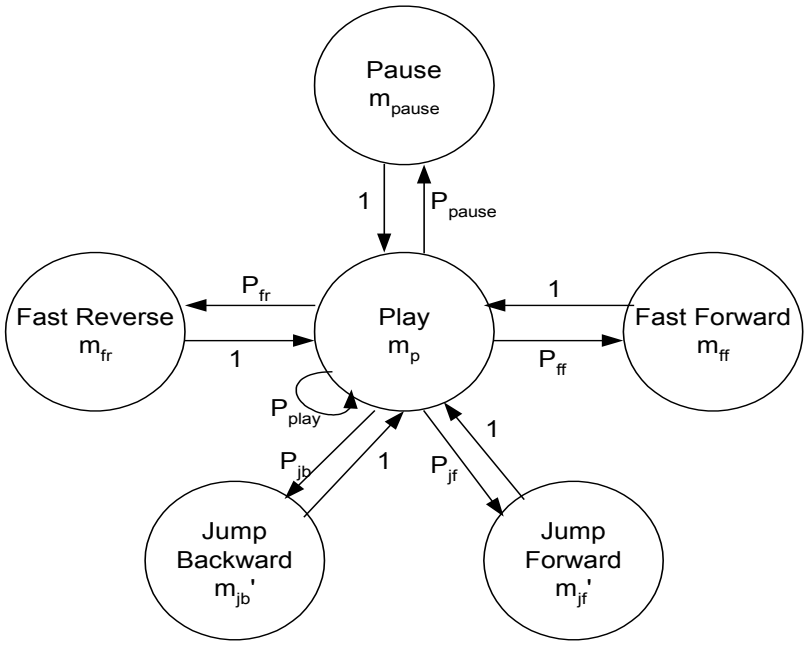


Figure 3. 7 User Interaction Model

It is assumed that durations for a play and for interactive actions are exponentially distributed with their respective mean shown in Figure3.7. Furthermore, the duration of all interactive operations are set equal with the same mean i.e. $m_{ff} = m_{fr} = m_{pause} = m_{jf} = m_{jb}$, that is referred to as m_i or the *interactive mean*. Finally, the interactive probabilities is set to be equal, i.e., $P_{pause} = P_{ff} = P_{fb} = P_{jf} = P_{jb}$. Therefore, $P_{pause} = P_{ff} = P_{fb} = P_{jf} = P_{jb} = P_i/5$. The *duration ratio dr* is referred to the average amount of interaction (m_i) over the average duration of a normal playback interval (m_p), hence $dr = \frac{m_i}{m_p}$. This ratio measures the degree of interaction.

3.4.2 Performance Metrics

An interaction can be successful or unsuccessful. An interaction is considered unsuccessful if the data currently in the buffers fail to accommodate the interaction. For instance, a jump to a destination point outside the buffer is considered unsuccessful. In the case of continuous actions, a long-duration fast-forward pushing the play point off the interactive buffer is considered as an unsuccessful interaction. To measure the percentage of unsuccessful cases, the *Percentage of Unsuccessful Actions* is used as the first performance metric in our study. For those unsuccessful cases, one would like to know the degree of incompleteness. Therefore, the *Average Percentage of completion* is used as our second performance metric. As an example, if the user wishes to fast forward the play point for 20 seconds, but is forced to resume the normal play after only 15 seconds, the percentage of completion is 15/20 or 75%. Finally, when the user resumes a normal play after a successful or unsuccessful interactive action, the beginning play point might be

slightly different from the desired destination point of the interactive action. To capture these displacements in the performance comparisons, the *Average Displacement* is used as the third metric. A good interaction technique should have a low percentage of unsuccessful interactive actions, high percentage of VCR action completion, and low average displacement.

3.4.3 Simulation results

Without loss of generality, the CCA parameter *c* is set to 3 in this study. That is, all clients use three loaders to load the regular segments. Two more loaders to load the compressed segments are utilized. Simulations are conducted on a video of two hours. The simulation results are presented in the following subsections. A first section shows that BIT technique outperforms the Active Buffer Management technique for longer interactions. Another section examines the effect of the client buffer size on performance, and demonstrates that good *average displacement* can be achieved. Finally, a last section illustrates the effect of changing the compression factor on the *percentage of unsuccessful actions* and the *average percentage of completion*.

3.4.3.1 Effect of the duration ratio

The compression factor is set to 4 during this simulation. The regular client buffer is 5 minutes; and the total buffer space is 15 minutes. The server uses 40 channels, from which 32 are used for the regular video, and 8 for the compressed version of the same video (i.e. $K_r=32$, $K_i=8$). This configuration shows 10 segments of unequal size and 22 segments of equal size. The size of the smallest segment is 2.84 seconds. Hence, the average access latency is 1.42 seconds.

The parameters for the user behavior are as follows. An equal probability is set for a normal play and for all the interaction types combined (i.e., $P_p = 0.5$, $P_i = 0.5$, $P_{pause} = P_{ff} = P_{fr} = P_{jf} = P_{jb} = 0.1$). The mean duration of a play operation m_p is 100 seconds. The duration ratio *dr* is varied from 0.5 to 3.5. The values selected for this ratio are high for two reasons: (1) Jump actions incur a big change in the position of the play point causing a larger *dr*; (2) this study is intended to test these techniques during periods of intense interactive activities such as searching for a desired video segment. In other words, the user interaction behavior is modeled, not video watching behavior from which the objective is to compare BIT and ABM in supporting interactions.

Figure 3.8 shows the effect of changing the duration ratio on the *percentage of unsuccessful actions* and the *average percentage of completion*. For $dr = 0.5$, 20% of the interaction actions are denied under ABM, compared to only 11% under BIT. One can notice that BIT is much less sensitive to changing the duration ratio. When $dr = 3.5$, BIT outperforms ABM by a factor of 48% in terms of *percentage of unsuccessful actions,* and 31% in terms of *average percentage of completion.* The poorer performance of ABM is partially due to a very fragmented buffer. The primary reason, however, is due to the fact that caching a compressed version is more effective in supporting longer-duration interactive actions.

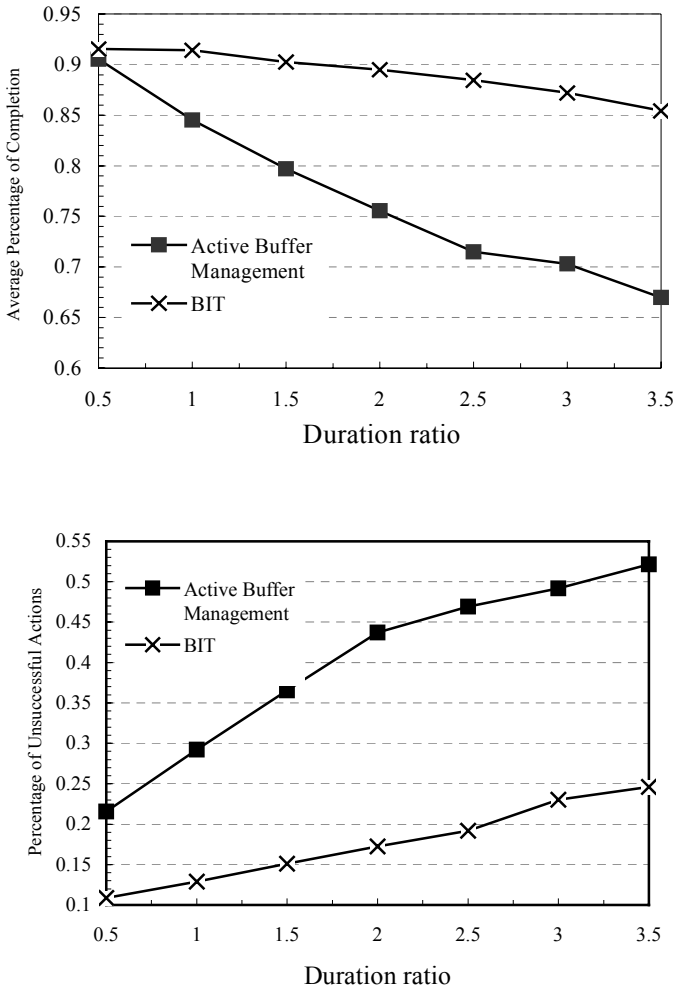


Figure 3. 8 Effect of changing the duration ratio

3.4.3.2 Effect of the buffer size

In this study, the client buffer size is varied from 3 minutes to 21 minutes. The same parameters are kept for the user behavior as in the last experiment except for the duration ratio that is set to

1.0 and 1.5 for two different simulation runs. Since the size of the regular playback buffer in the BIT technique is a third of the total buffer size (the other two thirds is used to hold the compressed segments), at least 120 regular channels are used to broadcast the entire video if the regular buffer size is 1 minute. However, if the regular buffer size is 7 minutes only 18 channels are needed. 132 regular channels are used, 14 of them broadcast segments of unequal size and 118 channels broadcast segments of equal size. Since the compression factor is 4, the total number of interactive channels is 33 (with $K_r = 132$ and $K_i = 33$). Figure 3.9 shows the effect of changing the buffer size for a duration ratio of 1.0 and 1.5. One can see from the figure that BIT does not require nearly as much buffer space as ABM to achieve an *average percentage of completion* of more than 80%. When the buffer size is small (i.e., 1 minute), BIT doubles the performance of ABM in terms of number of unsuccessful actions. As the buffer size increases, both techniques show better performance; but BIT continues to perform significantly better. Again, the advantage of BIT can be attributed to caching the compressed version in the interactive buffer.

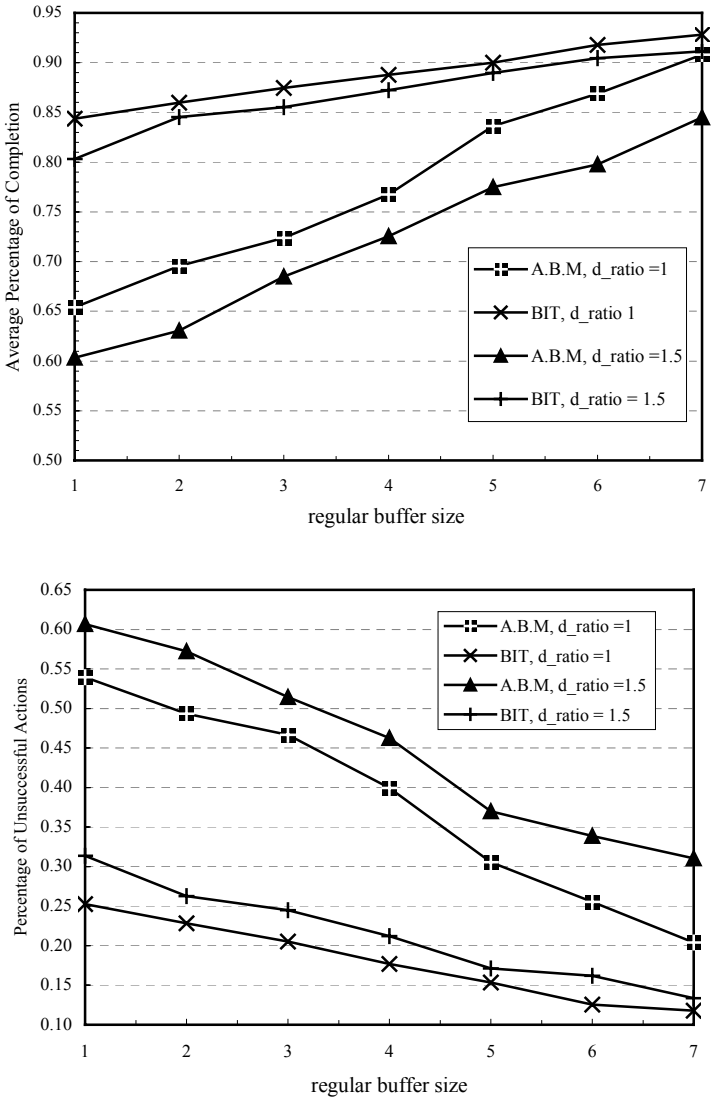


Figure 3. 9 Effect of changing the buffer size

3.4.3.3 Effect of server bandwidth to the total average displacement

One can recall that a displacement occurs when the resumption from an interactive action starts the normal playback at a play point different from the desired destination point of the interactive

action. In this experiment, the effect of server bandwidth on the *average displacement* is investigated. The same user behavior parameters as in the previous experiments are used, with the duration ratio set at 1.5. The server bandwidth is varied between 30 channels and 165 channels. The sizes of the W-segment corresponding to different server bandwidths are given in Table 3.4. The table also shows the number of regular channel (*Kr*) and the number of interactive channels (*Ki*). In all of these configurations, the smallest segment is less than 10 seconds. The plot for the average displacement under various server bandwidth (in terms of number of regular channels) is presented in Figure 3.10. One can observe that the average displacements of BIT and ABM are comparable. Their difference decreases with the increases in the server bandwidth. This is due to the fact that the size of the *W-segment* is smaller for a larger server bandwidth. When the *W-segment* is very small, the average displacement becomes small regardless of the interaction technique.

Table 3. 4 W-segments and their respective *Kr* and *Ki*

W-segments (mn)	1	2	3	4	5	6	7
K_r	132	72	48	36	32	28	24
K_i	33	18	12	9	8	7	6

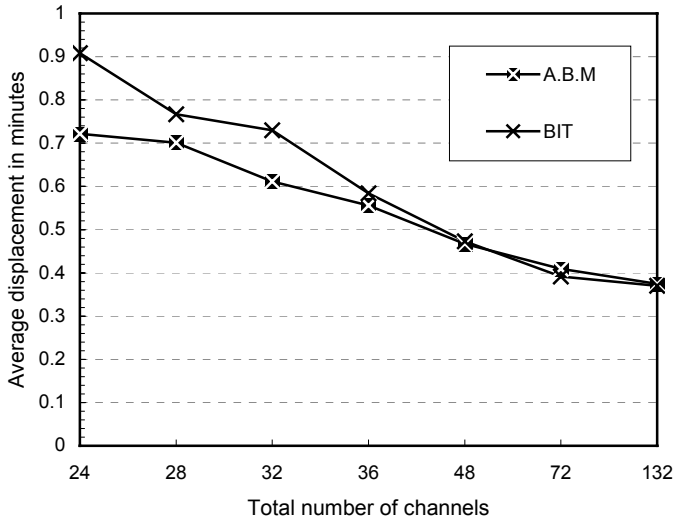


Figure 3. 10 Effect of W-segment size on the average displacement

3.4.3.4 The Effect of Changing the compression factor f:

In this experiment, the effect of changing the compression factor *f* is studied. The size of the regular playback buffer is set at 5 minutes and the total number of regular channels K_r to 48. The mean duration of a play is fixed to half the size of the total buffer space and the duration ratio to 1.5. Table 3.5 gives the total number of interactive channels along with their respective compression factor. The simulation results are shown in Figure 3.11. One can observe that by increasing the compression factor the performance of BIT is increased. However, an excessively high compression factor will result in a lower resolution of frames being rendered during a continuous interactive action.

Table 3. 5 Compression factor for a Total of 48 regular channels

f	2	4	6	8	12
K_r,K_i	48,24	48,12	48,8	48,6	48,4

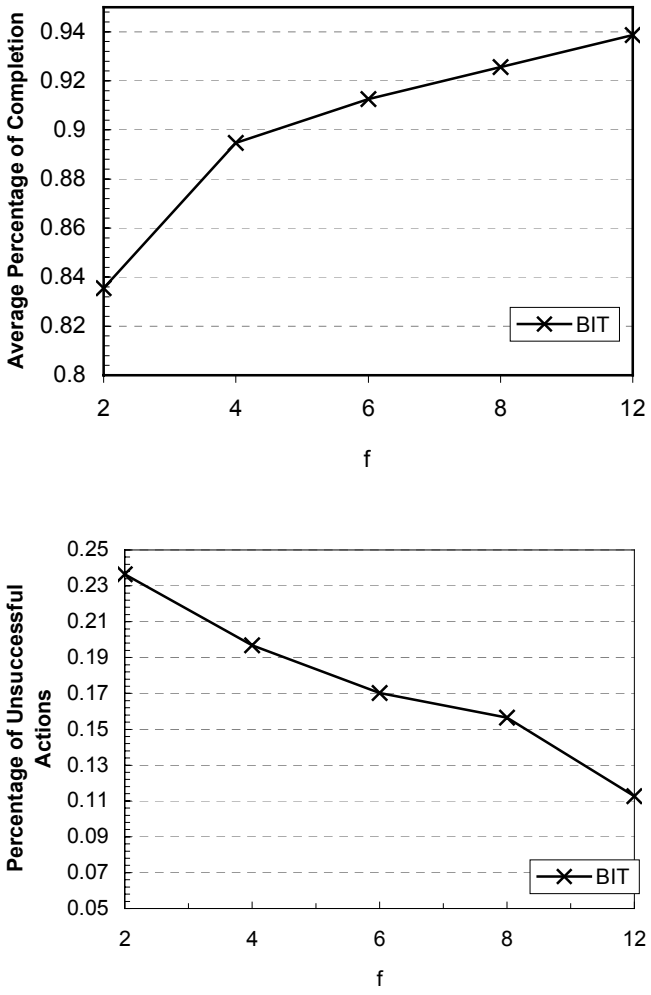


Figure 3. 11 Effect of changing the compression factor

3.5 Summary

Although multicast delivery scales well to a very large number of users, the problem of providing interactive functions without compromising the multicast paradigm remains. This chapter showed that existing techniques solved only partially the problem of VCR interactivity in a broadcast framework. There have been two approaches to date. The first approach relies on guard channels to deliver the emergency streams to provide VCR functions. Since each emergency stream is dedicated for one client, this approach is limited to small-scale deployment. The second approach attempts to maintain the play point at the middle of the video segment currently cached in the prefetch buffer. Although this strategy offers some advantage, it does not address the primary requirement of VCR interactivity which demands the data stream to arrive at a much greater speed than the normal playback rate.

In this chapter a new video interaction technique is introduced for the broadcast environment, that addresses the limitations of existing solutions. The *Broadcast-based Interaction Technique* (BIT) offers better interaction quality by repeatedly broadcasting the interactive (compressed) version of the video. By prefetching data from this version and caching them in the local buffer, the client is able to support longer-duration interactive actions. In terms of scalability, since the clients can share the interactive broadcasts, the bandwidth requirement of BIT is independent of the number of users the system is designed to support. This strategy is consistent with the

broadcast paradigm.

To assess the benefits of BIT, two detailed simulators are implemented to compare BIT to a recent technique called *Active Buffer Management* scheme. The simulation results indicate that BIT using the same amount of buffer space can offer significantly better performance.

CHAPTER FOUR: BANDWIDTH USER-HETEROGENEOUS BROADCAST

A number of periodic broadcast techniques have been proposed for the cost-effective implementation of VOD systems. Most of these techniques would either try to minimize the server bandwidth, user bandwidth, user storage, user access latency to the video, or a combination of some of the aforementioned parameters. On the other hand, the implementation strategies of these broadcast schemes would necessitate a minimum bandwidth requirement for all users. Multi-resolution techniques address the heterogeneity problem by sacrificing user video quality. In this chapter, a different approach that does not possess this disadvantage is considered. Using an incremental channel design at the server side, and a specific broadcast schedule, users can choose among a range of bandwidths to use to download the video at the cost of their access latency and not to the video quality. Section 1 discusses related work relative to the bandwidth heterogeneity problem. In section 2, a novel technique, called *BroadCatch* [THD04] is introduced. Section 3 proves the correctness of the proposed solution and provides mathematical analysis to demonstrate its heterogeneous behavior. Section 4 shows the convenience of using BroadCatch for live video streaming and for seamless channel transitions. In section 5, a performance study illustrates the benefit of using BroadCatch over a recent technique that provides bandwidth user heterogeneity and a recent scheme that minimizes user bandwidth without offering heterogeneity features. Finally, Section 6 summarizes the chapter.

4.1 Bandwidth User-Heterogeneity And Related Work

Essentially all existing periodic broadcast techniques require the VOD service providers to tailor their system design to fit the capability of a specific class of users. This approach has two disadvantages. First, users with lower bandwidth will not be able to watch the video as continuity cannot be guaranteed; and second users having more bandwidth at their disposal will not be able to benefit from their surplus.

To address the above drawbacks, one can consider encoding the video stream into a decomposition of layers [LW95, FCG96]. Depending on the bandwidth available at the end user, more or less layers are received, resulting in a variation in the display quality. This strategy has been proposed for multicast in heterogeneous environments [MJV96, CFMB98]. One can adapt the same idea for periodic broadcast to provide adaptability to different receiving bandwidths. This simple solution, however, would present some drawbacks for a high quality VOD system. Firstly, a user with lesser bandwidth is enforced to sacrifice video quality; secondly, it is very demanding on user bandwidth to obtain a satisfying image quality; and finally, it is difficult to implement. Some recent techniques have addressed the latter problem by coding "enhancement" layers from the base layer instead of calculating them independently. The *NAIVE* scheme [BGM99] reconstructs an image from any number of independent samples received for this specific image, making this approach particularly robust and adaptive to poor-performance connections. Another advantage of *NAIVE* is not to require a fixed bandwidth at the receiving end. Nevertheless, the loss of display quality remains significant between different users having distinct reception bandwidths.

The work in [SGT01] proposes several algorithms that deal with the unconstrained and constrained user bandwidth and buffer requirements in the periodic broadcast environment at frame level. This work offers a heuristic-lazy algorithm that minimizes the access latency given that all users are constrained by a minimum bandwidth requirement. In such a setting, a broadcast schedule is found; however this work does not address the heterogeneity problem of the periodic broadcast paradigm. That is, given only one video broadcast schedule, users with diverse communication capabilities need to be able to download the video according to their own reception schedule. This capability is first provided in the *HeRo* approach [BHO03]. This scheme employs only one broadcast schedule; and depending on the communication capability of the receiver, it needs to wait until an appropriate time slot to start the video service. The *HeRo* technique is an extension of the *Client Centric Approach* [HCS98, HCS01]. The CCA technique was intended to reduce the user access latency using the maximum allowed user bandwidth. By adding channels that periodically broadcast the last segments of the CCA approach but shifted by half of their size, the *HeRo* technique reduces the average required user bandwidth at each time slot. Therefore, in [BHO03] two types of channel are proposed; K regular channels similar to the *CCA* channels, with the addition of N_r shifted channels which periodically broadcast only the last segments but shifted with an offset equal to half of their sizes.

This early solution has a few drawbacks. First, it is extended from the Client-Centric Approach [HCS98, HCS01]. Since CCA was designed to take advantage of client bandwidth in a homogenous environment, HeRO can achieve excellent service latency for the most capable clients. The service delay experienced by less capable clients, however, leaves room for

improvement. Second, to decrease the access latency, the HeRo approach would have to re-fragment all available segments and re-schedule them at different times, which makes this technique inefficient for high adaptation.

4.2 Broadcatch Technique

A periodic broadcast technique consists of a fragmentation design of the video data, a broadcast schedule, a reception schedule, and a receiver playback strategy at the receiver end. In this section, first the channel design is described, which involves the broadcast schedule, then the reception and playback of the video. Notations used in this section is given in table 4.1.

Table 4. 1 Notations

b	Playback rate of the video in Mbps
K	Total number of server channels
d	Worst access latency to the video in seconds
L	Length of the video in seconds

4.2.1 Channel Design

Without loss of generality, the broadcast of a constant bit rate (CBR) encoded video is

considered. To deliver another video, the server allocates enough resources and uses the same broadcast strategy. Hence, the server broadcasting design can be seen as a collection of virtual servers, each for a specific video.

In *BroadCatch*, each channel repeatedly broadcasts a leading portion of the given video at the playback rate *b*. The size of the portion broadcast on channel *i* is denoted by *Period(i),* and is computed as follows:

$$Period(i) = \begin{cases} L & if \quad i = 1 \\ L & if \quad i = 2 \\ Period(i-1)/2 & otherwise \end{cases} \qquad (4.1)$$

Thus, the first two channels repeatedly broadcast the video in its entirety, and are referred to as the *base channels*. The subsequent channels broadcast the video only partially, and are referred to as *catching channels*. They are used to help users arriving at different times to "catch up" with an earlier full broadcast on one of the two base channels.

Let *Offset(i)* denote the offset of the first broadcast on channel *i* relative to the beginning of the first broadcast on channel 1. That is, the broadcast on channel *i* starts *Offset(i)* time units after the broadcast on the first channel has begun. The broadcast schedule of the *BroadCatch* technique is defined as follows:

$$Offset(i) = \begin{cases} 0 & if \quad i = 1 \\ L/2 & if \quad i = 2 \\ Offset(i-1)/2 & otherwise \end{cases} \qquad (4.2)$$

Figure 4.1 shows an example of the BroadCatch channel design technique using 5 channels

(K=5). The first two base channels periodically broadcast the entire video of size L. The catching channels periodically broadcast the leading portions of the video with sizes defined by Equation 4.1, and offsets by Equation 4.2.

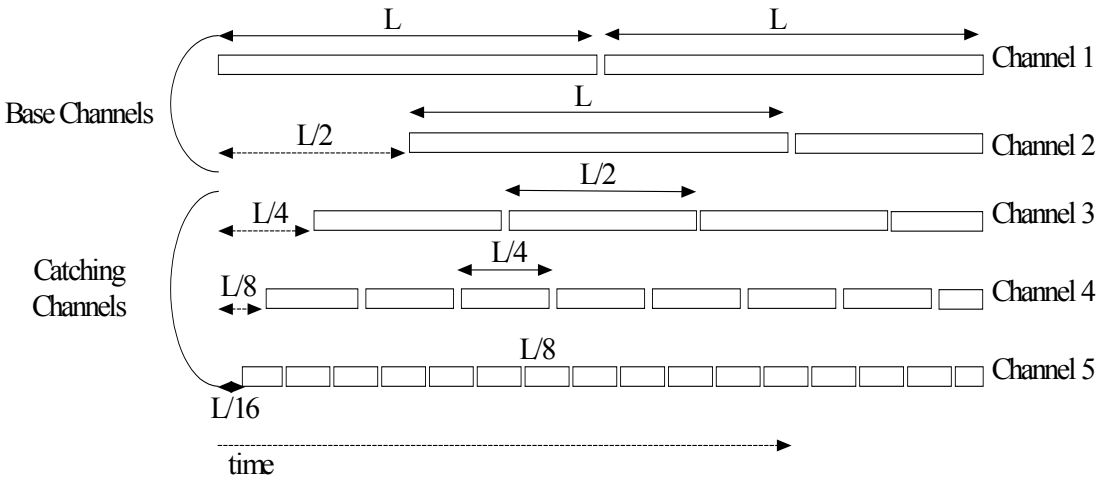


Figure 4. 1 BroadCatch Channel Design

4.2.2 Reception and Play Back of the Video

With the BroadCatch technique, users with a bandwidth capability in the range of b to $(K-2)\cdot b$ *Mbps* can download the video from the same periodic broadcast.

The video file is divided into a large number of data segments; each has the size half of the video portion broadcast on the previous channel. Let S_1 denote the first segment, and $|S_1|$ its size. Figure 4.2 depicts the broadcast schedule in terms of such segments. The numbers inside the

rectangles represent indices of segments. The numbers on top of *Channel 1* correspond to the required bandwidth in terms of the play back rate *b* of the video for a user arriving just before the broadcast of that segment, and the different shades indicate different reception schedules. One can notice that the last channel does not broadcast S_2 as it is sufficient to guarantee a jitter-free playback at the receiver end.

The total number of segments depends on the number of channels the server is allocating to that specific video. In general, with *K* channels allocated to the video, it has:

$$TotalSeg(K) = 2^{K-1} \text{ segments} \qquad (4.3)$$

In this chapter, the reception schedule of the BroadCatch technique is expressed using segments that are referred to as S_1, S_2,…., $S_{TotalSeg(K)}$, where the server allocates *K* channels of rate *b* to broadcast the video.

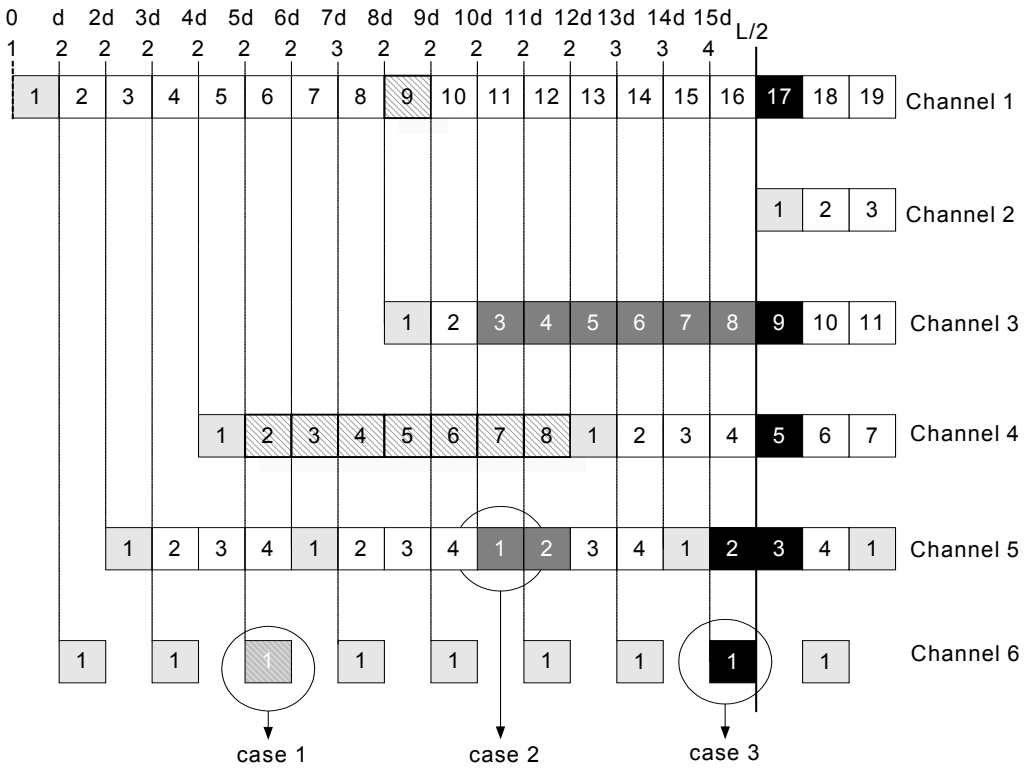


Figure 4. 2 BroadCatch Reception Strategy

S_1 is broadcast on every channel. A client arriving at some time always downloads the earliest S_1 segment, tunes to another channel to catch up and downloads S_2, S_3, S_4, and so forth, until it reaches a base channel to download the rest of the video. Because this technique uses periodic broadcast to deliver the video, and clients catch up to several channels to download it, thus its name *BroadCatch*. If a client starts its service at channel *i to get the first segment*, it downloads until the last segment, say segment *x*, on that channel. At the same time, it looks for the earliest appearance of segment $x+1$ in one of the upper channel j ($j < i$) to catch up to it. Once it catches up to channel j, it downloads segment $x+1$ and the subsequent segments until the last segment on

that channel. In the meantime, it repeats the process to look for the next upper channel to catch up until the entire video has been downloaded. This procedure is illustrated in Figure 4.2.

In case 1, the client starts the service at channel 6. It needs to download S_1 on channel *6*, and S_2 on channel *4* simultaneously (as shown in Figure 4.2 by the rectangles with similar shades). As the client downloads the remaining segments on channel *4*, it identifies that the next occurrence of segment S_9 will happen on channel *1*. Once the download of S_4 at channel 4 is complete, the client begins to download segment S_5 from channel 4 and S_9 from channel *1* at the same time. The client downloads the rest of the video from channel *1* since the latter is a base channel. In this case, the user needs to have $2 \cdot b$ communication bandwidth in order to receive the service. Because starting to download from channel *6*, case *3* is similar to case *1*. From Figure 4.2 case *3*, we can see that the client needs $4 \cdot b$ communication bandwidth since it must download segments S_3, S_5, S_9, and S_{17} at the same time.

Clients starting their download from channel *5* as opposed to channel *6*, such as in case 2, download different segment indices. In case 2, a client disregard totally channel 6 as it first encountered the beginning of the video (segment S_1) at channel 5. Therefore, the number of channels used by this client for the entire download is 5 channels. If one recalls Equation 4.3, the number of segments that the client in this case observes can be grouped into 16 segments (from S_1 to S_{16}) as opposed to 32 segments such as in cases 1 and 3. In fact, segments S_1 and S_2 depicted in the figure are relative to using 6 channels. Using only 5 channels segments S_1 and S_2 will constitute only one segment, that a client arriving at a time such as in case 2 will refer to as S_1. If the first segment using K channels is denoted as $S_{1,K}$, in case 2, $S_{1,6}$ and $S_{2,6}$ of Figure 2

constitutes $S_{1,5}$. $S_{3,6}$ and $S_{4,6}$ make up $S_{2,5}$ and so forth. Therefore, in case 2, the client downloads $S_{1,5}$ from channel 5, at the same time it downloads $S_{2,5}$ ($S_{3,6}$ and $S_{4,6}$ in Figure 4.2) from channel 3. Because channel 3 does not broadcast $S_{9,5}$ (which is $S_{17,6}$ and $S_{18,6}$ in Figure 4.2), the client catches up to channel 1 to download it as well as the rest of the video. At all times, the client does not listen to more than two channels making the bandwidth requirement for this case $2 \cdot b$ communication bandwidth.

When a client tunes to receive the video, the server informs it of the parameter K, and the beginning time of the broadcast of the entire video at the base *channel 1*. Using theses values, the client can exactly construct an image of all segments broadcast as the example in Figure 4.2. In the rest of this subsection, a formal algorithm is presented that describes the detailed reception of the video for a user arriving at time t, having a bandwidth C_b and a total number of loaders C_L, where $C_b = C_L \cdot b$.

Figure 4.3 describes the client reception algorithm. The client first tunes to the earliest appearance of segment $S_{1,K}$ at some specific channel say i. From that point on, the client disregards all channels having indices greater than i, case 2 in Figure 4.2 shows an example where the client disregards channel 6. Second, the client constructs schedule '***S***' specific to its arrival and compute the required number of loaders (***ReqLoaders***) for that specific schedule. If the available number of loaders of the client C_L is greater than the required number of loaders *ReqLoaders* specific to schedule S, then the client has found a feasible schedule and therefore follows it. Otherwise, the schedule S is not feasible. The client then waits for the next appearance of segment $S_{1,K}$ and repeats the process to find a schedule that fits its bandwidth capability.

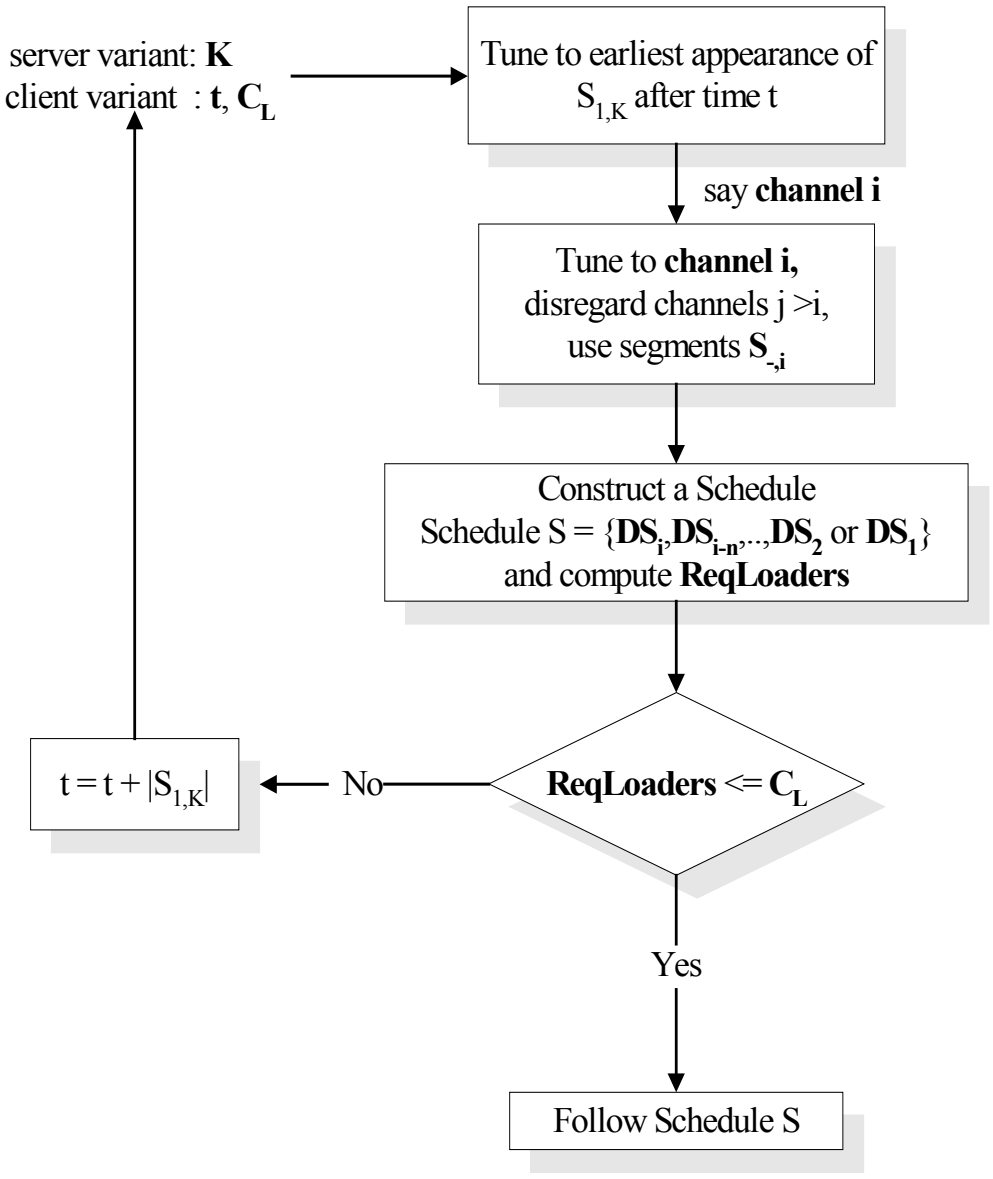
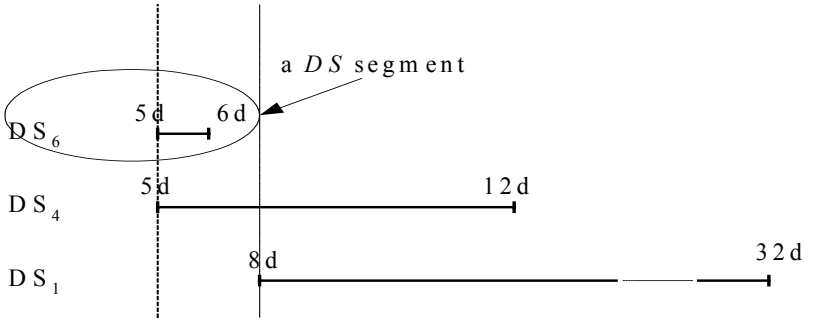


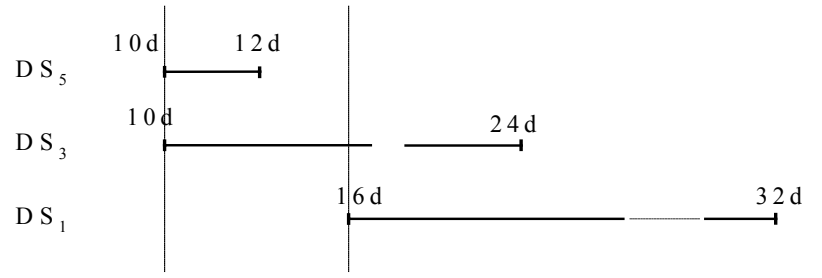
Figure 4. 3 Client Reception Algorithm

Schedule S is composed of several downloadable segments $\boldsymbol{DS_m}$. A downloadable segment DS_m is the concatenation of all segments downloaded from channel m. Because the client tunes to the first appearance of segment $S_{1,K}$ at channel i, the first downloadable segment of schedule S is always DS_i. The client downloads segment $S_{1,i}$ from channel i and at the same time look for the earliest appearance of segment $S_{2,i}$ in one of the upper channel j ($j<i$). Then it downloads all segments from channel j say $S_{2,i}$, $S_{3,i}$, ..until, $S_{x,i}$, forming a downloadable segment DS_j, at the same time looks for the earliest segment $S_{x+1,i}$ in one of the upper channel k ($k<j$) until it reaches a base channel, channel 1 or channel 2 forming either DS_1 or DS_2, where it downloads the rest of the video. Figure 4.4 shows all downloadable segments for the cases presented in Figure 4.2.

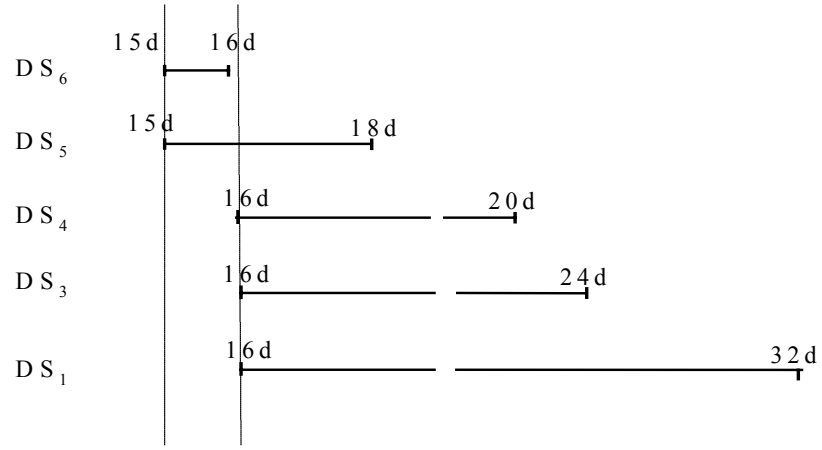
To compute the number of loaders required by a client (*ReqLoaders)* for a specific schedule S, the client computes the number of intersecting downloadable segments as shown in Figure 4.4. For example, case *1* and case *2* requires a user bandwidth of $2 \cdot b$ because DS_6 intersects DS_4 at time $5d$ and DS_4 intersects DS_1 at time 8d in case *1*. In case *2*, DS_5 intersects DS_3 and at another time DS_3 intersects DS_1. Case *3* presents an intersection with *4* segments at time $16d$ therefore the client bandwidth requirement is $4 \cdot b$. The number given at the beginning of each time slots in Figure 4.2 represents the required number of loaders for that specific slot and hence the bandwidth requirement to download the video. One can observe that all clients arriving during the first half portion of the video catch up to the first base channel while clients arriving during the second half catch up to the second base channel.



case 1. Schedule S = {DS_6,DS_4,DS_1}



case 2. Schedule S = {DS_5,DS_3,DS_1}



case 3. Schedule S = {DS_6,DS_5,DS_4,DS_3,DS_1}

Figure 4. 4 Downloadable Segments for Cases 1,2,and 3

4.3 Broadcatch Analysis

In this section analyzes the BroadCatch technique. As described earlier, for each time slot different requirements in terms of client bandwidth and buffer space are needed. First the client access latency to the video is investigated. Then a theoretical upper bound to the client bandwidth requirement at each time slot is given. Finally, the client buffer space requirement to download the video is examined.

4.3.1 Access Latency

To download and watch the video, clients try to access the first occurrence of segment S_1 relative to their arrival (in the rest of the paper and when no confusion occurs, S_1 is used instead of $S_{1,K}$ to represent the smallest broadcast segment). Due to the nature of the broadcast channel design, the worst access time to the first segment that is denoted by d is given by the following equation:

$$d = L / 2^{K-1} \qquad (4.4)$$

If a client possesses enough bandwidth to download the video at any given time slot, then Equation *4.4* represents the worst waiting time to access the video. Otherwise, the client is obliged to wait for the next appearance of S_1. Next section discusses this situation. If such case happens the client will probably hold enough bandwidth to download the video at the next time slot.

4.3.2 Client Bandwidth Requirement

In this subsection, a closed form to the *worst client bandwidth requirement* is presented at each time slot. A time slot is defined as the time of the appearance of segment S_1, and the worst client bandwidth requirement is described as enough bandwidth to download the video at a specific time slot.

The worst client bandwidth requirement at each time slot is generated incrementally and by adding a new broadcast channel at each step. Figure 4.5 describes this process. First and due to the symmetry of the BroadCatch design channel, and without loss of generality, one can focus only on the first half of the video. Therefore, only clients that catch up to the first base channel are considered. The case where clients catch up to the second base channel is similar.

If only one channel is used to broadcast the video (i.e. the first base channel), then there is only one time slot to catch the video represented at the beginning of the broadcast. If there are two channels (the first base channel and a catching channel, or channel *3*), then another time slot is added, in that case, the video file is decomposed into four segments (refer to Equation 4.3). A user arriving just before that time slot downloads the first segment from channel *3* and catches up to the first base channel. When the fourth channel is added two more time slots are added, the fifth four more, and so forth.

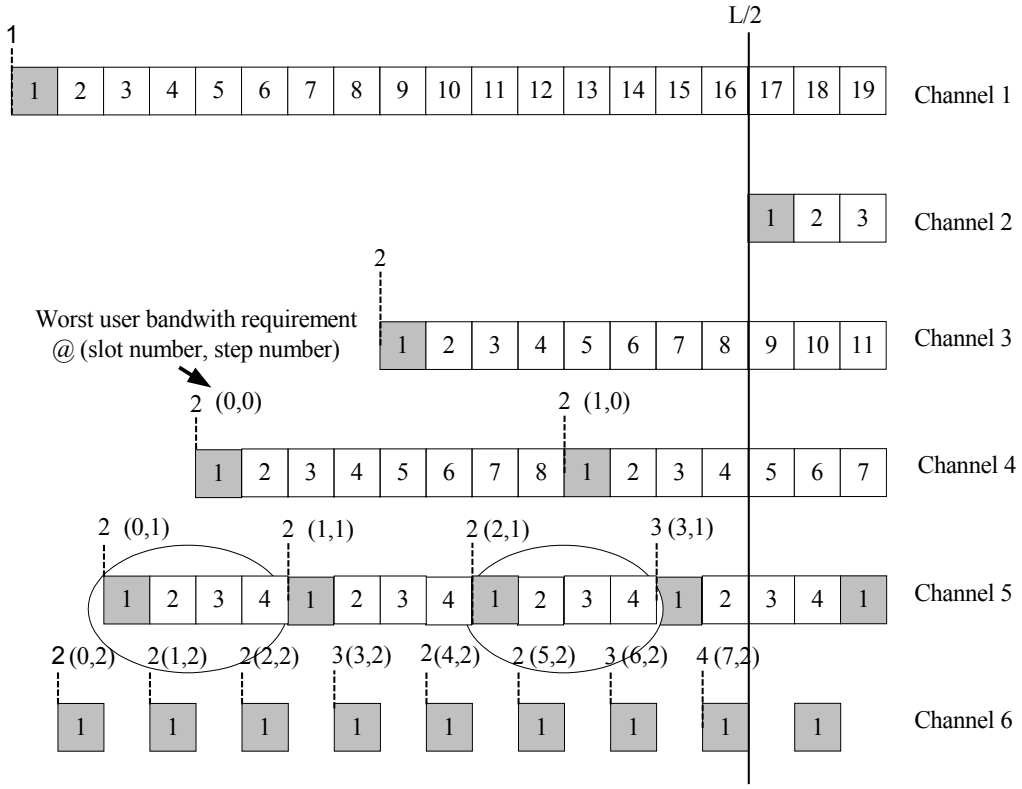


Figure 4. 5 Worst user required bandwidth for *K =6.*

In this construction, the initial step is chosen to be the slots defined in the fourth channel, from Figure *5 step 0* represents slots defined by channel *4*, *step 1* represents slots defined by *channel 5* and so forth. The total number of slots defined at each *step i* is 2^{i+1}. The *worst user bandwidth requirement* at time slot number *k* at step *n* is denoted by ***wub(k,n)*** , $\boldsymbol{0 \leq k < 2^{n+1}}$

Figure 4.5 gives the values of *wub(k,0)*, *wub(k,1)*, and *wub(k,2)* relative to the time slots of channels *4*,*5*, and *6*. Given *wub(-,n)*s for a specific *step n*, one can construct *wub(-,n+1)* by considering the following. First, *even* and *odd* appearances of segments in the n^{th} *step* (even

segments are encircled in Figure 4.*5* at channel *5*) are taken into account. Figure 4.*6* gives all the possible cases encountered. If a client arriving just before the broadcast of an even appearance of segment S_1 at *step n* can use *X* loaders to download the video, and arriving just before the next appearance of S_1 (an odd appearance of S_1) can use *Y* loaders to download the video, then one can generate the corresponding four cases of the appearances of S_1 broadcast by the next step *n+1* by:

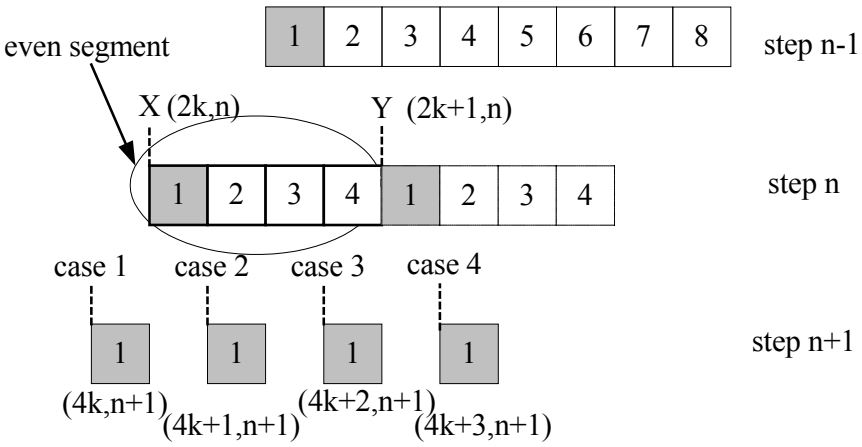


Figure 4. 6 Possible cases for the *wub(-,n+1)*

- ***Case 1***: because the *wub(-,n+1)* are generated by construction at each step, a client arriving just before the appearance of S_1 tunes to channel *j* such that $j<n-1$ to get segment S_2 as early as possible, such segment always exists. Therefore, the number of loaders used in this case is at most *X*.
- ***Case 2***: A client arriving just before the appearance of segment S_1 in this case tunes to

channel $n+1$ to get S_1, and channel n to get S_2 at the same time. The client downloads S_3 and S_4. However, S_5 is not present at channel $n-2$, therefore the client tune to channel j where $j< n-2$ to get S_5. In such case the client utilizes at most X loaders.

- ***Case 3***: A client in this case downloads S_1 from channel $n+1$ and catches up to channel $n-1$ to get S_2. However and as we shall see in the next case, only clients that catches up to an odd segment at channel n can catch up to channel $n-1$. The client in that case utilizes at most Y loaders.
- ***Case 4***: A client in this case downloads S_1 from channel $n+1$, catches up to channel n to download S_2, S_3 and S_4, and while downloading S_3 the client catches up to channel $n-1$ to get S_5. This process may include all catching channels until it reaches the base channel. Therefore, in this case a client arriving just before the appearance of S_1 may need to use $Y+1$ loaders. There are some cases however, where the client uses only Y loaders. Hence the client utilizes at most $Y+1$.

Using the above mentioned cases we can get $wub(i,n+1)$ for $0\leq i <2^{n+2}$ by the following:

$$\begin{cases} wub(4k, n+1) = wub(2k, n) \\ wub(4k+1, n+1) = wub(2k, n) \\ wub(4k+2, n+1) = wub(2k+1, n) \\ wub(4k+3, n+1) = wub(2k+1, n)+1 \end{cases} \qquad (4.5)$$

$$where \quad wub(0,0) = wub(1,0) = 2;$$

This can also be written for any index k at any step n, where $0\leq k< 2^{n+1}$ by the following

equation:

$$wub(k,n) = wub(\lfloor k/2 \rfloor, n-1) + \left\lfloor \frac{k - 4 \cdot \lfloor k/4 \rfloor}{3} \right\rfloor \qquad (4.6)$$
$$and \quad wub(0,0) = wub(1,0) = 2;$$

A closed form of *wub(k,n)* is given in Appendix A. We note that the form given in Appendix A comprises the sum of two parts. Because the situation that is analyzed is only when a client arrive after the first broadcast of S_1 at the base channel 1, the client needs at least two loaders, which constitutes the first term. The second term represents the case when the client uses intermediate channels to catch up to the base channel.

The rest of this section proves using the worst bandwidth requirement *wub(k,n)* that the heterogeneous aspect of the BroadCatch technique is scalable relatively to the server bandwidth allocated to the specific video. First, some definitions are given to evaluate the heterogeneity characteristics and then the scalability feature is proved.

<u>*Definition 1:*</u> S_n refers to the sum of the generated worst bandwidth requirement at *step n*, that is;

$$s_n = \sum_{k=0}^{2^{n+1}-1} wub(k,n) \qquad (4.7)$$

From Figure *4.5*, $s_0=4$, $s_1=9$, and $s_2=20$. Because we are concentrating on the first half of the video (similar behavior is observed during the second half where the same values of *wub(k,n)* are repeated), the average worst user bandwidth over the first half of the video is defined as follows:

<u>*Definition 2:*</u> *awub(n)* is the average worst user bandwidth at *step n* given by:

$$awub(n) = \frac{Sum\ of\ \ worst\ user\ bandwidths\ at\ all\ steps}{Total\ number\ of\ \ time\ slots} \qquad (4.8)$$

awub(n) reflects the average worst bandwidth requirement of a client arriving at any time during the first half of the video and it is specified by Equation *4.9*.

$$awub(n) = \frac{3 + \sum_{k=0}^{n} s_k}{2^{n+2}} \qquad (4.9)$$

In Equation 4.9, the denominator expresses the total number of appearances of segment S_1, or the total number of time slots. The numerator expresses the sum of the worst user bandwidths at all steps, the constant *3* characterizes the required user bandwidth when using the first two channels (channel *1* and *3*, refer to Figure *4.5*), and the second term is the sum of the worst user bandwidth requirement until *step n*. Using definition *1* and *2* we state the scalability proposition as follows.

<u>*Heterogeneous Scalability Proposition*</u>: Given *K* number of server channels periodically broadcasting at a rate of *b Mbps*. To benefit from a newly added server channel, a client would have to add to its bandwidth on the average at most *b/4 Mbps.* More formally it is to say:

$$\forall\ step\ n \quad awub(n+1) - awub(n) = 1/4 \qquad (4.10)$$

The Heterogeneous Scalability Proposition's proof is given in Appendix B.

The Heterogeneous scalability proposition expresses in terms of the video rate *b Mbps*, and on the average over all time slots, the maximum amount of bandwidth that a client has to add to benefit from a newly added server channel. Indeed, when an extra server channel is added the access latency to the video is reduced by half (refer to Equation 4.4) as the number of time slots

is doubled. A client can still utilize the same amount of bandwidth as before increasing the server bandwidth, and can still access the video with the same access latency. However, in the average case, and to access the video with the new access latency, the user would have to add some extra bandwidth. The heterogeneous scalability proposition states that on the newly added time slots, the worst required user bandwidth (*wub(-,-)*) of one of four time slots requires an extra *b Mbps* user bandwidth.

4.3.3 User Required Buffer Space

Given the situation where a client arrives before the first half of the video and catches up to the first channel (refer to Figure 4.2), the client buffer space requirement is the amount of video data it has missed. If we denote by t_d the time of occurrence of S_1 after the client arrival, then t_d can be written as:

$$t_d = n \cdot d \quad where \;\; 0 \le n < 2^{K-2} \tag{4.11}$$

d is given by Equation 4.4 and *n* a positive integer.

Due to the nature of the BroadCatch technique, the user buffer space requirement is the total amount of video the client has missed, and is also given by equation 11. The worst buffer space requirement is given at time $t_d = (2^{k-2}-1)\cdot d$ and formulated by Equation 12.

$$worst\ buffer\ space = d \cdot (2^{K-2} - 1)$$

$$worst\ buffer\ space = \frac{L}{2}(1 - \frac{1}{2^{K-2}}) \tag{4.12}$$

It is interesting to note that the client buffer space requirement never exceeds half the size of the video. Moreover, from Equation 4.11, a client requires more buffer space as it arrives late relatively to the broadcast of the video at the first base channel.

4.4. Live Broadcatch and Seamless Channel Transitions

Since the BroadCatch technique does not require any segmentation, it is better suited for real-time video-on-demand such as news on demand. If we suppose that a video is broadcast as soon as it is being made, the BroadCatch technique allows initial clients to get the video from the base channel, or from catching channels if their requests are made after the broadcast in the first channel (refer to Figure 4.2). However, and in the case of the *HeRo* technique, even if clients have enough bandwidth to download from all channels, they have to wait an amount of time proportioned to the size of the first segment. Certainly, if a client starts downloading immediately in the *HeRo* case, then it downloads the first segment and stops since the second segment is not yet stored on disk. This problem is aggravated if only few server channels are used in which case the size of the first segment is quite large. Figure 4.7 shows a broadcast schedule for the HeRo Approach. If clients arrive before the first broadcast of the video at time T_0, then they should downloads from all four channels. However the beginning portions (shaded area) of segments 2,3, and 4 are not yet available on disk. The darker segments represent a feasible downloading schedule for the HeRO approach for early clients arriving before T_0.

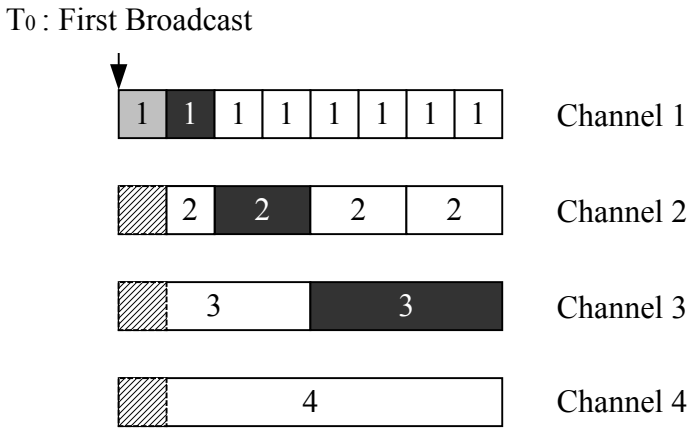


Figure 4. 7 Live Broadcast under HeRO

Most, if not all periodic broadcast techniques are concerned about the channel transition problem. A channel transition problem may arise in the following circumstances:

- The access delay to a video must be shortened to satisfy a large number of clients. If such video is no longer popular, part of the assigned channels may be released for other videos.
- If a network bottleneck occurs, the server must decrease the bandwidth to conform to this limitation.

A periodic broadcast technique offers a seamless channel transition scheme if the following is satisfied:

- ✓ Any proceeding service should not be interrupted during the channel transition.
- ✓ A maximum service delay should be restricted by an acceptable value during the channel transition.

A seamless channel transition scheme for Fast-Broadcasting [JT98] was proposed in [TYHLS01]. In this work a strategy of 'data padding' is used to enlarge the video until a suitable size before performing the transition process. This process causes considerable overheads, such as bandwidth waste from added data to the video that is also broadcast, longer service delay and larger client buffers.

Channel transitions in the BroadCatch technique are effortlessly seamless. Because no fragmentation of the video is performed, adding a new channel or releasing an old one is straightforward. Under the BroadCatch scheme, a channel is added without perturbing the delivery of other channels. The new channel will periodically broadcast only half the segment size of the previous channel (refer to Figure 4.1). If the server needs to release an old channel to accommodate either its bandwidth or network traffic, it only needs to drop the last channel. Again, this process is achieved without interrupting other channels.

4.5 Performance Evaluation

In this section, the performance of the *BroadCatch* technique is evaluated. The evaluation metrics are described in the first section. Then the average user bandwidth requirement is compared against the average worst bandwidth requirement given by Equation *4.8* in section 3.2. Next, the BroadCatch technique is compared to the *HeRo* approach [BHO03]. BroadCatch is also compared to the most performing broadcast strategy that does not provide user bandwidth heterogeneity namely the Generalized Fibonacci Broadcast (GFB) [YK03]. The evaluation of

BroadCatch against GFB is performed for mainly two reasons. First, one would like to know, under the same server bandwidth BroadCatch's performance. Second, one want to show that even a generalized technique such as GFB cannot outperform BroadCatch's best case. Finally the cost of the BroadCatch technique in terms of server bandwidth relative to the optimal periodic broadcast strategy is analyzed.

4.5.1 Metrics

To evaluate the heterogeneity of the BroadCatch technique, the following parameters are used. Firstly, the ***average user bandwidth requirement*** is denoted as the sum of all bandwidth requirement at each time slot to download the video without jitter over the total number of time slots. The average worst bandwidth requirement for the *BroadCatch* technique is analyzed in section *3.2*. Secondly, given K a total number of server channels, b_i the total number of client receiving channels, and ts_i the time slot representing the arrival time for user i. The normalized distance ***dist(***ts_i***,***b_i***)*** is set to be the number of time slots the client must wait to start downloading the video. From Figure *4.2*, if we suppose client C_i has a bandwidth of *2b Mbps* ($b_i = 2$), if C_i arrives just before time *13d*, then ***dist(13d,2)*** = *3*. Indeed, at time *13d* the client needs *3b* to download the video similarly for time *14d*, and at time *15d* it needs *4b*. The client can start downloading the video only at time *16d*, therefore ***dist(13d,2)*** = *3*. The ***average worst user access latency*** is defined to be the sum of all distances for each time slot, given a user bandwidth, over the total number of time slots. In fact, if a client possesses the same bandwidth

as the server than this distance at any time slot equals one.

4.5.2 User Required Bandwidth for BroadCatch

Figure 4.8 shows the average user bandwidth requirement for the BroadCatch technique where server channels are varied between 4 and 16 channels. One can see that until channel 7, the worst case given in section 3.2 is similar to the user bandwidth required by the BroadCatch technique. Using 8 channels or more, the BroadCatch technique presents less average user bandwidth requirement than the worst-case analysis.

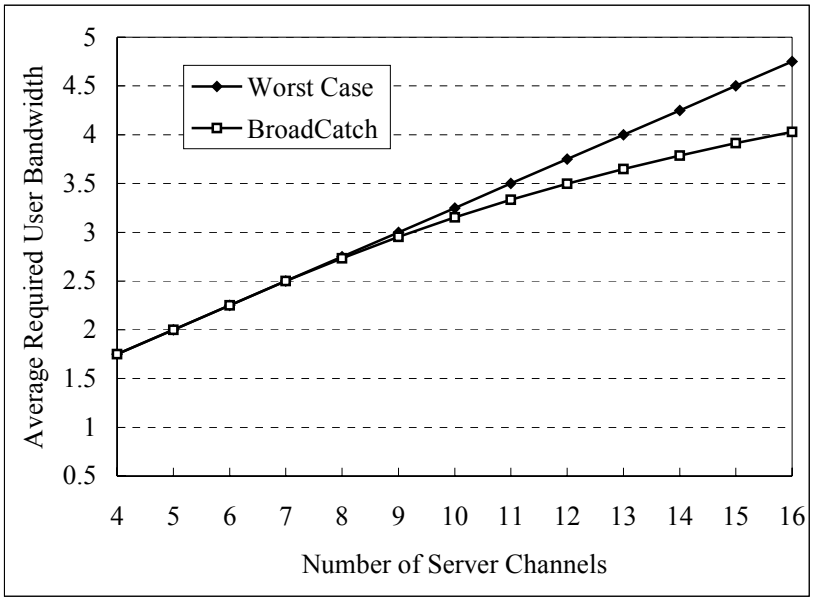


Figure 4. 8 BroadCatch average user bandwidth requirement

4.5.3 Comparing BroadCatch to the HeRo Technique

To compare the BroadCatch technique to the *HeRo* technique, the *average user bandwidth requirement* and the *average worst user access latency* are used as evaluation metrics. To provide similar user access latency as the BroadCatch technique, the *HeRo* approach is chosen to have $Nr = 1$, that is there is only one channel that periodically broadcast the last segment shifted by half of its size and the rest of server channels are similar to the CCA technique. The choice of these parameters is fair since it presents under the same server bandwidth the same worst user access latency or d as defined in section *3.1*.

Figure 4.9 shows the average user bandwidth requirement for different server bandwidth for both the *HeRo* approach and the BroadCatch technique. The server bandwidth ranges from $4b$ to $9b$ Mbps. One can see from Figure 4.9 that in the case of the *HeRo* approach, using more than *5* server channels is already worse than the BroadCatch's worst case analyzed in section *3.2*. From that point and adding more server channels, the benefit of using the BroadCatch over the *HeRo* approach in terms of the average user required bandwidth is about 10%. Using $K=9$ server channels is enough to obtain a reasonable access latency to the video. For example, given a one-hour movie the worst access latency is around 14 seconds.

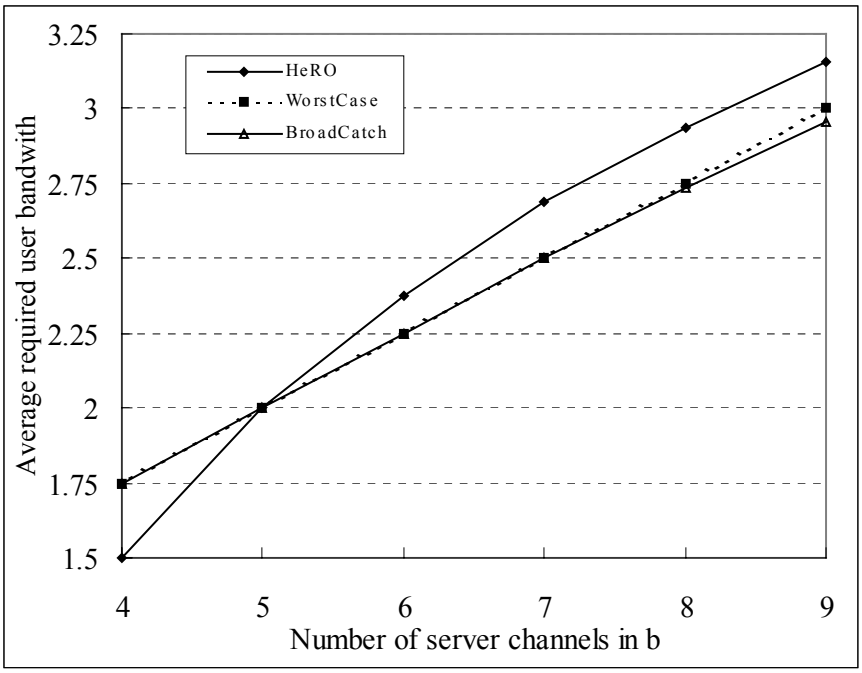


Figure 4. 9 Average required user bandwidth.

Figure 4.10 shows the worst average access time for BroadCatch and *HeRo* techniques when clients possess bandwidths that range between *2b* and *7b* for the cases where the server channel is equal to *8b* and *9b*. One can see that both techniques converge to one point, which is the worst access latency when clients have enough bandwidth to download from all channels and is given by Equation *4.4*.

However, in the case where clients have less bandwidth, one can see that the BroadCatch technique outperforms the *HeRo* approach. From Figure 4.10, we observe that using the BroadCatch, a client having only *2b* as a bandwidth has to wait on the average *27%* less time than using the *HeRo* technique. This is mainly due to the fact that in the BroadCatch technique

and at each time slot the client utilizes a minimum number of channels to catch up to the base channel. In fact, a client owning more than *2b* Mbps always downloads from the time slots that require *2b* Mbps bandwidth, and hence the shape of the curves. In the case of the *HeRo* approach, if the shifted channel is not used then the performance will be worse than shown in Figures 4.9 and 4.10.

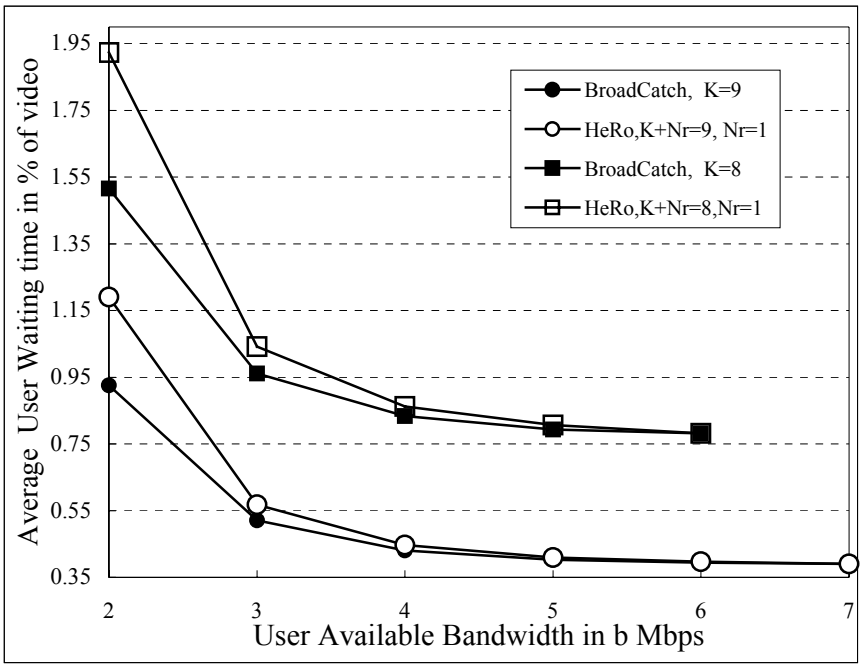


Figure 4. 10 Average worst access latency in % of the video length

4.5.4 Comparing BroadCatch to the Generalized Fibonacci Broadcast (GFB)

Fixed-Delay Pagoda Broadcasting (FDPB) [PA01] protocol and the *Generalized Fibonacci Broadcasting* (GFB) [YK03] minimizes server bandwidth. However, they are not intended for the video delivery to clients with heterogeneous bandwidths. In fact, while the *FDPB* can restrict its client to listen to only two channels simultaneously and offer better performance than the Skyscraper Broadcast in terms of server bandwidth, it cannot accommodate clients with more bandwidth capability.

The Generalized Fibonacci Broadcast (GFB) [YK03] is a generalization of the Fibonacci periodic broadcast similar to the generalization of the Skyscraper broadcast [HS97] to CCA [HCS98, HCS01] in the sense that increasing the number of client channels has the advantage of reducing its waiting time. The Generalized Fibonacci Broadcast introduces a configurable parameter $g>0$ to limit the server channel bandwidth to b/g. GFB divides the video into n segments where each segment is periodically broadcast on its specific channel of bandwidth b/g. To derive the continuity condition of a smooth play back, GFB uses segments of sizes that look like the Fibonacci sequence. The parameter K defines the total number of channels a client can tune to simultaneously. Therefore, the Generalized Fibonacci Broadcast is noted as GFB(K/g), where the parameter K/g represents the available client bandwidth, for example GFB(2/1), GFB(4/2), and GFB(6/3) use the same client bandwidth capability with a difference to the number of channels clients can tune to simultaneously. By choosing a parameter K and g, GFB outperforms recent techniques in terms of minimizing server bandwidth. But, GFB does not

provide bandwidth heterogeneity, in the sense that, if the server schedule's aim is to provide service to users with GFB(6/2) for example then all clients should possess a bandwidth capability of *3b*.

Figure 4.11 and 4.12 compare GFB(2/1), GFB(3/1), and GFB(4/2) to the best case of the BroadCatch technique. The best case of BroadCatch is where all clients have enough bandwidth to download the video as soon as they access the system and is computed using Equation 4.4. One can see from Figure 4.11 that BroadCatch can compete with the GFB technique for low server bandwidth. However, Figure 4.12 (a zoom-in of figure 4.11) shows that BroadCatch for server bandwidths between *15b* and *20b* illustrates approximately a 100% decrease of client waiting time compared to the best case of GFB. This is mainly due to the fact that in BroadCatch when adding a server channel the access latency to the video is reduced by half, which is much faster than the Fibonacci sequence. Furthermore, it was shown in [YK03] that GFB(8/4), GFB(16/8), and GFB(32/16) have not much gain compared to GFB(4/2).

Indeed, GFB(6/2) or GFB(8/2) outperform the BroadCatch technique in terms of access latency, however in those cases all clients in the system should be 3b or 4b bandwidth capable to access the video without jitter, in which case it not only limits the download capabilities of the client but also increases the cost of the overall system.

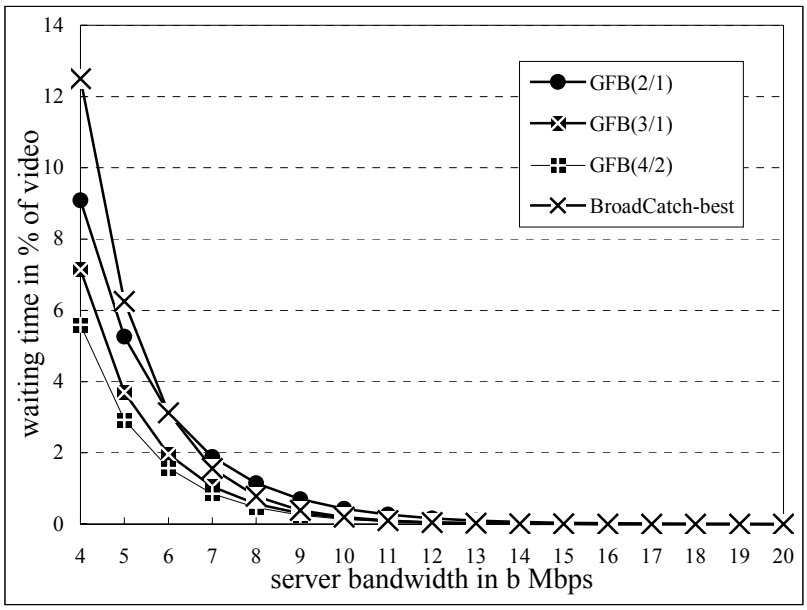


Figure 4. 11 BroadCatch vs. GFB

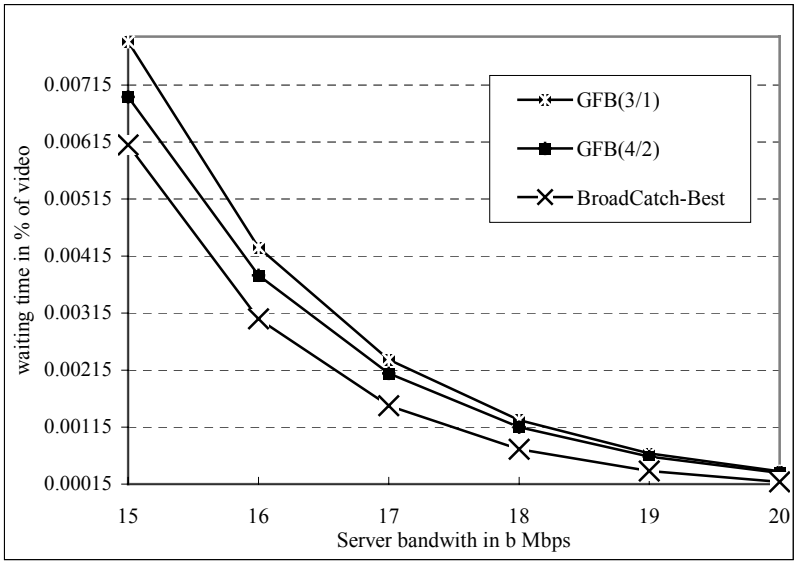


Figure 4. 12 BroadCatch vs. GFB (A zoom-in)

4.5.5 Cost of BroadCatch in Server Bandwidth

It was shown in [PCL98c, SGT01, YK03] that any periodic broadcast strategy would require the server bandwidth to be at least $\approx b \cdot \ln(L + d/d)$. Figure 4.13 shows this optimal value for different length of the video with a worst access delay *d* that ranges from 10 to 200 seconds. In the case of the *BroadCatch* technique the server bandwidth is given by:

$$server\ bandwidth = b \cdot (\left\lceil \log_2(\frac{L}{d}) + 1 \right\rceil - 0.5) \qquad (4.13)$$

The server bandwidth is computed from Equation 4.4, where it is equal to $b \cdot K$. Since K is an integer and represents the number of channels used, therefore the upper value of the first part in Equation 4.13 is taken. The second part derives from the fact that in the BroadCatch technique, the last channel is used only 50% of the time.

From Figure 4.13 one can notice, when the video length is one hour and a half (5400 seconds) with a delay *d=90* seconds, that BroadCatch employs an extra 2.5b Mbps to achieve heterogeneity, and at maximum an extra 3.5b for other cases. One also observes that this cost is negligible since it will allow an infinite number of users to use only few channels to download the video, as opposed to the optimal case where similar bandwidth should be matched at the user site. Undeniably, this extra cost allows users, in the BroadCatch technique, arriving at specific time slots to use only two channels to download the video. Hence, this cost is insignificant when a large number of users are utilizing the system.

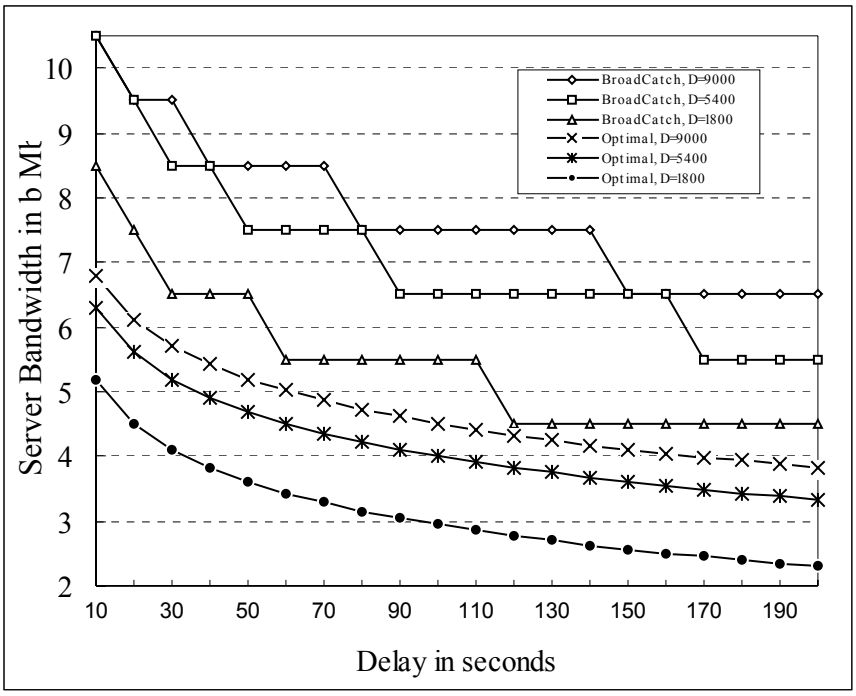


Figure 4. 13 Cost of BroadCatch

One can also remark that since the BroadCatch technique can only reduce its delay by half, using 6.5 channels (6 channels periodically broadcasting segments at full time, while another channel only half of the time) and a video length of L =5400 sec, shows a delay of less than 90 seconds. The latter delay is applicable in all cases where BroadCatch uses 6.5 channels, similarly for 5.5 channels the delay is less than 170 seconds for L=5400 sec.

4.6 Summary

This chapter proposed a new periodic broadcast technique, called *BroadCatch*, to support user heterogeneity. Given a unique BroadCatch delivery, a user can choose his/her reception schedule that fits his/her available bandwidth. In fact, a user can have an array of time slots where he/she can starts downloading the video, for each time slot corresponds a specific bandwidth requirement.

Using *BroadCatch* channels, a user downloads the first portion of a video and catches up to a base channel to continue the download of the rest of the video. In each time slot, the user computes the bandwidth needed for that purpose. If the user does not have enough bandwidth for that specific time slot, he/she has to wait for the next *time slot*, which presents less demanding bandwidth.

Periodic broadcast is mainly used in applications that need to serve a large community of users, such as movie on demand, news on demand, video catalog in electronic commerce, video announcement, etc. As such, the cost and usefulness of the overall system is much impacted by the requirements on the receiving bandwidth. To evaluate the performance of the BroadCatch technique, it is compared to a recently proposed method known as HeRO. Our analysis indicates that for each supported receiving bandwidth, BroadCatch offers significantly better service latency. In other words, BroadCatch is less demanding on client bandwidth in achieving a certain desired service delay. This scheme is therefore less expensive to provide heterogeneous services.

CHAPTER FIVE: MULTICAST vs. PERIODIC BROADCAST: A CASE STUDY IN DISTRIBUTED INTERACTIVE SIMULATION

In real-time interactive training and simulation systems, changes to the natural terrain surface and other non-moving objects are important to simulators. In fact, entities moving to new regions need to know the latest updates to the dynamic terrain. This chapter studies ways to disseminate dynamic terrain updates to simulators in a distributed interactive environment and is organized as follows. Section 1 discusses the distributed interactive simulation environment. The problem of Terrain updates is introduced in section 2. Two techniques that communicate dynamic terrain changes to simulators are analyzed in section 3. Section 4 shows simulation results to compare the proposed techniques and section 5 summarizes this chapter.

5.1 Introduction

Over the past years, there has been an increasing demand on supplementary realism within a simulation training system. To achieve this purpose, dynamic terrain plays an important role in enhancing and complementing 3D image generation. In a large-scale virtual environment, participants can modify the terrain and at the same time interact with other participants. However, dynamic terrain has a potential of producing a high degree of network traffic between simulated entities. Furthermore, and to meet the demands of heterogeneous simulations, a platform-independent dynamic terrain core module is essential. The core module would allow a multitude of simulation systems to add real-time dynamic terrain functionality by providing

interfaces that would remain common across all platforms.

The High Level Architecture (HLA) presents a framework for modeling and simulation within the Department of Defense (DoD). The goal of such architecture is to interoperate multiple simulations and facilitate the re-use of simulation components. The RTI, one of the main components of the High Level Architecture, is a collection of softwares that offers commonly required services to simulation systems, it also provides a degree of portability (across computing platforms, operating systems, and communication systems) and simulation interoperability [DF98].

In this framework, two novel schemes to communicate the dynamic terrain updates to simulated entities are presented in this chapter. Both schemes rely on a specific terrain partitioning. The first scheme uses a static multicast technique to communicate the changes of the dynamic terrain to simulators. In this environment, simulators make requests of dynamic terrain updates to the server (or set of servers); and it serves requests according to some scheduling policy. Each simulator expressing an interest in getting updates to the dynamic terrain, inside its region of interest, issue a request to a server or a set of servers. In the second scheme, simulators do not make requests to get the updates of the dynamic terrain; rather the server (or set of servers) periodically broadcast the updates in separate channels based on the terrain partitioning. Simulators then tune to the appropriate channels to get the latest updates to the dynamic terrain. Periodic broadcast content communication can provide a better latency in several situations. In this chapter focuses on exploring such situations.

5.2 Terrain Updates Problem Description

A dynamic terrain is a set of non-moving objects that can change in time or through an event triggering. For example if a tank is driving near a river bank, but not actually in the river, the ground underneath the tank may be soft. When the tank moves it should leave marks in the soft ground. Therefore, objects in the terrain can be classified into two main categories; moving objects, which are objects that can move in the terrain, such as tanks, soldiers, or airplanes, and non-moving objects, such as trees, bridges, the ground, etc. In the rest of this chapter we will refer to the first type of objects as dynamic entities, and the second type as dead entities. A dead entity is an observable structure of the simulation space. Not only that, but dynamic entities need to know states of dead entities for accurate representation of their simulated world which can influence their future behavior.

Dynamic entities have an *area of interest*. The area of interest for each dynamic entity is the space of the dynamic terrain that the entity can interact with. In general, an area of interest is divided into two categories; a *subscription region* and an *update region*. Each dynamic entity expresses an interest in receiving data and therefore subscribing to a specific region, and in sending data or updating a specific region. If a subscription region intersects an update region a communication between the subscribing entity and the updating entity takes place. Subscription and update regions are in fact filters to bind the routing space, hence reducing the amount of data received by each entity. Much previous works has been done on how to efficiently manage and distribute data between dynamic entities to reduce their interaction cost. The work in [MBDT99] improves the performance of the data distribution management through dynamic assignment of

the multicast groups. The Authors of [RSM97] proposes a prototype on how to efficiently use selective services to perform the data distribution management. A region or a routing space do not necessarily map to a physical geographical region, but to a multidimensional coordinate. In [RSM97] there are three factors to design the routing space, a range of interest, an interaction of interest, and a level of granularity which is the smallest portion of the space that can be individually subscribed. For example, a tank can subscribe to the ground space with a perimeter not more than 15kms of range, an interaction of an air_sea_space if the tank is to detect aircrafts, and a relatively fine granularity cells if there are many such ground entities. An aircraft on the other hand would express an interest in a larger range of interest with a larger level of granularity than the ground entities.

Data distribution management (DDM) in the High Level Architecture (HLA) organizes the distribution and routing of data between several entities. To prevent the delivery of irrelevant data by the Runtime Infrastructure (RTI), and therefore improve the utilization of processing and network resources, entities express their interest through the DDM services. To implement DDM, there is either a fixed grid method [RSM97] or a distributed region method [VC98]. Both methods use multicast routing as a mechanism for low-level relevance filtering. The main difference between the two methods is the association of the multicast groups and region matches for communication purposes.

In the fixed grid method, the RTI initially partitions the routing spaces along each dimension to perform grid cells, and then assigns a multicast group to each grid cell. Each entity determines the grid cells that a region intersects. The result associates a region with several multicast groups

in a fixed and predetermined manner. In the distributed region method, each entity exchanges its region information with other remote entities. The entity matches its local region used for subscription against remote update regions, and vice versa. The advantage of the fixed grid method is that entities do not require exchanging region information, however intersection between regions and grid cells do not reflect the exact intersection between the subscription and update regions. The advantage of the distributed region method is that entities communicate only relevant data to other entities, however as the number of regions increases region matching performs poorly [HWT01]. Figure 5.1 gives an example of the fixed grid method and the distributed region method.

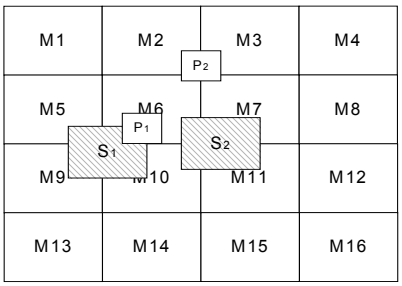


Fig 5.1.a. Fixed Grid

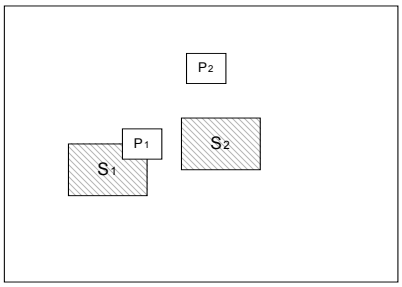


Fig 5.1.b. Distributed Region

Figure 5. 1 Fixed Grid vs. Distributed Region

In Figure 5.1.a, the publication regions, Pi, represent regions used for sending interactions and updating objects. The subscription regions, Si, represent regions used for subscribing to and receiving data. An entity e_i that subscribes to S_2 region joins M_6,M_7,M_{10}, and M_{11} multicast groups to receive data. Although an entity e_j that publishes using P_1 do not intersect with S_2, e_j will send data to e_i.

In the case of distributed regions, each update region has a multicast address, and a subscription region join the multicast group of overlapping update regions. In Figure 5.1.b, only entities that subscribe using S_1 and update using P_1 can communicate.

Static or dead objects cannot move on their own. They have an initial state at the beginning, and can change to different states through the entire simulation. Moreover, the state of dead objects must appear the same to all entities requesting similar interest. One of the main issues is the distribution of the change of states of dead objects. When an entity update a dead object, it can either communicate to other entities the action of the update and let the receiving entity calculate the update, or send the updated state. The latter case has the advantage of lessening the computational load of the receiving entity. Another issue is the distribution method of the dynamic terrain [LHM95]. When an entity updates the dynamic terrain, the entity sends updates to all other entities listening to its update region. However, other entities that were not subscribed to the same region would not get the update, and therefore would have an inconsistent view of the dynamic terrain. One way to solve this problem is to let a server or a set of servers collect the changes and communicate them to entities that requested them. The one-server approach

presents a simplified correlation model between simulators. Moreover, joining entities would go to one source to get the later updates of the dynamic terrain. However, it also presents one single point of failure. The several-server approach has the advantage of using each server for a particular region, and simulators would only interface the server within their own region. Another way is to use peer-management distribution. In this case each simulator on the network would manage a distribution list, and is responsible for its update and its distribution.

Without loss of generality, the responsible agents for collecting dynamic terrain updates and communicating them to simulators in the network is referred to as a *collector*. The collector could use a single server, several servers, or a peer based approach. We assume that each simulator has an initial copy of each dead object of its interest. The advantage of having a replicated homogenous world is that networked messages are object changes, which make the communication cost relatively small. The disadvantage is that it is relatively inflexible and that as the terrain content increases so must every site's database. Figure 5.2 depicts the global architecture.

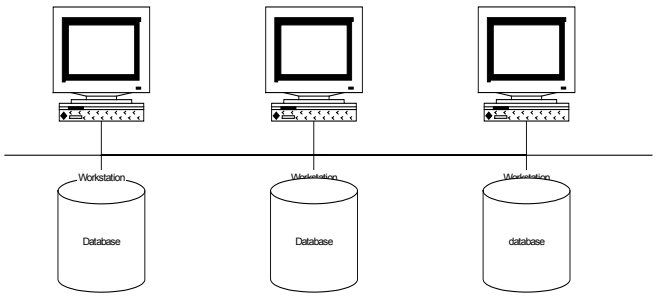


Figure 5. 2 Fully replicated databases for all simulators

Next section describes two novel methods that communicate dynamic terrain changes based on a specific terrain partitioning.

5.3 Proposed Techniques

The following section proposes two approaches to manage the dynamic terrain updates. The first approach uses a multicast strategy and the second approach a streaming periodic broadcast strategy. Both approaches rely on the partitioning of the dynamic terrain discussed in the next sub-section.

5.3.1 Dynamic Terrain Partitioning

Partitioning of the dynamic terrain uses a *clustering factor*. The clustering factor is a threshold from which several cells are constructed in the dynamic terrain, same as the fixed grid approach discussed in section 5.2 but based on the distribution of dead-objects. Therefore, each terrain will have its fixed but proper partitioning.

At the beginning, the entire terrain is considered as one cell. If the number of dead objects in the cell is bigger than the clustering factor, then the cell is split into two cells vertically. Each cell will then split horizontally if its content has more object than the clustering factor. Recursively, the terrain is partitioned into cells, until no cell exceeds the clustering factor. In each split action, the original cell generates two cells; a left cell and a right cell if the split action is vertical, or an

upper cell and lower cell if the split action is horizontal. A tree of cells can therefore represents all the action splits, where leaf nodes represent the resulting and actual cells of the partitioning, and internal nodes represent intermediate cells in the process of partitioning. Figure 5.3 gives an example of how the dynamic terrain is partitioned using a clustering factor of 3. Figure 5.4, gives the corresponding tree of the same partitioning. A higher clustering factor will generate a bigger tree and vice versa. Because the content of each cell will not exceed the clustering factor, a balanced tree generation means that dead objects are uniformly distributed throughout the terrain and a skewed tree generation means that dead objects are clustered in a smaller region.

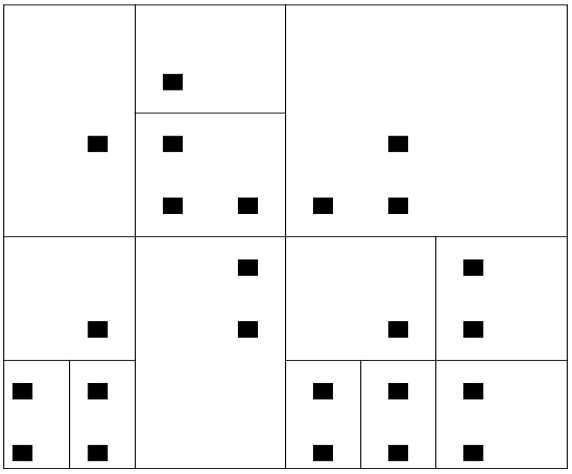


Figure 5. 3 Dynamic Terrain Partitioning

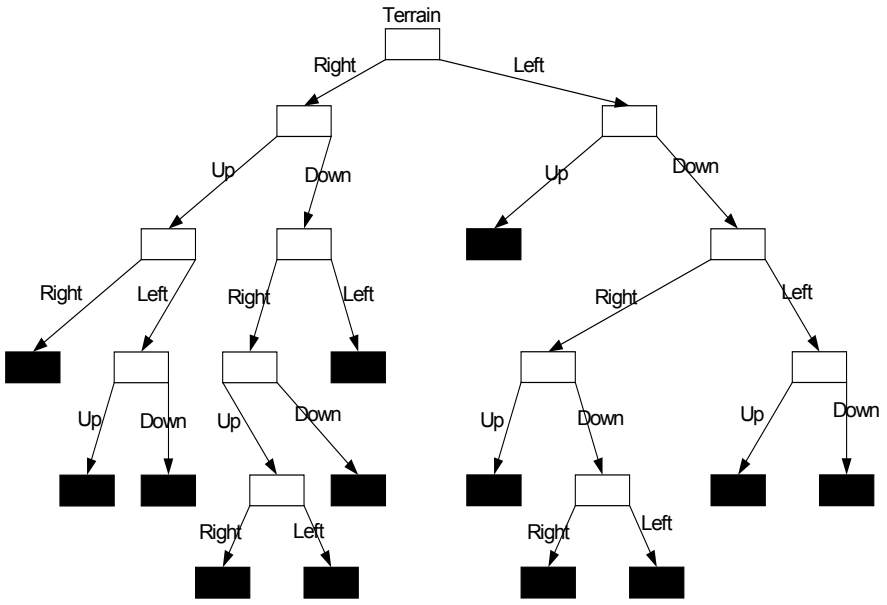


Figure 5. 4 A Tree Construction for the Partitioning

5.3.2 Multicast and Streaming Periodic Broadcast Techniques

Using the cells provided by the partitioning strategy described earlier, we propose two techniques that communicate the update changes of the dynamic terrain to the simulators. An entity e_i that updates a dead object communicate this change to all entities that have a subscription region intersecting e_i's publishing region, and also to the collector that manages those updates for future entities. The collector will only keep the latest change to the dead object and is responsible to communicating it to other entities. Each entity e_i has an area of interest AOI_i that represents the subscription region of the entity regarding dead objects. An area of

interest AOI of an entity can intersect several cells. The intersecting set of cells (*IC*) represents the current dynamic terrain that the entity is interested in. When an entity e_i moves, its intersecting cells IC_i can change. Figure 5.5 gives an example of the AOI of the entity e_i that intersects the set {1,2,3}of cells. An entity requests the content of its intersecting cells only if this one changes. In other words, when the IC set is the same, the entity is not moving much, and therefore e_i does not need to know any extra information about the terrain. If at the same time, other entities made changes to the terrain, those entities are responsible to communicating the updates to e_i if their publish region intersect IC_i. In both techniques, we do not focus on the inter-communication between entities, but rather in the communication between the collector and the entities, which holds for most of the dynamic terrain management.

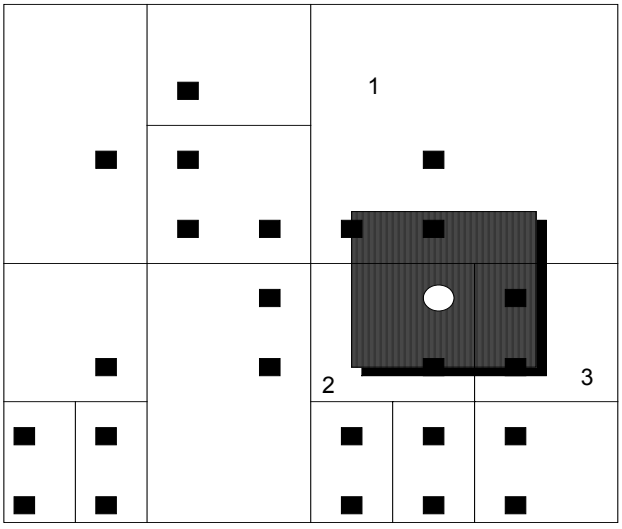


Figure 5. 5 Intersecting Cells IC of an entity

5.3.2.1 Multicast Technique

When an entity moves its IC can change from IC_t to $IC_{t'}$. Whenever an entity's IC changes, the entity makes a request to the collector that consist of the set of cells ($IC_{t'}$ - IC_t), which is the difference in cells after and before the change to the entitie's intersecting cells. The collector then gathers requests in a pool of request, batch the requests to the same cells in a multicast group, and send it to their appropriate receivers.

The algorithm for the entity and the collector using multicast is described more formally in Figure 5.6.

```
Entity's Algorithm
While (true){
  Calculate IC
 If (IC ≠IC-old)
  // Request the appropriate cells
  Send_request _to_collector(IC – IC-old)
 }
Collector's Algorithm
// N total number of entities
For all Entities from i to N{
   Receive(id_entity, requesting_cells)
}
batch entities requesting same cells into multicast groups
for each multicast_group
  send(appropriate_cellInfo)
```

Figure 5. 6 Entity and Collector's Algorithm

5.3.2.2 Streaming Periodic Broadcast Technique.

In this technique, entities express their interest in getting the content of their IC cells in the same way as in the multicast technique. However, entities do not make requests to the collector. Rather, the collector keeps the latest update states of all dead objects and periodically broadcast those updates to every entity. If we denote by ***C*** the capacity of the collector, which is the maximum number of state of dead objects that can be sent by the collector in each unit of time, and ***Nc*** the total number of cells generated by the dynamic terrain partitioning, then we can allocate to each cell a capacity of ***C/Nc***. The collector allocates to each cell a channel to stream all changes made to dead objects inside the cell. Therefore, each channel has a capacity of *C/Nc*. We denote by a ***cycle*** the amount of time taken to send all states of dead objects. Figure 5.7 is an example of two streams that corresponds to $cell_i$ and $cell_j$. The cells have *m* and *n* number of dead objects respectively. $S_{i,j}$ represents state j of dead object i. During the first cycle, object O_1 and O_n in $cell_i$ have changed to another state, whereas in the same cycle, there was no activity in $cell_j$. Entities wishing to download the latest updates to the dynamic terrain tune to the appropriate channels based in the difference of their intersecting cells IC.

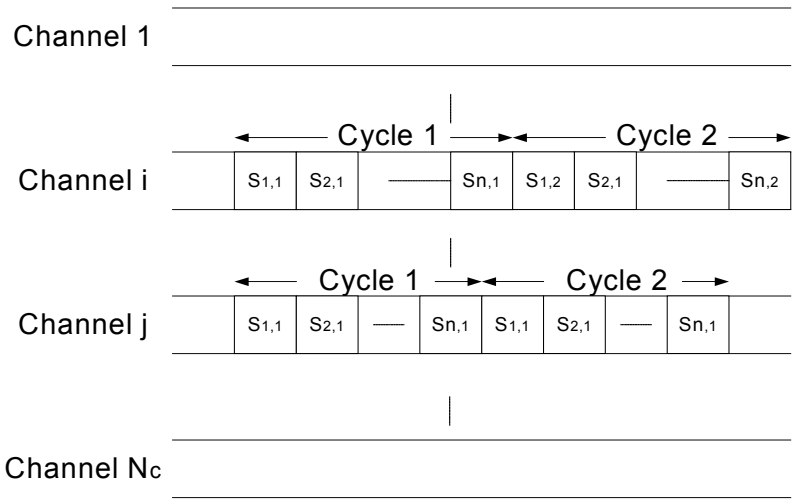


Figure 5. 7 Streaming updates in the periodic broadcast

5.4 Performance Study

To evaluate and compare the performance of both techniques, detailed simulators are implemented. This section discusses the simulation study. First the entity behavior model is presented, then the performance metrics and simulation variables are discussed, and finally simulation results are presented in the last section.

5.4.1 Entity Behavior Model

Each entity e_i has a speed, an initial direction, and an area of interest represented by a square. For

the sake of experimentation the speed during the same simulation is not varied. Entities can move inside the terrain in 9 different directions, which are, north, south, east, west, northeast, northwest, southeast, southwest, and idle. Idle is also considered a direction since an entity can stay at the same place for a longer time. $\mathbf{P_s}$ is denoted as the probability that an entity stays in its current direction. The entity changes from its current direction to another direction with a probability $\mathbf{(1-P_s)/8}$, that is all other directions have the same probability once the entity change its direction. By setting a higher $\mathbf{P_s}$ we can achieve a more realistic movement of entities. When an entity encounters the boundary of the terrain, the entity chooses the opposite direction.

5.4.2 Performance Metrics and Simulation Variations

To compare the multicast and the streaming periodic broadcast techniques, one can define the latency to be the waiting time of an entity to get the content of one specific cell from the time the entity showed interest in that cell to the actual time it gets it. It is interesting to recall that for both techniques, entities are interested in the content of the cell if that cell is in the set (IC – IC-old). One is therefore interested in the average latency for all cells. The lowest is the average latency the better is the technique. The parameters used in the simulation are described next, and their variation is given in table 5.1.

- C: Bandwidth of the collector (or server) in Mbps.
- E: Number of entities simulated.
- SimTime: Simulation Time in Hours

- SizeTerrain: The size of the terrain in km^2.
- O: Total number of dead objects in the terrain.
- SizeState: Size of a state change of objects in Bytes
- Cf : The Clustering Factor.
- AOI: Size of the side of a square in meters representing the area of interest.
- Sp: Speed of entity in km/h.

Table 5. 1 Simulation Parameters and Variations

Simulation Parameter	Variations
Ps	0.75
C	1.5 Mbps
SimTime	1 Hour
SizeTerrain	Square of 100 km^2
O	200,000
SizeState	10 Bytes
E	100 - 5000
Cf	50 – 600
AOI	10 – 100 meter
Sp	36 – 1980 km/h

5.4.3 Simulation Results

In the rest of the simulation, dead objects in the terrain are uniformly distributed. The simulation results are presented in the following subsections, it shows that under extensive traffic the streaming periodic broadcast outperforms the multicast technique. Then it examines the scalability factor for both techniques in terms of the number of entities. Finally it illustrates the effect of changing the size of the AOI and the clustering factor on the average latency. In the next simulation runs the clustering factor *Cf* is set to be 200.

5.4.3.1 Effect of changing the speed of entities

The traffic of the entities by varying their speed between 36 km/h to 1980 km/h is simulated. The number of entities E is set to be 1000, each with an area of interest AOI of 50 meters. Figure 5.8 shows the performance of both the streaming periodic broadcast and multicast techniques. One can see from the figure that the multicast technique outperforms the streaming periodic broadcast for small speeds. However, the streaming periodic broadcast has better average latency when the speed of entities is approximately 180 km/h and bigger. The streaming periodic broadcast scales very well with the extensive traffic. In fact, as entities move faster, as they share more the streaming channels. However, the multicast technique performs well for small speeds but reaches a maximum very quickly.

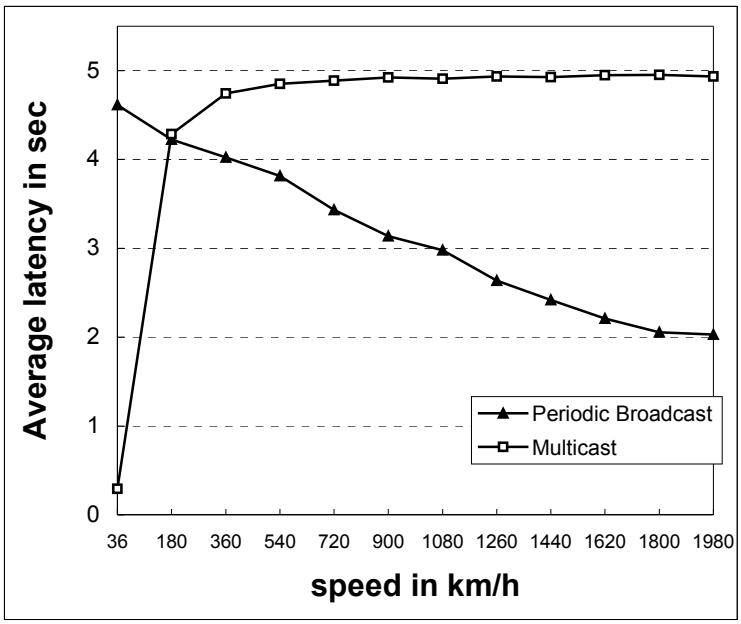


Figure 5. 8 Effect of changing the speed parameter

5.4.3.2 Effect of changing the number of entities

In this simulation, the number of entities is varied from 100 to 5000 entities. Figure 5.9 gives the performance for both techniques for 3 different speeds, 36, 180, and 360 km/h. The same area of interest is kept for all entities as before namely 50 meters. The multicast technique exhibits similar behavior as the last simulation, especially for 180 and 360 speeds. However, the streaming periodic broadcast scales well relatively to the number of entities. In fact, the streaming periodic broadcast does not depend on the number of entities in the dynamic terrain, which is desirable for large-scale virtual environments.

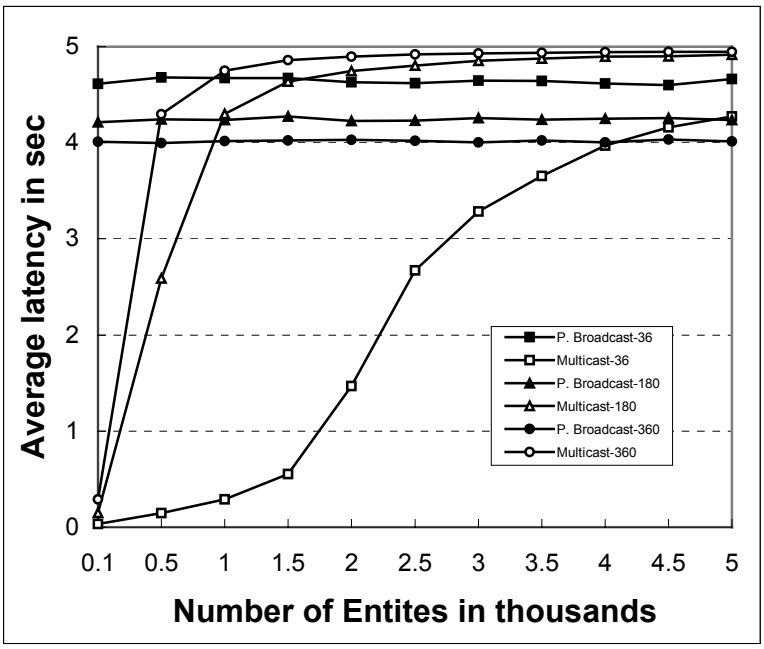


Figure 5. 9 Effect of changing the number of entities

5.4.3.3 Effects of changing the speed of entities and the number of entities

In this simulation the same parameters as before are kept but the number of entities from 100 to 5000 is varied with a step size of 500 entity, and the speed of entities from 36 km/h to 1440 km/h with a step size of 180 km/h. Figure 5.10 and 5.11 gives the surface generated for both the multicast technique and streaming periodic broadcast respectively. Each surface consists of 99 points that represents each simulation run.

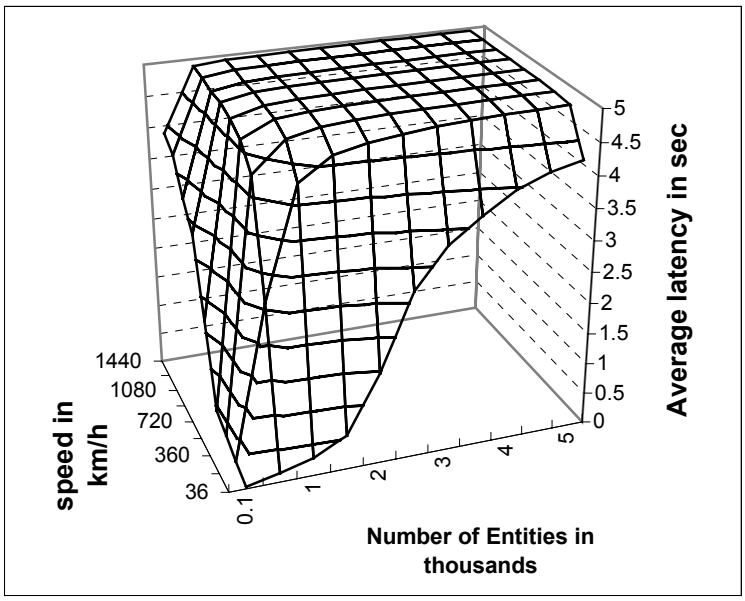


Figure 5. 10 Surface for the multicast technique

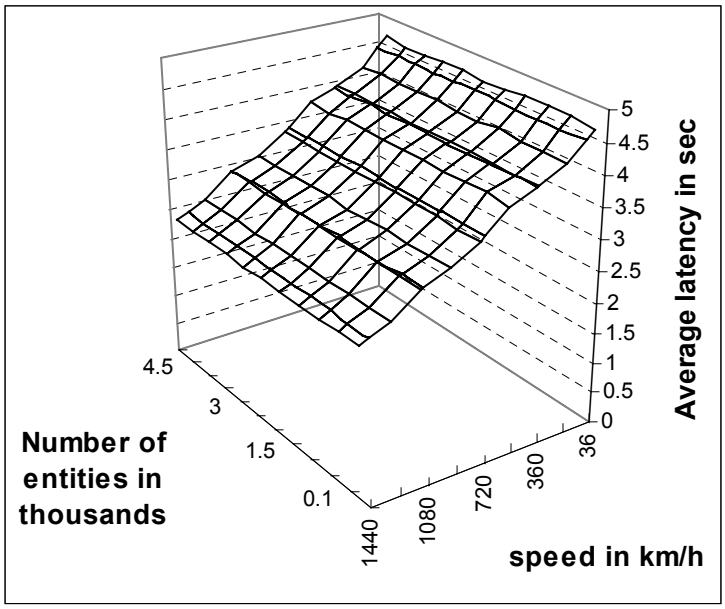


Figure 5. 11 Surface for the streaming periodic broadcast

5.4.3.4 Effect of changing the area of interest

In this simulation, the number of entities is set to 1000 and the speed of all entities to 180 km/h. The area of interest AOI side is varied from 10 meters to 100 meters. The result of the simulation is given in figure 5.12. Again, in the streaming periodic broadcast technique, entities can exploit more sharing of the streaming channels as the area of interest of entities increases.

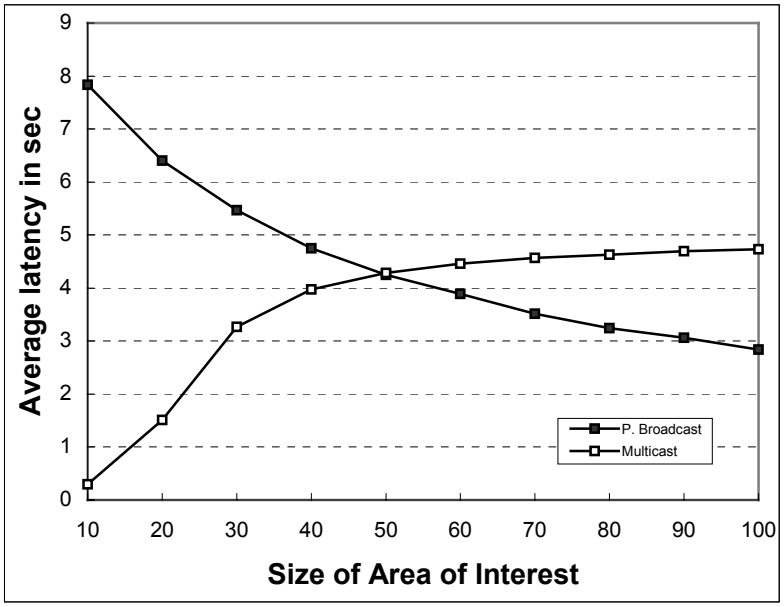


Figure 5. 12 Effect of the size of the area of interest

5.4.3.5 Effect of changing the clustering factor

In all previous simulations the clustering factor is set to 200 dead objects, which splits the terrain into approximately 1300 cells. In this simulation, the same parameters are kept for entities as before, namely E =1000, Sp = 180 km/h, and the clustering factor changed from 50 to 600 dead objects with a step size of 50 objects. Table 5.2 gives the number of cells in the terrain for the different values of the clustering factor. In fact, the number of cells is inversely proportional to the clustering factor because dead objects are uniformly distributed in the terrain. From Table 5.2, the number of cells for a Cf=250 is the same as for a Cf=300, similarly for clustering factors of 500,550,and 600. This is due to the partitioning, of the terrain, in which the small increase in the clustering factor would not necessarily decrease the total number of cells generated. Figure 5.13 gives the performance of both the multicast and the streaming periodic broadcast techniques. One can see from the figure that for those specific parameters in the simulation the multicast technique does not depend on the clustering factor. Whereas, in the streaming periodic broadcast technique as the clustering factor decreases the total number of cells increases, as the streaming periodic broadcast performs well. In fact, the total number of cells is also the number of channels, and therefore if clustering number is small, entities will share more channels. The fluctuation presented in Figure 5.13 for the streaming periodic broadcast is due the partitioning and the relation between the number of cells and the clustering factor discussed earlier.

Table 5. 2 clustering factor and number of cells

Cf	N. of Cells	Cf	N. of Cells
50	5684	**350**	1017
100	2813	**400**	663
150	2047	**450**	514
200	1369	**500**	512
250	1024	**550**	512
300	1024	**600**	512

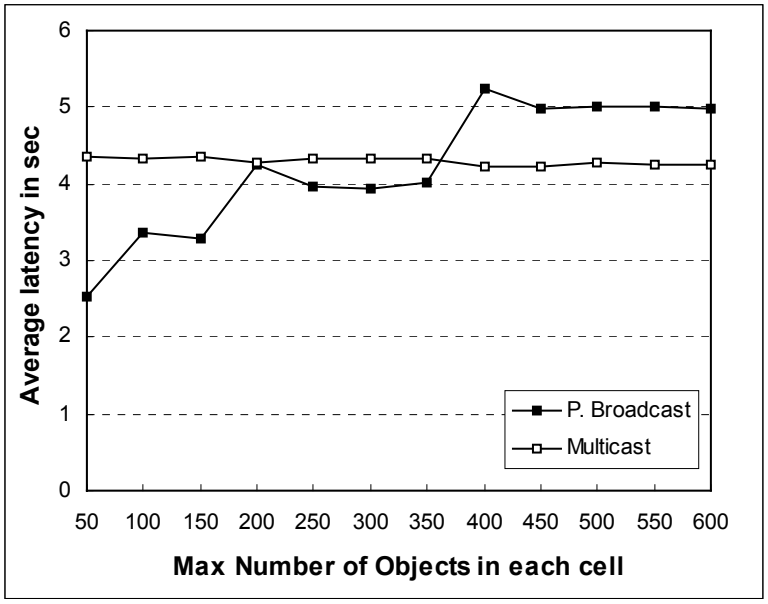


Figure 5. 13 Effect of changing the clustering factor

5.5 Summary

In this chapter, the problem of communicating the dynamic terrain updates to simulators is introduced. Entities moving to new regions need to know the latest updates to the dynamic terrain. Two novel approaches are then proposed to solve this problem using a specific terrain partitioning. The first scheme uses a multicast approach and the second scheme a streaming periodic broadcast approach to get the latest updates to the dynamic terrain.

Through extensive simulation, this chapter shows that under extensive traffic, the periodic broadcast scheme outperforms the multicast approach. Furthermore, as the number of entities increases, the streaming periodic broadcast technique scales well, this is particularly due to the fact that the periodic broadcast technique is independent of the number of entities the system can support.

CHAPTER SIX: CONCLUSION

Video on demand is certainly a promising technology for many multimedia applications still to come. Unlike traditional data, delivery of a video can take up substantial bandwidth for a long period of time. Recent research has shown that periodic broadcast is an efficient paradigm to address the limitation of the network I/O bandwidth of video servers. In a periodic broadcast environment popular videos are pushed to end-clients.

Two alternatives have been considered for addressing this limitation [HTT04]: the *Server-Oriented Approach* (e.g., Skyscraper Broadcast, Striping Broadcast, Client-Centric Approach) reduces service delay by increasing server bandwidth; whereas the *Client-Oriented Approach* (e.g., *Cautious Harmonic Broadcast, Pagoda Broadcast*) requires client to equip with significantly more download bandwidth. Chapter 2 discusses in details both alternatives and described some cost effective and scalable solutions for the deployment of VOD systems.

VCR-like interaction is a desirable feature for many VOD applications. It provides a convenient environment to browse and search for video content. For periodic broadcast, the *broadcast-based interaction* technique (BIT) [THS02, THS02b] offers a highly scalable solution. In fact, the amount of server bandwidth required to support interactivity is independent of the number of users currently using this service. Chapter 3 described the BIT technique in detail and compared its performance to the best to date technique that uses data prefetching to achieve VCR functionality.

Another important consideration in designing VOD systems is the capability to handle receiver

heterogeneity. Multi-resolution encoding techniques provide a good solution for many applications. For those that demand the same high QoS for clients of various capabilities, techniques such as the BroadCatch technique [THD04] can be used in the periodic broadcast framework. BroadCatch is a different approach that handles differences in receiving bandwidths using only a single broadcast scheme. This is achieved by indicating in the broadcast when a client with a particular receiving capability can start its download. Chapter 4 described the BroadCatch Technique and compares its performance to the most recent technique that provides user-bandwidth heterogeneity, and to a recent technique that minimizes end-client's bandwidth without providing heterogeneous reception. BroadCatch technique is shown to present better performances and a simplicity and effectiveness in implementation.

Chapter 5 presented two techniques that disseminate changes to a terrain database in a distributed interactive simulation environment [THJ03]. The first technique uses a static multicast delivery approach to deliver the update changes to the terrain, while the second technique uses a periodic broadcast streaming approach. Both techniques use the same terrain partitioning. In this chapter, it was shown that as the number of simulators increase and/or the speed of entities in the terrain increases, the streaming periodic broadcast present better performances.

Multicast and Broadcast can be employed on both LANs and WANs. Nodes connected to a LAN often communicate via a broadcast network, while nodes connected to a WAN communicate via switched networks. In a broadcast LAN, transmission from any node is received by all the other nodes on the network; therefore it is simple to implement Multicast/Broadcast on a broadcast LAN. On the other hand, and due to scalability issues and to the fact that today's WANs are

designed to mainly support Unicast communications, it is difficult to implement a Multicast/Broadcast on switched network.

Future research will focus on streaming bandwidth-intensive media on demand from a source to a large number of receivers in the Internet. A simple solution would be to dedicate an individual connection to deliver the content to each receiver. However, this method would consume a tremendous amount of server bandwidth. Another option is the use of IP multicast, where for each client requesting the same video, the server creates a multicast address and sends a stream of the same video to all clients in the multicast tree. This way, server and network bandwidth are optimized. However, and to provide late arrivals to the system, the server has to create new multicast addresses. This process could therefore generate a large number of separate multicasts and would therefore exhaust server and network bandwidths.

This study will focus on how to efficiently provide clients with Broadcast services at the application layer using only the IP Unicast. Protocol. It will mainly focus on how to deploy relay nodes or special machines whose only objective is to act as routers at the end-system nodes that can forward video data to any newly incoming arrival.

APPENDIX A: WORST USER-BANDWIDTH CLOSED FORM

For each step n and $0 \le k < 2^{n+1}$ we have:

$$wub(k,n) = u(\lfloor k/2 \rfloor, n-1) + \left\lfloor \frac{k - 4 \cdot \lfloor k/4 \rfloor}{3} \right\rfloor$$

If we consider the function $f(x)$ to be $f(x) = \lfloor x/2 \rfloor$, and the notation $\underbrace{f \circ f \circ f \cdots f \circ f}_{i}(x) \equiv f^{(i)}(x)$, where $f^{(0)}(x)=x$, $f^{(1)}(x)=f(x)$, and so forth. Then we can write $wub(k,n)$ as the following:

$$\begin{aligned} wub(k,n) &= wub(f^{(i)}(k), n-i) + \sum_{j=0}^{i-1} \left\lfloor \frac{f^{(j)}(k) - 4 \cdot \lfloor f^{(j)}(k)/4 \rfloor}{3} \right\rfloor \\ &= wub(f^{(n)}(k), 0) + \sum_{j=0}^{n-1} \left\lfloor \frac{f^{(j)}(k) - 4 \cdot \lfloor f^{(j)}(k)/4 \rfloor}{3} \right\rfloor \end{aligned}$$

the index k can be written in its binary form as $(k)_b=(k_n \ldots k_1 k_0)$ and $k = \sum_{i=0}^{n} k_i \cdot 2^i$ where k_i's are either 0 or 1. Therefore:

$f^{(1)}(k) = \sum_{i=0}^{n-1} k_{i+1} \cdot 2^i$ and $f^{(j)}(k) = \sum_{i=0}^{n-j} k_{i+j} \cdot 2^i$ Hence we have:

$$
\begin{aligned}
wub(k,n) &= wub(f^{(n)}(k),0) + \sum_{j=0}^{n-1} \left\lfloor \frac{\sum_{i=0}^{n-j} k_{i+j} \cdot 2^i - 4 \cdot \left\lfloor \sum_{i=0}^{n-j} k_{i+j} \cdot 2^i \Big/ 4 \right\rfloor}{3} \right\rfloor \\
&= wub(f^{(n)}(k),0) + \sum_{j=0}^{n-1} \left\lfloor \frac{\sum_{i=0}^{n-j} k_{i+j} \cdot 2^i - 4 \cdot \sum_{i=0}^{n-j-2} k_{i+j+2} \cdot 2^i}{3} \right\rfloor \\
&= wub(f^{(n)}(k),0) + \sum_{j=0}^{n-1} \left\lfloor \frac{1}{3} \cdot \left(\sum_{i=0}^{n-j} k_{i+j} \cdot 2^i - \sum_{i=2}^{n-j} k_{i+j} \cdot 2^i \right) \right\rfloor \\
&= wub(f^{(n)}(k),0) + \sum_{j=0}^{n-1} \left\lfloor \frac{1}{3} \cdot (k_j + 2k_{j+1}) \right\rfloor
\end{aligned}
$$

for $k = 2^{n+1} - 1$ we have:

$$f^{(n)}(k) = 1 \Rightarrow wub(2^{n+1} - 1, n) = wub(1,0) + n = 2 + n$$

for $k < 2^{n+1} - 1$ we have:

$$f^{(n)}(k) = 0 \Rightarrow wub(k,n) = wub(0,0) + \sum_{j=0}^{n-1} \left\lfloor \frac{1}{3} \cdot (k_j + 2 \cdot k_{j+1}) \right\rfloor$$

in all cases the closed form is given by:

$$wub(k,n) = 2 + \sum_{j=0}^{n-1} \left\lfloor \frac{1}{3} \cdot (k_j + 2 \cdot k_{j+1}) \right\rfloor$$

APPENDIX B: HETEROGENEOUS SCALABILITY PROPOSITION

To prove the heterogeneous scalability proposition, first the following assertion is proved:

$$\forall n \quad s_n = 2^n \cdot s_0 + n \cdot 2^{n-1} \tag{B.1}$$

$$s_n = \sum_{k=0}^{2^{n+1}-1} wub(k,n) = \sum_{i=0}^{2^n-1} (wub(2i,n) + wub(2i+1,n))$$

from equation 4.6, *we have*

$$= \sum_{i=0}^{2^n-1} \left(wub(i,n-1) + wub(i,n-1) + \left\lfloor \frac{2i - 4\lfloor 2i/4 \rfloor}{3} \right\rfloor + \left\lfloor \frac{2i+1-4\lfloor (2i+1)/4 \rfloor}{3} \right\rfloor \right)$$

$$= 2\sum_{i=0}^{2^n-1} wub(i,n-1) + \sum_{i=0}^{2^n-1} \left(\left\lfloor \frac{2i - 4\lfloor 2i/4 \rfloor}{3} \right\rfloor + \left\lfloor \frac{2i+1-4\lfloor (2i+1)/4 \rfloor}{3} \right\rfloor \right)$$

$$= 2s_{n-1} + \sum_{i=0}^{2^n-1} \left(\left\lfloor \frac{2i - 4\lfloor i/2 \rfloor}{3} \right\rfloor + \left\lfloor \frac{2i+1-4\lfloor (2i+1)/4 \rfloor}{3} \right\rfloor \right)$$

changing the index i to $i=2j$ and $i=2j+1$ we have:

$$= 2s_{n-1} + \sum_{j=0}^{2^{n-1}-1} \left(\left\lfloor \frac{4j - 4\lfloor 2j/2 \rfloor}{3} \right\rfloor + \left\lfloor \frac{4j+2-4\lfloor (2j+1)/2 \rfloor}{3} \right\rfloor + \left\lfloor \frac{4j+1-4\lfloor (4j+1)/4 \rfloor}{3} \right\rfloor + \left\lfloor \frac{4j+3-4\lfloor (4j+3)/4 \rfloor}{3} \right\rfloor \right)$$

$$= 2s_{n-1} + \sum_{j=0}^{2^{n-1}-1} \left(\lfloor 0 \rfloor + \left\lfloor \frac{1}{3} \right\rfloor + \left\lfloor \frac{2}{3} \right\rfloor + \left\lfloor \frac{3}{3} \right\rfloor \right)$$

$$= 2s_{n-1} + 2^{n-1}$$

Now, Equation B.1 is proved, by induction on n.

For $n=0$, $s_0 = 2^0 \cdot s_0$, for $n+1$ we have,

$$
\begin{aligned}
s_{n+1} &= 2s_n + 2^n \\
&= 2 \cdot (2^n s_0 + n \cdot 2^{n-1}) + 2^n \\
&= 2^{n+1} s_0 + n \cdot 2^n + 2^n \\
&= 2^{n+1} s_0 + (n+1) \cdot 2^n
\end{aligned}
$$

From Equations 4.9 and B.1 we know that,

$$
awub(n) = \frac{3 + \sum_{k=0}^{n} s_k}{2^{n+2}} = \frac{1}{2^{n+2}} (3 + \sum_{k=0}^{n} s_0 2^k + k \cdot 2^{k-1})
$$

therefore,

$$
\begin{aligned}
\forall n \quad awub(n+1) - awub(n) = \frac{1}{2^{n+3}} \Big[& 3 + \sum_{k=0}^{n+1} (s_0 2^k + k \cdot 2^{k-1}) \\
& - 6 - \sum_{k=1}^{n+1} (s_0 2^k + (k-1) \cdot 2^{k-1}) \Big] \\
= & \frac{-3 + s_0 + \sum_{k=1}^{n+1} 2^{k-1}}{2^{n+3}} \\
= & \frac{-3 + s_0 + 2^{n+1} - 1}{2^{n+3}} = \frac{s_0 - 4 + 2^{n+1}}{2^{n+3}}
\end{aligned}
$$

Since $s_0=4$ (*step 0* has only two time slots where *wub(0,0)=wub(0,1)=2*, then

$$
\forall n \quad awub(n+1) - awub(n) = \frac{2^{n+1}}{2^{n+3}} = \frac{1}{4} \square
$$

LIST OF REFERENCES

[AA94] K. C. Almeroth and M. Ammar. A Scalable Interactive Video-on-Demand Service Using Multicast Communication. In the Proceedings of the International Conference on Computer Communication Networks, pp. 292-301, 1994.

[AA96] K.C. Almeroth and M. Ammar. On the Use of Multicast Delivery to Provide a Scalable and Interactive Video-on-Demand Service. IEEE Journal of Selected Areas in Communications, vol. 14, No 6, pp. 1110-1122 August 1996.

[APS98] E. L Abram-Profeta and K. G. Shin: Providing unrestricted VCR functions in multicast video-on-demand servers. In the Proceedings of the IEEE International Conference on Multimedia Computing and Systems (ICMCS'98), pp. 66-75, Austin, Texas, 1998.

[AS00] S. Acharya and B. C. Smith. MiddleMan: A Video Caching Proxy Server. In the Proceedings of NOSSDAV 2000, paper# 16 Chapel Hill, NC, USA, June, 2000.

[AWY96] C.C Agarwal, J. L. Wolf, and P.S. Yu. On Optimal batching policies for video on demand storage servers. In the Proceedings of the IEEE ICMCS'96, pp.253-258, Jun. 1996.

[AWY96b] C. C. Aggarwal, J. L. Wolf, and P. S. Yu. A Permutation-based Pyramid Broadcasting Scheme for Video-on-Demand Systems. In the Proceedings of the Internatinal Conference on Multimedia Computing and Systems, Hiroshima, Japan, pp. 118-126. June, 1996.

[BGM99] H. M Briceo, S. Gortler, and L.McMillan. Naïve-network aware internet video encoding. In the proceedings of the 7th ACM International Multimedia Conference, pages 251-260, October 1999.

[BHO03] Olivier Bagouet, Kien A. Hua, and David Oger. A periodic broadcast protocol for heterogeneous receivers. In the SPIE Conf. Multimedia Computing and Networking 2003, (MMCN'03) pages 220-231, Santa Clara California, January 23-24 2003.

[BWGST01] M. K. Bradshaw, B. Wang, S. Sen, L. Gao, J. Kurose, P. Shenoy, and D. Towsley. Video Storage: Periodic Broadcast and Patching Services - implementation, measurement, and analysis in an internet streaming video testbed. In the Proceedgins of ACM Multimedia'01, pp. 638-639 Canada, October, 2001.

[BWGST01b] M. K. Bradshaw, B. Wang, S. Sen, L. Gao, J. Kurose, P. Shenoy, and D. Towsley. Periodic Broadcast and Patching Services - implementation, measurement, and

analysis in an internet streaming video testbed. In the Proceedings of ACM SIGMETRICS 2001, pp. 312-313.

[CFMB98] Y. Chawathe, S. A. Fink, S. McCanne, E. A. Brewer. A proxy architecture for reliable multicast in heterogeneous environments. In ACM Multimedia, pages 151-159, 1998.

[CHV99] Y. Cai, K. A. Hua, and K. Vu. Optimizing Patching Performance. In the Proceedings of ACM/SPIE Conf. on Multimedia Computing and Networking, San Jose, CA, January, 1999, pp. 204-215.

[CL99] S. W. Carter and D. D. E. Long. Improving bandwidth efficiency of video-on-demand servers. In the Proceedings of Computer Networks and ISDN Systems, pp. 99-111, March, 1999.

[DF98] Judith S. Dahman, Richard Fujimoto, Richard M. Weatherly. The DoD High Level Architecture: an update. Winter Simulation Conference 1998. pp. 797-804

[DHT04] Tai Do, Kien A. Hua, and Mounir A. Tantaoui. P^2VOD: Providing Fault-Tolerant Video on Demand Streaming in Peer-to-Peer Environment. To appear in the Proceedings of the IEEE International Conference on Communications, Paris 20-24 June 2004.

[DLLKB00] C. Diot, B. N. Levine, B. Lyles, H. Kassem, and D. Balensiefen. Deployment issues for the IP multicast service and architecture. IEEE Network magazine special issue on Multicasting, vol. 14, 2000, pp. 78-88.

[DS94] A. Dan and P. Shahabuddin. Scheduling Policies for an on-Demand Video Server with Batching. In the Proceedings of ACM Multimedia'94, San Francisco, CA, USA, October, 1994, pp. 15-23.

[DSS96] A. Dan, D. Sitaram, and P. Shahabuddin. Scheduling policies for on demand video server. In Multimedia Systems. 4(3): 112-121, June 1996.

[EFV99] D. L. Eager, M. C. Ferris, and M. K. Vernon. Optimized Regional Caching for On-Demand Data Delivery. In the Proceedings of the IS&T/SPIE Conference on Multimedia Computing and Networking 1999, San Jose, CA, January, 1999, pp. 301-316.

[EVZ99] D. Eager, M. Vernon, and J.Zahorjan. Optimal and efficient merging schedules for video-on-demand servers. In the Proceedings of ACM Multimedia'99, Orlando, FL, USA, November, 1999, pp. 199-202.

[FCG96] A. Finkelstein, Charles, E. Jacobs, D. H. Salesin. Multi-resolution video. In the

Proceedings of SIGGRAPH 96, in Computer Graphics Proceedings, Annual Conference Series, 281-290, August 1996

[FKMA99a] Z. Fei, I. Kamal, S. Mukherjee, M. Ammar: Providing Interactive Functions For Staggered Multicast Near Video-on Demand Systems. In the Proceedings of the IEEE International Conference on Multimedia Computing and Systems, vol2, 949-953. June 1999.

[FKMA99b] Z. Fei, I. Kamel, S. Mukherjee, and M. Ammar. Providing Interactive Functions Through Active Client Buffer Management in Partitioned Video Broadcast. In the Proceedings of First the International Workshop on Networked Group Communication, (NGC'99) pp. 152-169 Pisa, Italy, Nov. 1999.

[GGUST03] Y. Guo, Z. Ge, B. Urgaonkar, P. Shenoy, and D. Towsley. Dynamic Cache Reconfiguration Strategies for A Cluster-Based Streaming Proxy. In the Proceedings of the International Workshop on Web Content Caching and Distribution, Hawthorne, NY, USA, October, 2003.

[GM96] J. L. L. Golubchik and R. Muntz. Adaptive piggybacking: A novel technique for data sharing in video-on-demand storage servers. In ACM Multimedia Systems, 1996, pp. 140-155.

[GKT98] L. Gao, J. Kurose, and D. Towsley. Efficient Schemes for Broadcasting Popular Videos. In the Proceedings of the International Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV 98), Cambridge, UK, July, 1998.

[GKT02] L. Gao, J. Kurose, and D. Towsley. Efficient Schemes for Broadcasting Popular Videos. In the Proceedings of ACM Multimedia Systems,2002 vol 8 Issue 4 pp. 284-294.

[GSKT03a] Y. Guo, K. Suh, J. Kurose, and D. Towsley. A Peer-to-Peer On-Demand Streaming Service and Its Performance Evaluation. In the Proceedings of the IEEE International Conference on Multimedia and Expo (ICME'03), 2003, pp. 649-652.

[GSKT03b] Y. Guo, K. Suh, J. Kurose, and D. Towsley. P2Cast, P2P Patching Scheme for VoD Service. In the Proceedings of the WWW, 2003, pp. 301-309.

[GST03] Y. Guo, S. Sen, and D. Towsley. Prefix Caching Assisted Periodic Broadcast for Streaming Popular Videos. In the Proceedings of the IEEE International Conference on Communications, Anchorage, Alaska, USA, May, 2003, pp. 2607-2612.

[HCS98] K. A. Hua, Y. Cai, and S. Sheu. Exploiting Client Bandwidth for More Ffficient Video

Broadcast. In the Proceedings of the International Conference on Computer Communications and Networks, October 1998, pp. 848-856.

[HCS98b] K. A. Hua, Y. Cai, and S. Sheu. Patching: A Multicast Technique for True Video-on-Demand Services. In the Proceedings of ACM Multimedia'98, Bristol, UK, September, 1998, pp. 191-200.

[HCS01] K. A. Hua and Y. Cai. S. Sheu. Leverage client bandwidth to improve service latency in distributed multimedia applications. In Journal of Applied Systems Studies, Vol.2 No. 3, pp. 686-704, 2001.

[HKLLR01] G. B. Horn, P. Knudsgaard, S. B. Lassen, M. Luby, and J. E. Rasmussen. A Scalable and Reliable Paradigm for Media on Demand. In the Proceedings of *IEEE Computer*, 2001, pp. 40-45.

[HO02] K. A. Hua, J.-H. Oh, and K. Vu. An Adaptive Video Multicast Scheme for Varying Workloads. In *ACM-Springer Multimedia Systems Journal*, vol. 8, pp. 258-269, August, 2002.

[HS97] K. A. Hua and S. Sheu. Skyscraper Broadcasting: A New Broadcasting Scheme for Metropolitan Video-on-Demand Systems. In the Proceedings of ACM SIGCOMM'97, Cannes, France, September, 1997, pp. 89-99.

[HT03] Kien A. Hua, M. Tantaoui. Cost Effective and Scalable Video Streaming Techniques. In Borko Furht, Oge Marques (eds.): Handbook of Video Databases: Design and Applications in Borko Furht and Oge Marquez in CRC Press, 2002/2003.

[HTT04] K.Hua, Mounir Tantaoui, and Wallapak Tavanapong. Video Delivery Technologies for Large-Scale Deployment of Multimedia Applications. To appear in the Special Issue of the Proceedings of IEEE Evolution of Internet Technologies 2004.

[HTV00] K. A. Hua, D. A. Tran, and R. Villafane. Caching Multicast Protocol for On-Demand Video Delivery. In the Proceedings of ACM/SPIE Conference on Multimedia Computing and Networking (MMCN 2000), San Jose, CA, January, 2000, pp. 2-13.

[HU01] A. Hu. Video-on-Demand Broadcasting Protocols: A Comprehensive Study. In the Proceedings of IEEE INFOCOM'01. Anchorage, Alaska, USA, April, 2001, pp. 508-517.

[HWT97] B. Helfinstine, D. Wilbert, M. Torpey, W. Civinskas .Experiences with Data Distribution Management in Large-Scale Federations. Simulation Interoperability Workshop, paper 032, Fall 2001

[JB02] S. Jin and A. Bestavros. Cache-and-Relay Streaming Media Delivery for Asynchronous Clients. In the Proceedings of the International Workshop of Networked Group Communication NGC'02, Boston, MA, USA, October, 2002.

[JGJKO00] J. Janotti, D. Gifford, K. Johnson, M. Kaashoek, and J. O'Toole. Overcast: Reliable multicasting with an overlay network. In the Proceedings of the 4th Symposium on Operating Systems Design and Implementation, San Diego, CA, USA, 2000, pp. 197-121.

[JT97] L. Juhn and L. Tseng. Harmonic Broadcasting for Video on Demand Services. In the *IEEE Transactions on Broadcasting*, vol. 43, pp. 268-271, September, 1997.

[JT97b] L. Juhn and L. Tseng. Staircase Data Broadcasting and Receiving Scheme for Hot Video Service. In the IEEE Transactions on Consumer Electronics, vol. 43, No. 4 pp. 1110-1117, November, 1997.

[JT98] L. Juhn and L. Tseng. Fast Data Broadcasting and Receiving Scheme for Popular Video Service. In the IEEE Transactions on Broadcasting, vol. 44, No. 1 pp. 100-105, Mars, 1998.

[KHRR01] J. Kangasharju, F. Hartanto, M. Reisslein, and K. W. Ross.Distributing Layered Encoded Video Through Caches. In the Proceedings of the IEEE INFOCOM, Alaska, USA, April, 2001, vol. 3, pp. 1791-1800.

[KS04] T. Kameda and Y. Sun, Optimal truncated-Harmonic windows scheduling for broadcast systems. To appear in ACM SAC04, March 2004, Nicosia, Cyprus.

[LL97] W. Liao and V. O. Li. The Split and Merge (SAM) Protocol for Interactive Video-on-Demand. IEEE Multimedia, vol.4 pp. 51-62, October-December 1997.

[LHM95] Xin Li, Kien A. Hua, and Michael Moshell. Distributed Database Designs and Computation Strategies for Networked Interactive Simulations. Journal of Parallel and Distributed Computing,72-90 (1995)

[LV96] T. D. C. Little and D. Venkatesh. Prospects for interactive video on demand. In The *IEEE JSAC*, vol. 14, pp. 1099-1109, August, 1996.

[LW95] Gary Lilienfield , John Woods. Scalable High-Definition Video Coding. In the Proceedings of the International Conference on Image Processing (ICIP) (vol-2) p-2567, ICIP'95, Oct. 23-26 1995.

[MBDT99] KatherineL.Morse, Lubomir Bic, Michael Delancourt, and Kevin Tsai. Multicast

grouping for Dynamic Data Distribution Management. In the Proceding the 31st Society for Computer Simulation Conference SCSC1999.

[MEVS01] A. Mahanti, D. Eager, M. Vernon, and D. Sundaram-Stukel. Scalable On-Demand Media Streaming with Packet Loss Recovery. In the Proceedings of ACM SIGCOMM 2001, San Diego, CA, USA, August, 2001, pp. 97-108.

[MJV96] S. McCanne, V. Jacobson, M. Vetterli. Receiver-driven layered multicast. In the Proceedings of ACM SIGCOMM'96, pages 117--130, Stanford, CA, August 1996

[MS02] H. Ma and K. G. Shin. Multicast Video-on-Demand Services. In the Proceedings of *ACM* Communication Review, pp. 31-42, January, 2002.

[PA01] Jehan-Francois Paris. A Fixed-Delay Boradcasting Protocol for Video-on-Demand. In the Proceedings of the 10th International Conference on Computer Communications and Networks (ICCN'01) October 2001, pp. 418-423

[PBC00] Y.-W. Park, K.-H. Baek, and K.-D. Chung. Reducing Network Traffic Using Two-Layered Cache Servers for Continuous Media Data on the Internet. In the Proceedings of the IEEE International Conference on Computer Software and Applications, 2000, pp. 389-374.

[PCL98] J.-F. Paris, S. W. Carter, and D. D. E. Long. Efficient Broadcasting Protocols for Video on Demand. In the Proceedings of International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOT'98), July, 1998, pp. 127-132.

[PCL98b] J.-F. Paris, S. W. Carter, and D. D. E. Long. A Low Bandwidth Broadcasting Protocol for Video on Demand. In the Proceedings of the International Conference on Computer Communications and Networks, Lafayette, LA, USA, October, 1998, pp. 690-697.

[PCL98c] J. F. Paris. A simple low-bandwidth broadcasting protocol for video on demand. In the Proceedings of the International Conference on Computer Communications and Networks, 1998, pp. 118-123.

[PCL99] J.-F. Paris, S. W. Carter, and D. D. E. Long. A Hybrid Broadcasting Protocol for Video on Demand. In the Proceedings of ACM/SPIE Multimedia Computing and Networking Conference (MMCN'99), San Jose, CA, January, 1999, pp. 317-326.

[PWC02] V. N. Padmanabhan, H. J. Wang, and P. A. Chou. Distributing Streaming Media Content Using Cooperative Networking. In the Proceedings of the International

Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV'02), May, 2002, pp. 177-186.

[RRG01] S. Ramesh, I. Rhee, and K. Guo. Multicast with Cache (Mcache): An Adaptive Zero-Delay Video-on-Demand Service. In the Proceedings of the IEEE INFOCOM, Alaska, USA, April, 2001, pp. 85-94.

[RSM97] Steven J. Rak, Marnie Salisbury, Robert S. MacDonald. HLA/RTI data distribution management in the synthetic theater of war. Simulation Interoperability Workshop, paper 119, Fall 1997.

[SDKST97] Subhabrata Sen, Jayanta Dey, Jim Kurose, John Stankovic, and Don Towsley. Streaming CBR transmission of VBR stored Video. In the Proceedings of the SPIE Symposium on Voice Video and Data Communications (Dallas, TX, November 1997). Pp. 26-36.

[SHT97] S. Sheu, K. A. Hua, and W. Tavanapong. Chaining: A Generalized Batching Technique for Video-on-Demand Systems. In the Proceedings of IEEE ICMCS'97, Ottawa, CA, 1997, pp. 110-117.

[SGT01] S. Sen, Lixin Gao, and Don Towsley. Frame-based periodic broadcast and fundamental resource tradeoffs. In the Proceedings of the IEEE International Performance, Computing, and Communications Conf. (IPCCC 2001), April 2001.

[SRT99] S. Sen, J. Rexford, and D. Towsley. Proxy Prefix Caching for Multimedia Streams. In the Proceedings of the IEEE INFOCOM'99, New York, NY, 1999, pp. 1310-1319.

[SRT99b] S. Sen, J. Rexford, and D. Towsley. Optimal patching schemes for efficient multimedia streaming. In the Proceedings of the IEEE NOSSDAV'99, June, 1999, and Technical Report 99-22, University of Massachussetts

[THD02] D. A. Tran, K. A. Hua, and T. T. Do. Scalable Media Streaming in Large Peer-to-Peer Networks. In the Proceedings of ACM Multimedia, Juan-les-pins, France, December, 2002, pp. 247-250.

[THD04] Mounir Tantaoui, K. Hua, and T. Do. BroadCatch: A Periodic Broadcast Technique For Heterogeneous Video-on-Demand. To appear in The IEEE Transactions on Broadcasting, 2004.

[THJ03] Mounir Tantaoui, K. Hua, N. Jiang. A Limitless Infrastructure for Next-Generation Large Scale Simulation and Training Systems. In the Proceedings of the Advanced Simulation Technologies Conference, (ASTC 2003) Military, Government, and

Aerospace Symposium, Pages(s): 86-93, Orlando, FL March 30th – April 3rd.

[THS02] Mounir Tantaoui, K. Hua, S. Sheu. Interaction with Broadcast Video. In the Proceedings of the 10th ACM International Multimedia Conference, ACM Multimedia 2002, Page(s): 29-38. Juan-les-Pins, France December 1st – 6th 2002.

[THS02b] Mounir Tantaoui K. Hua, S. Sheu. A Broadcast Technique for Providing Better VCR-like Interactions in a Periodic Broadcast Environment. In the Proceedings of the Global Telecommunications Conference 2002, Globecom'02, IEEE. Volume: 2, Page(s): 1648 – 1652. Taipei, Taiwan, November 17th – 21st.

[THT02] D. Tran, K. Hua, M. Tantaoui. A Multi- Multicast Sharing Technique for large-scale Video Information Systems. In the Proceedings of the International Conference on Communications, ICC'02, IEEE. Page(s): 2496-2502 vol 4. New York, New York. April 28th – May 2nd 2002.

[TYHLS01] Y. C. Tseng, M. H. Yang, C. M. Hsieh, W. H. Liao, and J. P. Sheu. Data Broadcasting and Seamless Channel Transition for Highly Demanded Videos. In IEEE Transactions on Communications, vol 49, No. 5, May 2001.

[TPL03] K. Thirumalai, J.-F. Paris, and D. D. E. Long. Tabbycat: An Inexpensive Scalable Server for Video-on-Demand. In the Proceedings of the IEEE International Conference on Communications, Anchorage, AK, USA, May, 2003, pp. 896-900.

[TPTW04] M. Tran, W. Putthividhya, W. Tavanapong, and J. Wong. A Case for a Generalized Periodic Broadcast Server: Design, Analysis, and Implementation. In the Proceedings of the IEEE International Symposium on Applications and the Internet (SAINT 2004) , Tokyo, Japan, 2004, pp. 127-134.

[VC98] D. Van Hook, J. Calvin. Data Distribution Management in RTI 1.3. Simulation Interoperability Workshop, paper 206*, Spring 1998*

[VI96] S. Viswanathan and T. Imielinski. Metropolitan Area Video-on-Demand Service using Pyramid Broadcasting. In *Multimedia Systems*, vol. 4, pp. 179-208, August, 1996.

[XHB02] M. H. D. Xu, S. Hambrush, and b. Bhargava. On Peer-to-Peer Media Streaming. In the Proceedings of the IEEE International Conference on Distributed Computing and Systems, July, 2002, pp. 363-371.

[YK03] Edward M. Yan and Tiko Kameda. An Efficient VOD Broadcasting Scheme with User Bandwidth Limit. In the Proceedings of the ACM Multimedia Computing and Networking, vol 5019, Santa Clara, CA 20-24 January 2003.

[ZWDS00] Z.-L. Zhang, Y. Wang, D. H. C. Du, and D. Su. Video Staging: A Proxy-Server-Based Approach to End-to-End Video Delivery over Wide-Area Networks. IEEE/ACM Transaction on Networking, vol. 8, pp. 429-442, August, 2000.