NETWORK ARCHITECTURE IN A LARGE-SCALE FULLY INTERACTIVE VOD SYSTEM BASED ON HYBRID MULTICAST-UNICAST STREAMING

CHAN Kwun-chung

A THESIS

SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR THE DEGREE OF MASTER OF PHILOSOPHY

IN

INFORMATION ENGINEERING

AUGUST 2001

摘要

我們已經提出了一個新穎的大規模完全交互的視頻點播系統，它是一個基於混合型多點和單點資料流傳送的分散式結構。在這個結構中，中央伺服器族能夠用批次處理的多點傳送傳輸方式處理沒有數目限制的正常播放客戶。而分散式交互伺服器用單點傳送方式來處理本地的交互請求。在用戶端的緩衝管理和資料匯流操作等重要特徵也是使這個視頻點播能成爲一個大規模，低價格而完全交互的解決方案。

在這個結構中，交互伺服器決定了整個系統的性能。因此，我們分析交互伺服器的性能。它的性能取決於（一）在交互功能後，匯流資料流的加速率，（二）用戶交互所持續的時間，（三）在相鄰多點傳送的資料流的間隔。我們研究了這些參數的不同組合在系統容量的影響。它顯示對於不同的視頻內容和不同地區的用戶習慣可以優化交互伺服器的性能，並且這些參數以有趣的方式來決定系統的性能。

爲了進一步提高系統的可升級性和反應時間，我們提議了用一個視頻代理伺服器高速緩衝視頻片段來減低骨幹網路和中央伺服器的負擔。不象傳統的替換演算法，我們提議的替換演算法考慮了包括普及程度估計，點播頻率和最近視頻點播的時間。此外，我們介紹了一個兩階段的替換策略來最小化啓動的等待時間。

Abstract

We have proposed a novel large-scale fully interactive VOD system. It is a distributed architecture based on hybrid multicast-unicast streaming. In this architecture, the central server cluster is able to serve nearly unlimited number of normal play clients by batched multicast transmission. And the distributed interactive servers serve the interactive requests from its local region by unicast streams. The buffer management at the client stations and the stream merging operation are also the important features to make this VOD system to be a large-scale, cost effective and fully interactive solution.

In this architecture, the performance of the whole system depends on that of the interactive servers. Thus, the performance of the interactive server is analyzed. It is found that its performance depends on (i) speed-up-ratio of the merging stream that brings a client back to multicast stream after interactive functions, (ii) time duration of user interaction, (iii) stream interval between adjacent multicast streams. The effect of different combination of these parameters on the system capacity is studied. It is shown that the performance of the interactive server can be optimized with respect to different types of video content and the user behavior in different regions. And the system performance depends on these parameters in an interesting way.

In order to further improve the scalability and response time, we have proposed a video proxy system that aims at reducing the backbone network and central server loading by caching the video clips in the video proxy servers. Unlike the traditional replacement algorithms, the proposed replacement algorithm will consider the combination of the estimated popularity, access frequency and recency of a video clip. Moreover, a two-stage replacement policy is introduced to minimize the start-up latency.

Acknowledgement

First of all, I would like to express my sincerest gratitude to my supervisor, Prof. Kwok-wai Cheung, for his guidance, insights, suggestions, constructive criticisms, sustained encouragement and support.

I am also grateful to thank Mr. Baron Ng, Raymond Chan, Fox Tam and Calvin Chan, who worked closely with me in my research, for their guidance and support

Finally, I would like to thank my family for their love and endless support.

Table of Contents

List of Figures

List of Symbols

T_{Fill}:	The time to fill up the buffer of CS
τ_{Play}:	The time at the user resumes the normal play back
τ_{Full}:	The time at the buffer full
T_{FF} :	The time for the fast forward action that served by the I-stream
T_{REW}:	The time for fast backward action that served by the I-stream
Λ_{Leave}:	The time of the frame being delivered by the originally multicast stream when the CS leaves it
Λ_{Full}:	The time of the frame at the end of the buffer after J-stream fills it up.
r:	The data delivery rate of the multicast stream and I-stream
s:	The speed-up-ratio of J-stream
P_{PAU}:	The probability for an interactive request to be pause
P_{SM}:	The probability for an interactive request to be slow motion
P_{FF}:	The probability for an interactive request to be fast forward
P_{REW}:	The probability for an interactive request to be fast backward
P_{JF}:	The probability for an interactive request to be jump forward
P_{JB}:	The probability for an interactive request to be jump backward
$T_{FF/REW}$:	The time duration of fast forward or backward

R:	The average data delivery rate from DIS for each user interaction
B:	The bandwidth of the DIS allocated to the interactive functions
n:	The maximum number of concurrent streams DIS can support
λ :	The average requests arrival rate
μ :	The average service rate
P_{BL} :	The blocking probability
$P_i(t)$:	The popularity function of a cached content i at time t
E_i:	The estimated popularity of a video clip i
N_i:	The number of times video clip i is accessed within a period
N_{total}:	The total number of video clips accessed within a period
t_{last}:	The time that video clip i was last accessed
α:	The bias value to the video clip's accessed frequency
β:	The bias value to the video clip's accessed recency
S_i:	The minimum amount of an video object prefix to be stored
T_{DPS}:	The transmission delay between DPSs
T_{UVSC}:	The transmission delay between the DPS and UVSC
BRR:	The Bandwidth Reduction Ratio
BW_{cached}:	The bandwidth consumed by the cached object in DPS
BW_{server}:	The bandwidth consumed by the un-cached object that needed to be delivered from UVSC

Chapter 1

Introduction

Audio and video streaming are becoming more and more important in the Internet. And Video-on-Demand (VOD) service is one of the most feasible multimedia streaming service. In a true VOD system, users are allowed to view any video programs at any time and perform any VCR-like interactive functions, such as fast forward, fast rewind, jump forward, jump rewind, slow motion and pause.

In a unicast VOD system, it can be easily achieved because the system will dedicate a channel to each user and handle each user independently. However, the resource needed to operate such a system is proportional to the number of users. Thus, this method is very expensive, inefficient and not scalable.

Thus, multicast delivery is regarded as one of the solutions to reduce the cost and increase the scalability of a large-scale VOD system. In a multicast VOD system, the multicast streams can be shared by nearly unlimited number of normal play users. Unfortunately, it is difficult to implement interactive functions in this type of system because the server cannot serve one user's interactive request by modifying the multicast stream. It is a challenging and hot topic in recent years as to find out how to satisfy one user's interactive requests without affecting other users in the same multicast group.

In addition, a large-scale Video-on-Demand (VOD) system needs to support a large number of users. Thus, there are heavy demands on the central server and backbone bandwidth. The backbone bandwidth cost is still much higher than the local area network bandwidth cost. Therefore, the caching of video content at the local distribution point is a possible solution. It utilizes the local area network bandwidth and disk space in the video proxy server to reduce the backbone network and central server loading. Moreover, the start-up latency can be reduced, as the proxy servers are closer to the clients.

1.1 Contributions

This thesis has two main contributions. First, we propose a novel large-scale fully interactive VOD system. It is a distributed architecture based on hybrid multicast-unicast streaming. In this architecture, the central server cluster is able to serve nearly unlimited number of normal play clients by batched multicast transmission. And the distributed interactive server is responsible for serving the interactive requests from its local region. This arrangement has the advantages of increasing scalability, reducing central multicast server loading and improving response time. The buffer management at the client stations and the stream merging operation are also the important features to make this VOD system to be a large-scale, cost effective and fully interactive solution. We also analyze the system performance and show that it can be optimized with respect to different types of video content and the user behavior in different regions.

Second, we propose a video proxy system that aims at reducing the backbone network and central server loading by caching the video clips in the video proxy servers. Unlike the traditional replacement algorithms, the proposed replacement algorithm considers the combination of the estimated popularity, access frequency and recency of a video clip with the use of a priority function. Moreover, a two-stage replacement policy is introduced to minimize the start-up latency.

1.2 Organization of the Thesis

This thesis comprises five chapters. Chapter 2 gives a brief overview on the previous research on VOD system and caching algorithm. Different kinds of service model and architecture will be discussed. It also shows how the interactive functions are done in the previous research. In addition, different types of caching algorithm are discussed.

Chapter 3 describes the design details of our VOD system. We first show the system architecture of our VOD system and explain why it is highly scalable. Then, we discuss how the system provides full interactivity. Finally, the performance results are presented. We also discuss how the performance can be optimized.

In chapter 4, we discuss the design details of our video proxy system. The priority function, replacement and caching policy will be described and explained. Then, we will present the performance results.

Finally, we will conclude in chapter 5.

1.3 Publications

The following lists the research papers written during the course of this study:

1. C.H. Ng, K.C. Chan, K.W. Chan, C.H. Chan, C.Y. Tam and K.W. Cheung, "DINA – System for a Large-scale Fully interactive VOD System based on Hybrid Multicast-unicast Streaming", IEEE-PCM'2000, Dec 2000

2. K.C. Chan, K.W. Cheung, "Performance Analysis on Distributed Interactive Server in a Large-scale Fully Interactive VOD System", IEEE-ICOIN-15, Feb 2001

3. K.C. Chan, K.W. Cheung, "Video Proxy System for a large-scale VOD System", 5th WSES/IEEE CSCC2001, July 2001

4. A patent, which titled "Method and System for Delivering Media Selections through a Network", is pending.

Chapter 2

Related Works

This chapter reviews the previous work on VOD system and caching system. Chapter 2.1 describes the previous VOD system. The common service model and architecture will be discussed in chapter 2.1.1 and 2.1.2 respectively. Chapter 2.1.3 reviews the limited and unlimited interactive functions working mechanisms in previous research. In chapter 2.1.4, previous split and merge operation will be described. Chapter 2.2 reviews the traditional caching algorithm. Chapter 2.2.1 and 2.2.2 describes two well known caching algorithm, namely LFU and LRU. Then, an overview on previous media stream caching algorithm will be given in chapter 2.2.3.

2.1 Previous VOD System

2.1.1 Service Model

2.1.1.1 Unicast VOD

In a unicast VOD system, a dedicated channel is allocated to each user [1]. The relationship between the resource and user is in a one-to-one basis. When a user makes a request to the video server, the request will be satisfied if the server and network resources are available. Then, the server transmits the requested video content to the user immediately.

Since the user occupies his own resource and the server handles each client independently, the user's interactive functions can be satisfied easily. It is the advantage of a unicast VOD system. However, as there is a one-to-one relationship between the resources and customer requests, the system does not scale well.

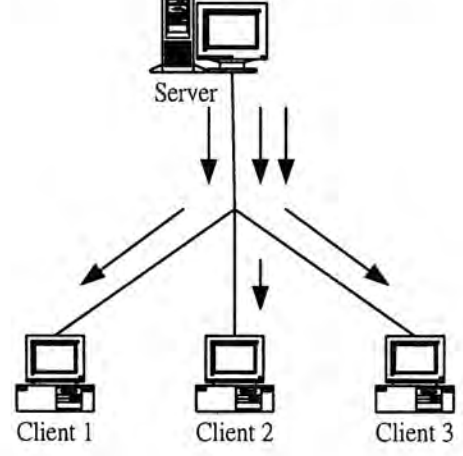


Figure 1. Unicast VOD System

2.1.1.2 Multicast VOD

Multicast transmission is an efficient way to deliver data to multiple users at the same time. It saves both the server and network resources. In a multicast VOD system, batching is commonly used to transmit video stream at a fixed time interval [2]. When a user makes a request to the server, the request will be satisfied in the coming stream delivery. Thus, all user requests within a batching interval can be served by only one multicast stream.

A multicast VOD system can serve nearly unlimited number of users with a fixed resource, so it is highly scalable [3], [4]. However, in this type of system, the user cannot get the video content immediately. Also, it is difficult to provide interactivity because the server cannot serve one user's interactive request by modifying the stream. Otherwise, other users in the same multicast stream will be affected. Thus, only limited interactivity is provided in this type of system. In order to provide full interactivity, additional resources are needed [3].

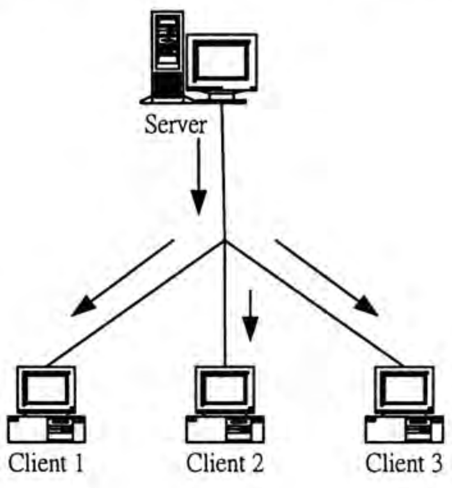


Figure 2. Multicast VOD System

2.1.2 Architecture

2.1.2.1 Centralized Architecture

In a centralized architecture, there is a central server or server cluster that is responsible for storing all video contents and handling all user requests. This type of system has the advantages of simple management. However, if there is a long distance between the server and clients, the clients may experience a large delay. Also, in this type of architecture, the central server and backbone network loading will be very heavy when there is a large amount of users. Thus, this architecture is not well scaled.

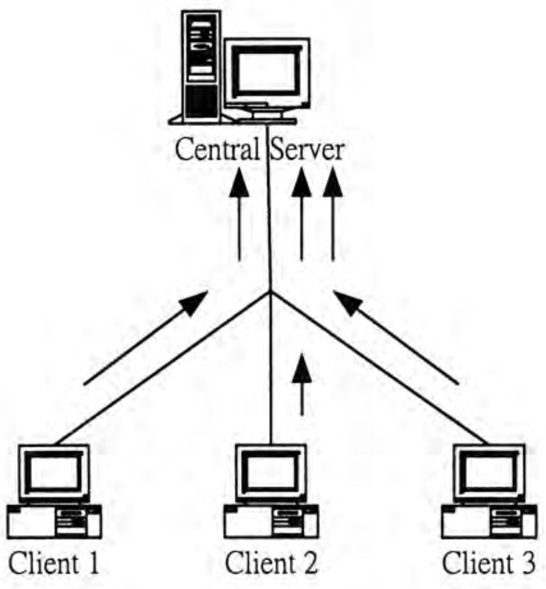


Figure 3. Centralized Architecture

2.1.2.2 Distributed Architecture

A distributed architecture is usually composed of both central server and local servers [5]. The central server acts as a video library and stores all the video contents, while the local server stores a portion of the contents. When a user makes a request, the local server will check whether it has the requested video content and resources. If yes, it will serve the user. Otherwise, the request will be forwarded to the central server. Since some of the requests can be handled by the local servers, the central server and backbone network loading can be reduced. Also, the response time is improved. However, the management of this kind of system is more complex.

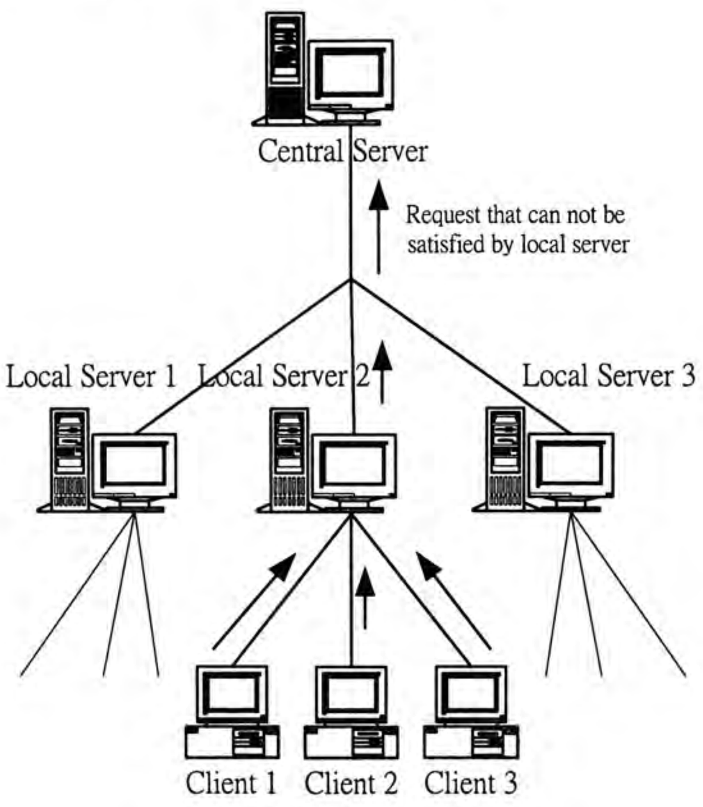


Figure 4. Distributed Architecture

2.1.3 Interactive Function

2.1.3.1 Limited Interactive Function

In a pure multicast VOD system with batched stream delivery, only limited interactivity can be provided. The amount of interactivity depends on the buffer size [3]. For example, when a user issues fast forward action, it can be achieved by skipping the frames that are stored in the buffer. After all the frames in the buffer are consumed, the user can only experience discontinuous interaction. For instance, if the time interval between the batched multicast streams is 30 sec, the user can only perform the fast forward action with a step of 30sec. The client jumps to the previous multicast stream in order to get the frames for the later part of the video. It is easy and simple to provide this type of interaction. However, it may not be able to meet the requirement of the users.

2.1.3.2 Unlimited Interactive Function

Unlimited interactive function means the VCR-like interactive function. It is continuous and does not have bound. For example, the user can perform continuous fast forward action till the end of the video content. As mentioned in chapter 2.1.1, a unicast VOD system can easily provides this kind of interaction. However, for a multicast VOD system, additional resources are necessary. Ref. [3], [4], [6]-[8]

suggests that the system creates a new unicast stream to handle the users interactive requests.

Take fast forward as an example, in Ref. [7], it is suggested that the server serves the request by creating a new unicast stream with higher transmission rate. Then the client plays the video content with some of the frames skipped.

2.1.4 Split and Merge Operation

Creating unicast streams to handle the user interactive requests is regarded as a solution for providing unlimited interactivity in a multicast VOD system. However, in this approach, the interactive user may eventually hold the unicast stream till the end of the video. Then, some users' interactive requests may not be able to be served when there is not enough resource for creating a new unicast stream. It will also reduce the scalability of the system. Thus, some mechanisms are needed to bring a user back to multicast stream after the requested interaction is finished.

2.1.4.1 SAM Scheme (Split and Merge)

The SAM protocol makes use of a shared buffer at the access node to allow a unicast stream to merge back to the multicast stream [4]. The merge operation depends on whether the buffer size is large enough to store the frames between the client play-point (the frame that the client is currently playing) and the play-time (the frame that is being delivered by the multicast stream) of the target multicast stream. In a batched multicast transmission, the shared buffer should be large enough to hold the frames within a stream interval. Otherwise, a client may not be able to rejoin a multicast stream.

Figures 5 illustrates how SAM works. Suppose after an interactive function, the client resumes normal play and begins to play the 110sec frame. At this time, a unicast stream that begins with 110sec frame is created to serve the normal play. Meanwhile, a suitable multicast stream needs to be found so that the client can merge back to a multicast stream as soon as possible. In this example, the (k)-th multicast stream has the least offset to the current play-point (110sec). So the shared buffer at the access node starts to receive frames from the (k)-th multicast stream. Then, the frames after 120sec are available in the shared buffer. Thus, the unicast stream can be freed after 10 seconds because the client can continue normal play with the frames in the shared buffer. At this time, the client merges back to the multicast stream successfully.

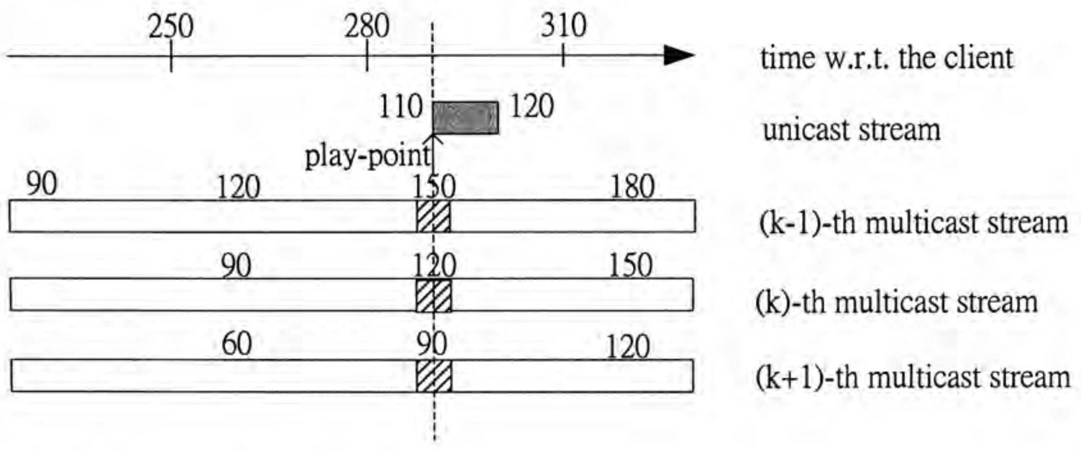


Figure 5. Illustration of SAM Scheme

In this scheme, since most of the job is done is the shared buffer, the design of the client set-top-box can be simpler. However, since a shared buffer at the access node is shared by many users. Failure of it will affect many users.

2.1.4.2 SRMDRU Scheme (Single Rate Multicast Double Rate Unicast)

In Ref. [8], the client buffer is used for merging. This is easier to manage and more scalable as no separate device is needed. Similar to SAM, the operation also depends on whether the buffer has stored all the frames between the client play-point and the targeted stream's actual play-time. The client buffer should be large enough to hold the frames within a stream interval.

This scheme suggests that the server delivers a double rate unicast stream that begins with the required frame to the client after the user interaction. During double rate transmission, since the frames incoming rate is higher than the frames play rate, the unplayed frames accumulates and eventually fills up the buffer. Once the buffer is filled up, the client is able to merge back to a multicast stream. The detailed description about the merging mechanism after the buffer has been filled up with unplayed frames can be found in chapter 3.3.

2.2 Previous Caching Algorithm

2.2.1 LFU (Least Frequently Used)

LFU measures the frequency of the object being referenced within a period [9]. It assumes that the frequently accessed object is more likely to be re-accessed in the future. Thus, the least frequently referenced object will be removed when there is not enough space.

2.2.2 LRU (Least Recently Used)

In LRU, the history of the access time is maintained for each object [10]. LRU only maintains and considers the time of the last access, while the LRU-k makes use of the last k times of access. It assumes that the more recently accessed object is more probably to be accessed again in the future. Thus, the least recently accessed object will be replaced when there is not enough space.

2.2.3 Media Stream Caching

In addition to the above two well known caching algorithms, many researches on the caching of media stream have been performed recently [11]-[16]. Ref. [11] suggests a prefix caching technique to store the initial frames of popular clips. It performs workahead smoothing to reduce the peak bandwidth and the burstiness of the variable-bit-rate stream. In Ref. [12], it is suggested that a media object should be partitioned into two parts. Usually, the front-end partition is stored in the cache closer to the client. The rear-end partition is stored in the cache away from the client and will be transferred to the cache if the media is accessed frequently. Ref. [13] suggests that the amount of object stored in the cache should be proportional to the popularity of that media. It is to reduce the replacement overhead and increase the byte hit rate.

Chapter 3

Design of a Novel VOD System

In this chapter, we discuss the design of a large-scale fully interactive VOD system. In chapter 3.1, we will give an overview of the system architecture. The features and functions of each component will be described. Chapter 3.2 explains the batched multicast transmission that is employed in the central server. The split and merge operation between multicast and unicast stream is described in chapter 3.3. It is one of the keys that make this VOD system highly scalable. Chapter 3.4 describes how the interactive functions can be achieved. Finally, we will discuss how to optimize the performance of the VOD system in chapter 3.5

3.1 System Architecture

The architecture shown in Figure 6 is a hybrid scheme based on multicast-unicast streaming which provides a VOD system with full interactivity and high scalability [6]. This system is designed for metropolitan area with millions of users. It allows users to perform any interactive functions (e.g. Pause, Slow Motion, Fast Forward/Backward and Jump Forward/Backward) in an unrestricted manner. In addition, the system only requires a modest amount of system resources to support a large number of users.

In this system, three types of streams are defined. The co-operation of these streams is the key to make this VOD system fully interactive and highly scalable.

1) M(i)-stream – a multicast stream for normal play starting at the beginning of the i-th stream interval.
2) I-stream – a unicast stream that transmits at normal data rate (e.g. 1.5Mbps for MPEG-1) and opened for serving interactive function. It is responsible for delivering the pre-processed video contents. For example, it transfers the video content that is pre-recorded as the double speed of the original contents.
3) J-stream – a unicast stream that transmits at higher data rate (say, 1.5 to 4 times the normal data rate) and allows a user to merge back to a multicast stream.

This system is composed of seven main components: Multicast Video Server Cluster (MVSC), Unicast Video Server Cluster (UVSC), Client Stations (CS), Multicast Backbone Network (MBN), Local Distribution Network (LDN), Distributed Interactive Server (DIS) and Distributed Proxy Server (DPS). A brief overview of these components is as follows,

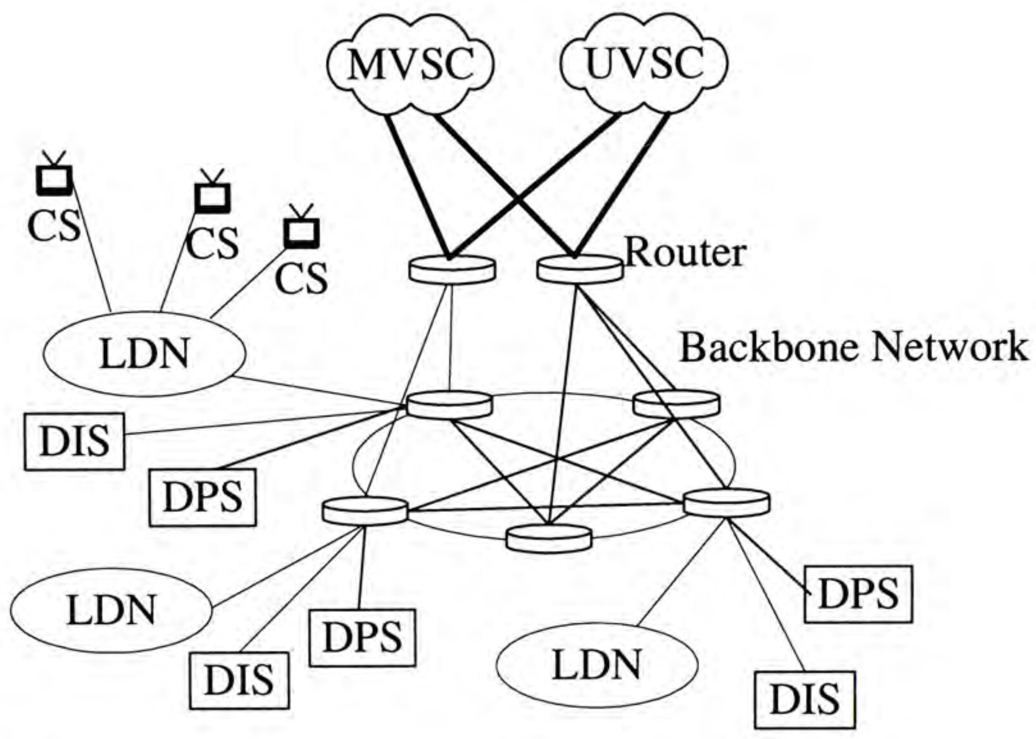


Figure 6. System Architecture

Note that the Multicast Backbone can use any arbitrary topology that supports the multicast protocol. The structure shown in the Figure is used for simplicity.

3.1.1 Multicast Video Server Cluster (MVSC)

MVSC is responsible for storing and delivering the hot video contents. Batching [2] is deployed to generate multicast stream (M(i)-stream) at each fixed batch interval in a round-robin manner, say around 30 - 60 sec. The hot contents are delivered by the multicast streams because they are expected to be viewed by many users at the same time. Since one multicast stream can support all the user requests within a batching interval, the backbone bandwidth loading can be reduced significantly.

Each MVSC consists of a cluster of servers (e.g. 5 to 15 video servers), and each server stores part of the video content in a simple striped format with parity added for forward error correction. Thus, a simple error recovery is possible if one video block of the parity group is lost. Moreover, the server cluster is still functional if some of the servers are down.

3.1.2 Unicast Video Server Cluster (UVSC)

UVSC stores and delivers the non-hot video contents using unicast streams. It acts as a video library to store a large amount of video contents. When a user requests a non-hot content and the DPSs do not have it, UVSC will deliver that content to the user. The configuration and management is similar to that of MVSC.

3.1.3 Multicast Backbone Network (MBN)

MBN carries the video streams from MVSC and UVSC to LDN. It comprises the high-speed routers that are capable of running multicast routing protocol such as PIM-DM, PIM-SM, DVMRP, MOSPF, etc.

3.1.4 Local Distribution Network (LDN)

LDN carries the video streams from MVSC, UVSC and DIS to each CS. The LDN should support multicast routing protocol and layer 2 IGMP snooping.

3.1.5 Distributed Interactive Server (DIS)

DIS is responsible for providing interactive functions and error recovery to CS. All the hot contents are cached in the DIS, so that DIS can serve the interactive and retransmission requests from the CS in its local region without intervening the MVSC. It also stores the video pre-recorded at different speeds for both forward and reverse direction. These pre-recorded contents are used for serving the fast forward and backward requests.

To serve the interactive requests, it provides two kinds of stream – I-stream and J-stream. I-stream is to serve the fast forward or backward interaction by delivering the pre-recorded stream. J-stream is to bring a user back to multicast stream after the user interaction is finished. For error recovery, DIS provides a unicast or multicast stream to retransmit the missing packets to the CS.

3.1.6 Distributed Proxy Server (DPS)

DPS is to cache the non-hot contents that are frequently accessed. The replacement policy will choose the higher priority contents and cache them in the proxy server. These contents are expected to be re-accessed in the near future, so that the DPS can serve the future requests without accessing the UVSC or consuming backbone bandwidth. In order to utilize the disk space more efficiently and achieve higher bandwidth reduction, the cached contents in one DPS can be shared with other DPSs. In addition, it is also responsible for the retransmission of missing packet to the client that it is currently serving.

3.1.7 Client Station (CS)

CS receives video stream from the network and then plays it at the display equipment of the client. It has a buffer that can hold all the frames within one stream interval. This buffer size requirement is necessary for CS to be able to rejoin a multicast stream after performing interactive functions. In addition, CS should have a broadband network connection that can support about 1.5 to 4 times of the video transmission rate.

When a video block is missed or corrupted, the CS will try to recover it from the parity information. If it cannot reconstruct the missing block or multiple blocks are missing, it will request the DIS or DPS to retransmit the missing packets.

In this architecture, the interactive functions are provided by the Distributed Interactive Servers (DIS). It has the advantages of increasing scalability, reducing central multicast server loading and improving response time. Nearly all the normal-play users can be served by the multicast servers. However, one single centralized interactive server cannot serve all the users because each user requires a dedicated channel. Thus, a distributed approach is more preferable. More DISs can be installed in each local distribution region in scaling with the growing number of users. Therefore, high scalability, capacity and flexibility can be achieved with this approach. In addition, the central multicast server loading can be reduced, as it only needs to provide the multicast streams and does not need to take care of the interactive requests. This makes the multicast delivery more reliable. Moreover, since DIS is distributed and closer to the CS, the packet delay is minimized and so this would improve the response time of the interactive functions and the success rate of retransmission.

3.2 Batched Multicast Transmission

Batching [2] has been proposed to reduce the cost of VOD by grouping users together so that they can share the same video stream at the same time. This scheme allows one multicast stream to serve all the user requests arrived in a stream interval. Both the server and network loading can be reduced.

Figure 7 shows our batching system. Multicast stream will start to transmit at the beginning of each stream interval no matter whether there are user requests arrived within that stream interval or not. It is because the multicast streams are not only responsible for serving the start-up user requests, but also prepare for any user merging back to the multicast stream after user interaction. This arrangement makes the multicast transmission simpler and more reliable.

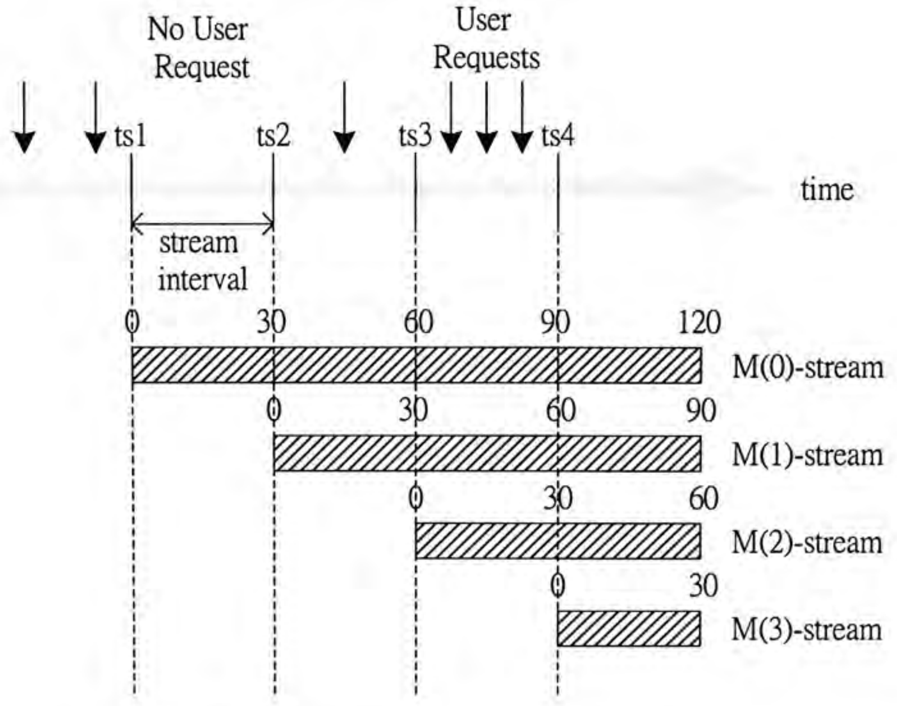


Figure 7. Illustration of Batched Multicast Transmission

In our batching system, the stream interval is set at around 30 - 60 seconds. Thus, the average waiting time is about 15-30 seconds that is acceptable. It may be argued that the stream interval is too short and a large number of multicast streams are needed. However, it is worthwhile to do so as it can reduce the buffer requirement at the CS because the buffer in the CS must be able to hold up all the frames within a stream interval. Moreover, with a short stream interval, it can reduce the holding time of the J-stream required for merging. Thus, more interactive requests can be satisfied.

3.3 Split and Merge Operation

When a user engages in an interactive function, a new unicast stream will be provided to the user according to the request, and thus client splits from the original multicast stream. The splitting operation is straightforward as only a new unicast stream is provided to handle the requested interaction.

On the other hand, the merge operation is much more complicated and is extremely important as it allows a client on a unicast stream to merge back to a multicast stream and release the I-stream and J-stream after performing the interactive function. A great improvement in the VOD interactive feature is achieved because the merging reduces the number of unicast streams. It in turns allows the same number of streams to serve more users' interactive request, thus better quality and scalability are achieved.

Our VOD system can ensure that all the clients can merge back to a multicast stream provided that the following requirements are met:

1) the CS buffer is large enough to hold all the frames within one stream interval.
2) the J-stream can transmit at a higher rate than the multicast streams and therefore can fill up the necessary buffer before merging.

This scheme is an enhancement from the SRMDRU that is mentioned in chapter 2.4.2, the transmission rate proposed here is not restricted to the double speed [8]. The J-stream is opened as soon as a user resumes normal play. Let the data delivery rate of the I-stream be r (say 1.5Mbps for MPEG-1), the data delivery rate of J-stream be $s \bullet r$ where s is the speed-up-ratio (say, 1.5 to 4X), Since the frames incoming data rate is faster than the frames consumption rate r, the buffer will be filled up. The speed-up-ratio s can be chosen with different values according to the network architecture. Network with more bandwidth can support a larger s, which means that the client can merge back to the multicast stream in a shorter time.

The buffer is being filled up at a rate of $(s-1)r$ Mbps. Thus, the required time to fill up the buffer T_{Fill} is,

$$T_{Fill} = \frac{buffer\ size}{(s-1)r}\ sec \tag{1}$$

$where\ \ buffer\ \ size = stream\ \ interval \times r\ bit$

Once the buffer is filled up, the client must be capable of merging back to a multicast stream. An example is shown in Figure 8. Suppose that after some interactive action and J-stream transmission, the buffer has been filled up at an arbitrary time mark 280sec relative to the CS. The current play-point is at 90sec of the video content so the CS buffer stores the frames from 90sec to 120sec of the content. Then, the CS leaves the J-stream as no more data can be stored, thus freeing up a J-stream to serve other users.

Since the stream interval between the multicast streams is the same as the CS buffer size, an multicast stream delivering frames that already in the CS's buffer can always be found. For example, at time mark of 280sec, the play-time of the M(k)-stream is 110sec. And this frame is already in the CS buffer. CS needs to continue to play frames beyond what have been buffered, so it needs to merge back to the appropriate multicast stream. To achieve this, it can start to receive frames at 120sec of the M(k)-stream (i.e. at time mark 290sec relative to the CS). At that time, 10 seconds of buffer space in CS is freed as the frames from 90sec to 100sec have already been played. Thus, CS is able to accept the frames after 120sec of M(k)-stream and is successfully merged back to the shared M(k)-stream. Then, CS will continue to play the video contents by receiving frames from this multicast stream.

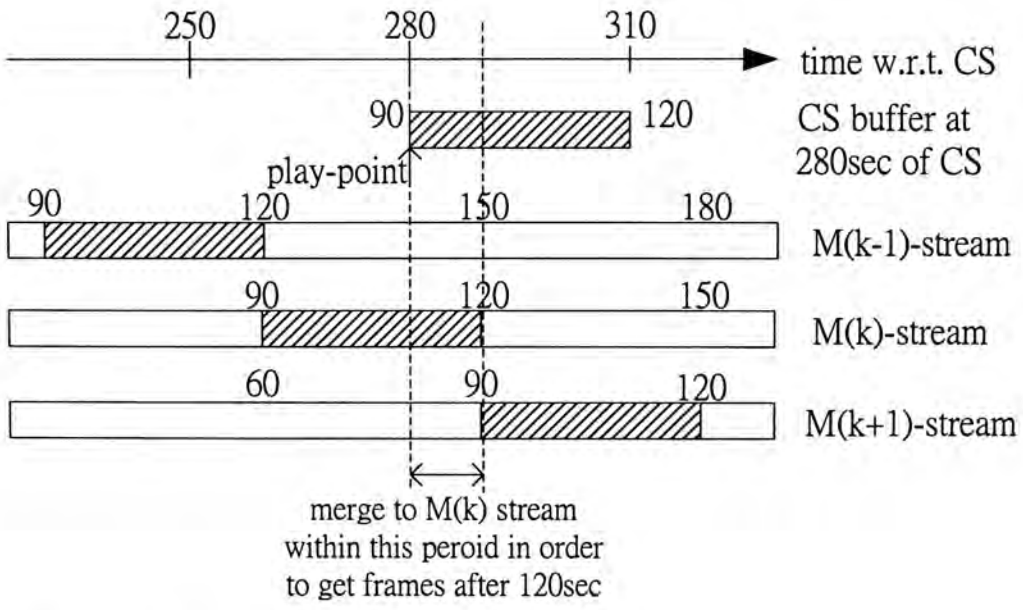


Figure 8. Merge Operation

In this operation, it is difficult though not impossible to control the exact time when the CS merges back to the M(k)-stream. To be safe, the CS should join the M(k)-stream before the 120sec time mark of the M(k)-stream in order not to miss any packet after the 120sec mark. Since the frames received before the 120sec time mark of M(k)-stream duplicates with those in the buffer, those frames will be discarded.

In addition, for the sake of simplicity, we describe that the buffer in CS must be large enough to hold the frames within a stream interval. However, the CS buffer capacity should actually be a little bit higher to prevent the fail of merging operation due to transmission delay.

The actual buffer size is shown in Figure 9. The right part of the buffer is the additional buffer space that needs to store the frames for the guard time. The guard time can be different for different network architecture. Generally, the guard time should be larger than the transmission delay between the CS and the MVSC.

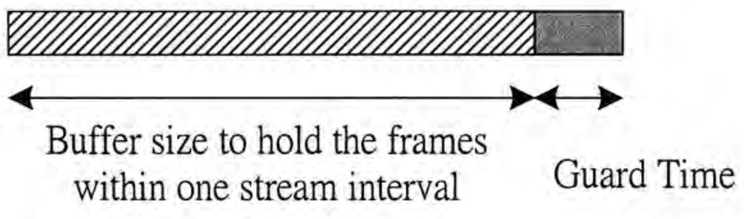


Figure 9. Actual buffer size

Figure 10 shows why the additional buffer is necessary for merging. Suppose the CS buffer is filled up just before CS 290sec, Ideally, CS is able to get the 120sec frame form M(k)-stream immediately. But actually, CS may not be able to join the

M(k)-stream in time to get the 120sec frame due to the network transmission delay or some other reasons. Also, it can not join the M(k+1)-stream because all the frames in the buffer will be played out before the arrival of 120sec frame from the M(k+1)-stream. Thus, the merge operation will fail in this case. However, if the buffer is a little bit larger with a guard time portion (e.g. 5sec), the merge operation will be successful. The additional frames in the guard time portion allows the CS to join the M(k+1)-stream to get the 125sec frame onwards.

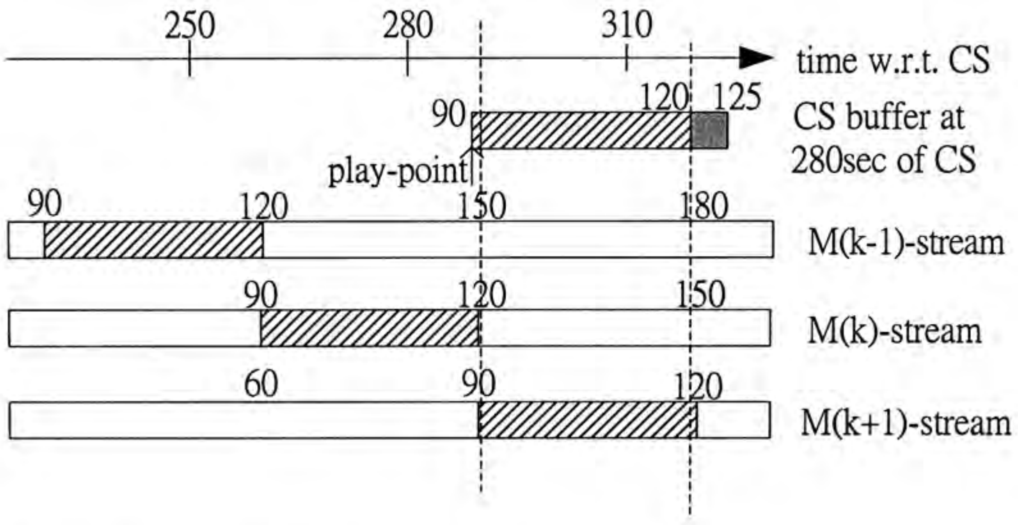


Figure 10. Merge operation with guard time buffer

3.4 Interactive Function

3.4.1 Pause

Pause freezes the normal play and keeps the play-point unchanged. During Pause, the CS buffer continues to receive data from the current multicast stream, while no data is consumed. Thus, data will accumulate at the buffer. If normal play is resumed before the CS buffer is full, the CS can continue to receive data from the same multicast stream. Only the play-point position in the buffer is changed.

If Pause continues until buffer is full, the CS must leave the current multicast stream, as it cannot accept any more data. Some past researches suggested that pause could be provided by continuous jumping to the next multicast stream [7]. However, it is inefficient when the client pauses the video for a long time as it makes many unnecessary group jumping.

In this proposal, the CS does nothing after the buffer is filled up. It keeps the frames in the buffer for merging. Once normal play is resumed, it will try to find the appropriate multicast stream to merge. The merge operation is the same as what has been described in Chapter 3.3.

Figure 11 shows the buffer state of CS during the pause operation. In this example, the batching interval is assumed to be 30sec. Thus, the CS buffer must be capable of holding the frames of 30 seconds. The vertical time mark is relative to the

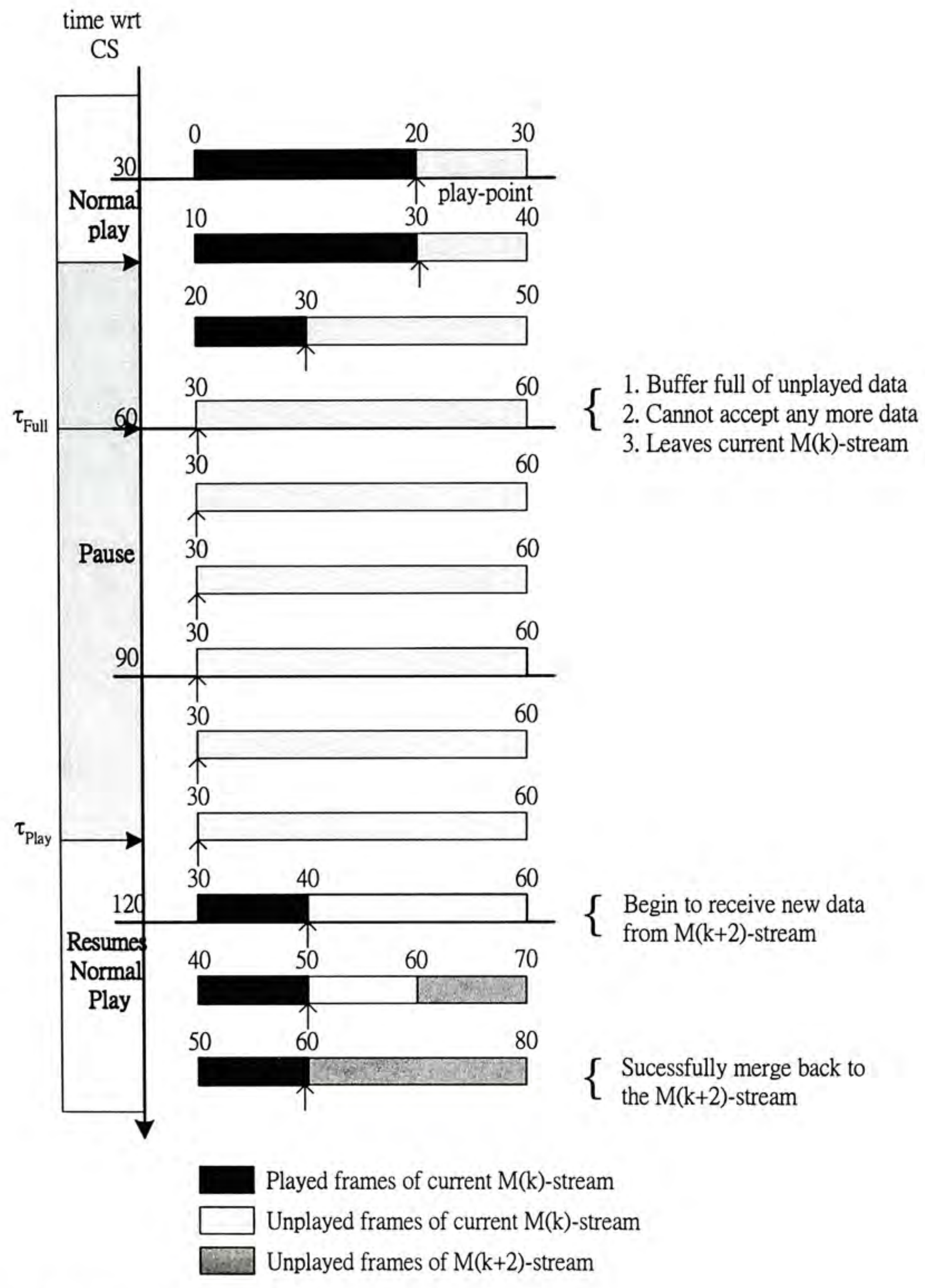

time wrt CS
30
Normal play
τFull
60
Pause
90
τPlay
120
Resumes Normal Play
0 20 30
play-point
10 30 40
20 30 50
30 60
1. Buffer full of unplayed data
2. Cannot accept any more data
3. Leaves current M(k)-stream
30 40 60
Begin to receive new data from M(k+2)-stream
40 50 60 70
50 60 80
Sucessfully merge back to the M(k+2)-stream
Played frames of current M(k)-stream
Unplayed frames of current M(k)-stream
Unplayed frames of M(k+2)-stream

Figure 11. CS buffer state in Pause

CS. During normal play, the frames consuming rate is equal to the frames incoming rate. Thus, the play-point position in the buffer is unchanged. After pause is issued, no frames is played and consumed while new frames coming from the network continuously. The play-point position moves towards the front end of the buffer until the buffer is filled up. At this time, the CS cannot accept any more frames and leaves the current multicast stream, say M(k)-stream.

Suppose the user pause for 70 seconds and resumes normal play again at CS 110sec time mark. The CS starts to play and consume the frames in the buffer. Since the frames stored in CS buffer is up to the 60sec of the video content, CS needs to merge back to a proper multicast stream to get the frames from 60sec onwards. In this example, suppose the CS is originally in the M(k)-stream, it will merge to the M(k+2)-stream at CS 120sec in order to receive the frames after 60sec of the video content. It is because each stream is separated by 30 seconds. At CS 60sec, the M(k)-stream delivers the 60sec frame. Thus, the M(k+1)-stream delivers the 60sec frame at CS 90sec and M(k+2)-stream delivers the 60sec frames at CS 120sec. Thus, CS can only joins M(k+2)-stream to get the frame after 60sec. Figure 12 illustrates the merge operation in this example.

CS will start to receive frames from M(k+2)-stream before CS 120sec in order not to miss any frames after the 60sec of the video content. However, they will be discarded because those frames are duplicated with the frames already in the buffer. At CS 140 time mark, CS plays out all the frames from its original stream and continues to play with the new frames coming from the new multicast stream. The

CS successfully merges back to the multicast stream at this moment.

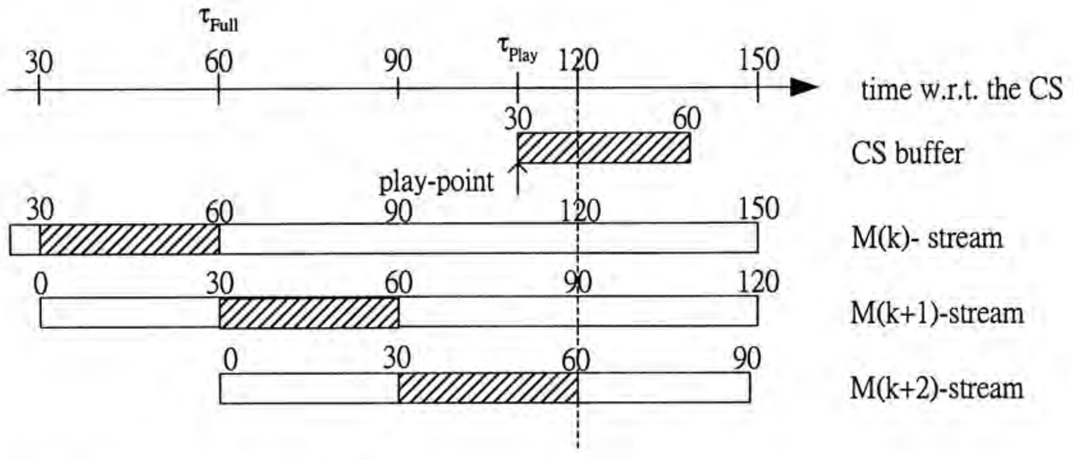


Figure 12. Merge operation in Pause

To choose an appropriate multicast stream, the following algorithm is used. Let τ_{Play} be the time at the user resumes the normal play and τ_{Full} be the time at the buffer full. Thus, $(\tau_{Play} - \tau_{Full})$ is the time lags the original multicast stream.

$$\begin{array}{l} \textit{If } m \times \textit{stream interval} \leq (\tau_{Play} - \tau_{Full}) < (m+1) \times \textit{stream interval}, \\ \textit{then merge to } M(k+m+1) \textit{ stream} \end{array} \tag{2}$$

where *m* should normally be non-negative as the CS rejoins the later multicast streams for it to maintain the same position in the video. When the play-point is paused near the end of the stream and the pause time is large, there may be a wrap around of the streams and m can take on negative values.

In the above example,

$$\begin{array}{l} \textit{Since } 1 \times 30 \leq 110 - 60 < (1+1) \times 30, \textit{ CS merges to } M(k+1+1) \textit{ stream} \\ \textit{i.e. } M(k+2) \textit{ - stream} \end{array}$$

3.4.2 Slow Motion

Slow motion is to play a stream at a slower speed, e.g. 0.5X. During the slow motion, data consumption rate is smaller than the arrival rate. Thus data will be accumulated in the buffer. Similar to pause, if playback is resumed before the CS buffer is full, the CS can continue to receive data from the current multicast stream.

If slow motion continues until the buffer is full, the CS must leave the current multicast stream. CS will continue the slow motion up to the end of the buffer. Meanwhile, CS needs to join the next stream in order to get the required frame for continuing the slow motion. It is also necessary for the user resumes normal play at any time.

The CS buffer state shown in Figure 13 helps to explain how slow motion works. In Figure 13, slow motion at 0.5X begins at CS 20sec. Afterwards, frames of 5 seconds are played in every 10 seconds of CS time. However, the incoming frame rate is unchanged. Thus, frames of 5 seconds are accumulated in every 10 seconds of CS time. The buffer will be full at CS 80sec and the CS must leave the current M(k)-stream. Then the CS joins the next M(k+1)-stream to get the required frames after 80sec. Since every stream is separated by 30sec, the frames after 80sec from M(k+1)-stream will be available at CS 110sec. At CS 120 sec, the CS resumes normal play. CS merges successfully when the old frames have already been played out and the CS continues normal play with the incoming frames from the new M(k+1)-stream.

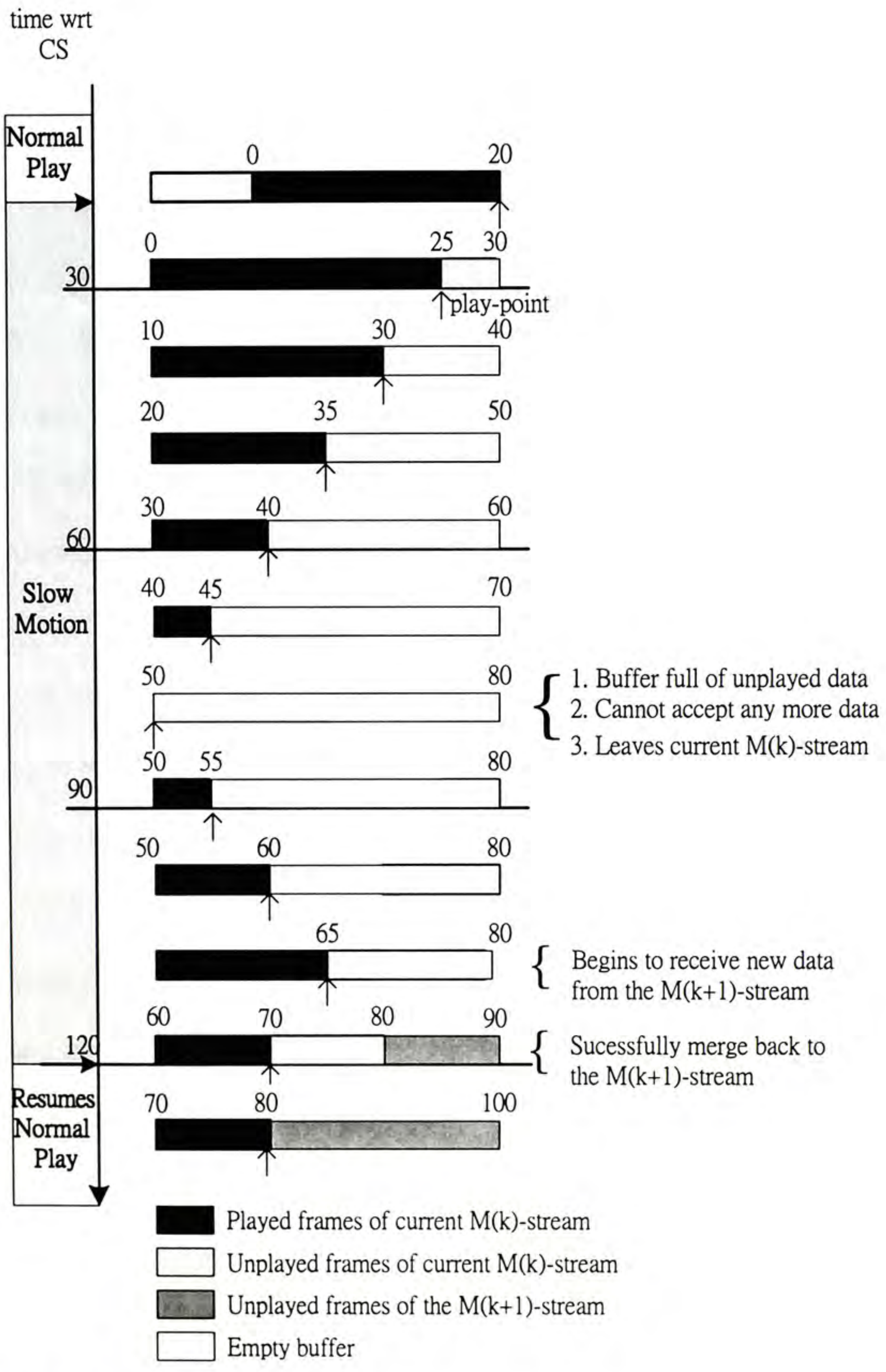

time wrt CS
Normal Play
30
60
Slow Motion
90
120
Resumes Normal Play
0
20
0
25
30
play-point
10
30
40
20
35
50
30
40
60
40
45
70
50
80
1. Buffer full of unplayed data
2. Cannot accept any more data
3. Leaves current M(k)-stream
50
55
80
50
60
80
65
80
Begins to receive new data from the M(k+1)-stream
60
70
80
90
Sucessfully merge back to the M(k+1)-stream
70
80
100
Played frames of current M(k)-stream
Unplayed frames of current M(k)-stream
Unplayed frames of the M(k+1)-stream
Empty buffer

Figure 13. CS buffer state in Slow Motion

3.4.3 Various Speed Fast Forward / Fast Rewind (FF/REW)

Fast forward is to play frames faster than the normal speed. The CS first tries to use the frames in its own buffer to serve FF by skipping some of the frames. If the FF action exceeds the range of the frames in buffer, some other methods will be needed to deal with these requests. In Ref. [7], it is suggested that the server could increase the transmission rate to higher rate by skipping some of the frames. However, this method is very inefficient. It wastes the server and network resources, as the client only needs some of the frames.

We propose to use the video pre-recorded at different speeds for both forward and reverse direction to serve the fast forward and backward requests respectively. This scheme not only utilizes bandwidth more efficiently, but also provides various speeds for the fast forward and backward actions (e.g. 2X, 4X, etc) that are offered in advance VCR. These pre-recorded video contents are stored in DIS and delivered by the I-streams from DIS.

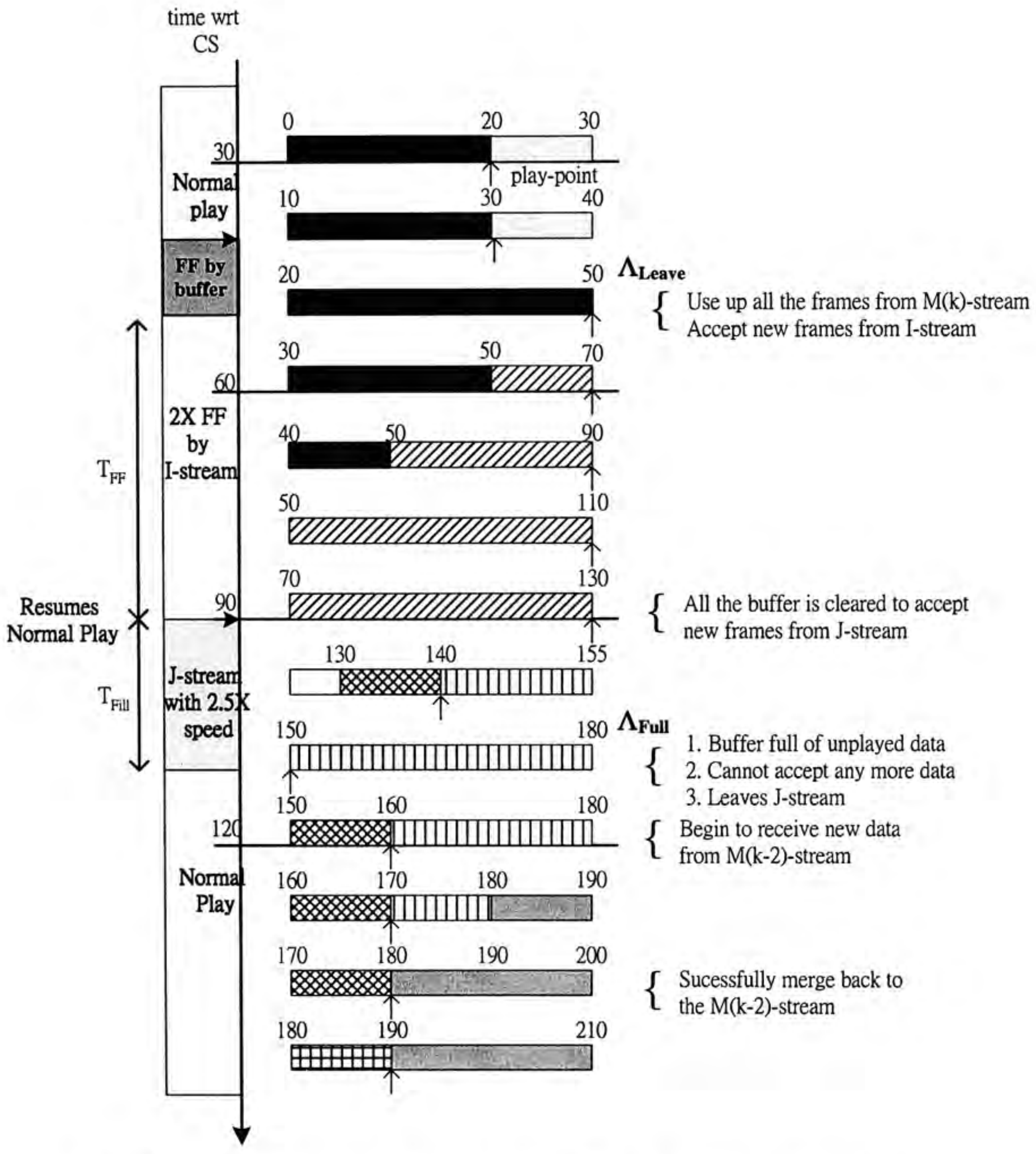
When a user requests a 2X fast forward action, the DIS sends the pre-record 2X forward I-stream at the required time to the CS. CS will play the frames without wasting any bandwidth. It may be argued that this method wastes hard disk space. However, this is a trade-off and the hard disk cost is clearly cheaper than the bandwidth cost. More importantly, this method can provide better service and more choices to the users.

When the CS ends the interactive function and resumes the normal play, the entire buffer in the CS is cleared, as they are no longer valid. Then, a J-stream is sent by the DIS to transmit data at a faster rate to the CS. It is to fill up the buffer for the merging operation as mentioned in chapter 3.3.

To illustrate how it works, Figure 14 shows the CS buffer state during a fast forward operation. At CS 40sec, the user issues fast forward command (e.g 2X FF) and CS first uses the unplayed frames in its buffer to serve this interaction. Fast forward is served by skipping frames with new frames coming from the current multicast stream continuously.

After 10 seconds, all the frames in the buffer are used up. CS needs to leave the current multicast stream and request DIS to provide the required frames. Then, DIS sends the requested pre-recorded double speed video content to the CS through I-stream. It can be seen that the video play-point moves at double speed and 10-second buffer space contains 20-second video content.

When the user resumes normal play at CS 90sec, all the frames in the buffer will be discarded because they are no longer useful. Meanwhile, CS requests the DIS to transmit J-stream so that it can merge back to a multicast stream. In this example, the J-stream is transmitted at 2.5X speed. 25-second video content is delivered to the CS for 10 seconds and 10-second of them are played. The time for filling up the buffer is 20 seconds that is calculated from Eq (1). As mentioned in chapter 3.3, CS can rejoins a suitable multicast stream once the buffer is filled up.



■ Played frames of current M(k)-stream	⊞ Played frames of M(k-2)-stream
□ Unplayed frames of current M(k)-stream	▩ Unplayed frames of M(k-2)-stream
▦ Played frames of J-stream	▨ Played frames of I-stream
▥ Unplayed frames of J-stream	□ Empty Buffer

Figure 14. CS buffer state in Fast Forward

Let T_{FF} be the time for the fast forward action that served by the I-stream, T_{Fill} is the time for filling up the buffer that defined in chapter 3.3, Λ_{Leave} is the play-time being delivered by the originally multicast stream when the CS leaves it and Λ_{Full} is the time for the frame at the end of the buffer after J-stream fills it up. After the fast forward operation and buffer filled up, the difference of the play-time between the original and the target multicast stream is $(\Lambda_{Full} - \Lambda_{Leave})$ and $(T_{FF} + T_{Fill})$ is the total time for this operation. Thus, $[(\Lambda_{Full} - \Lambda_{Leave}) - (T_{FF} + T_{Fill})]$ is the play-time of the new multicast stream ahead of the play-time of the original multicast stream. The timing diagram is shown in Figure 15. Suppose the CS is originally in the M(k)-stream.

The algorithm to determine the suitable multicast stream is as follows:

$$\begin{aligned}&\textit{If } m \times \textit{stream interval} \le [(\Lambda_{Full} - \Lambda_{Leave}) - (T_{FF} + T_{Fill})] \\ &\quad < (m+1) \times \textit{stream interval} \\ &\textit{then merge to } M(k-m) \textit{ stream}\end{aligned} \tag{3}$$

where m should be non-negative as it needs to go to the earlier opened stream in order to get to the later part of the viewing.

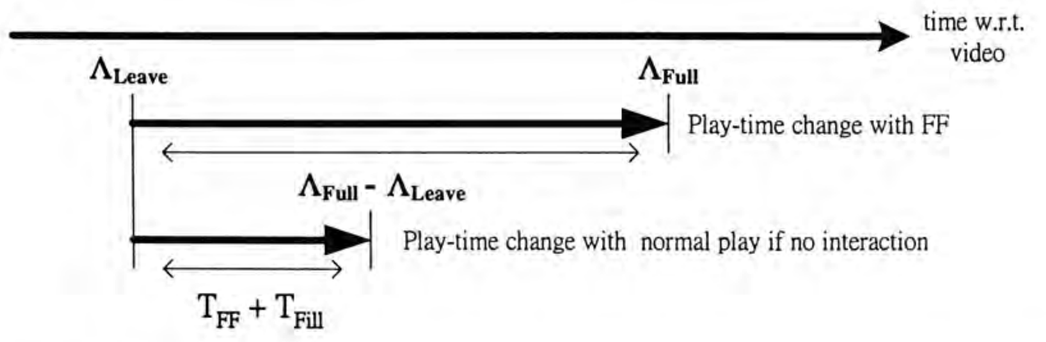


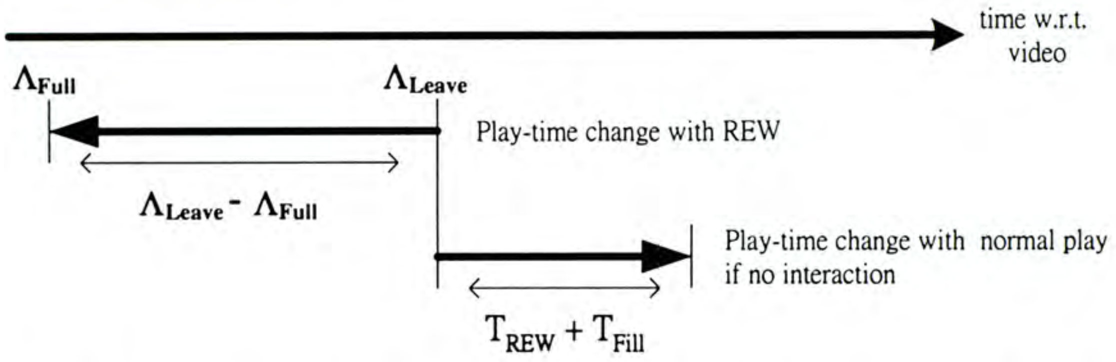
Figure 15. Difference between play-time for fast forward operation and normal play back operation

In the above example, $\Lambda_{Full} = 180$, $\Lambda_{Leave} = 50$, $T_{FF} = 40$, $T_{Fill} = 20$ and *stream interval* = *30*. Thus,

$[\ (\ \Lambda_{Full} - \Lambda_{Leave}\ ) - (T_{FF} + T_{Fill}\ )\ ] = 70.$

$2 \times 30 \leq 70 < 3 \times 30$

Then, CS merges to M(k-2)-stream that is the same as the result in the Figure 14.



time w.r.t. video
Λ_{Full}
Λ_{Leave}
Play-time change with REW
$\Lambda_{Leave} - \Lambda_{Full}$
Play-time change with normal play if no interaction
$T_{REW} + T_{Fill}$

Figure 16. Difference between play-time for fast backward operation and normal play back operation

For fast backward, the operation is similar to fast forward. However, it will bring the play-time backwards. The DIS will send a reverse pre-recorded 2X or 4X I-stream to CS and J-stream is also used to fill up the buffer for merging. Let T_{REW} be the time for fast backward action that served by the I-stream. In this case, after the fast backward operation, the play-time moves backwards by $(\ \Lambda_{Leave} - \Lambda_{Full}\ )$ and the time for the whole operation is $(T_{REW} + T_{Fill}\ )$. Thus, the play-time of the new multicast stream lags that of the original multicast stream by $[\ (\ \Lambda_{Leave} - \Lambda_{Full}\ ) + (T_{REW} + T_{Fill}\ )\ ]$. The timing is shown in Figure 16. Suppose the CS is originally in the M(k)-stream.

The algorithm to determine the suitable multicast stream is as follows:

$$\begin{aligned}&\textit{If } m \times \textit{stream interval} \leq [\,(\Lambda_{Leave} - \Lambda_{Full}) + (T_{REW} + T_{Fill})\,] \\ &\quad < (m+1) \times \textit{stream interval} \qquad (4)\\ &\textit{then merge to } M(k+m) \textit{ stream}\end{aligned}$$

where *m* should be non-negative in order to go to later stream for the earlier part of the viewing.

3.4.4 Jump Forward/Jump Backward (JF/JB)

Jump forward or backward is to go to a specific play-point immediately. It is an advanced feature in VCD and DVD players that allows user to search frames by directly going to that play-point. When a user issues a jump forward request, it will first determine whether the target time frame is in the buffer. If yes, the user can be served by just moving the play-point position in the buffer to the required frames.

If the target frame is not in the buffer, a J-stream beginning at the required time will be sent immediately from the DIS. The CS clears its own buffer and plays frames from the J-stream. It receives frames from the J-stream until the buffer is full, then leaves the J-stream and rejoins back to a multicast stream. In fact, these operations are similar to fast forward except that no I-stream is involved.

Figure 17 shows a sample jump forward operation. At CS 50sec, the user issues a jump forward command that jumps from 40sec to 190sec of the video content. A J-stream that starts at 190sec is transmitted from DIS to the CS immediately. Thus

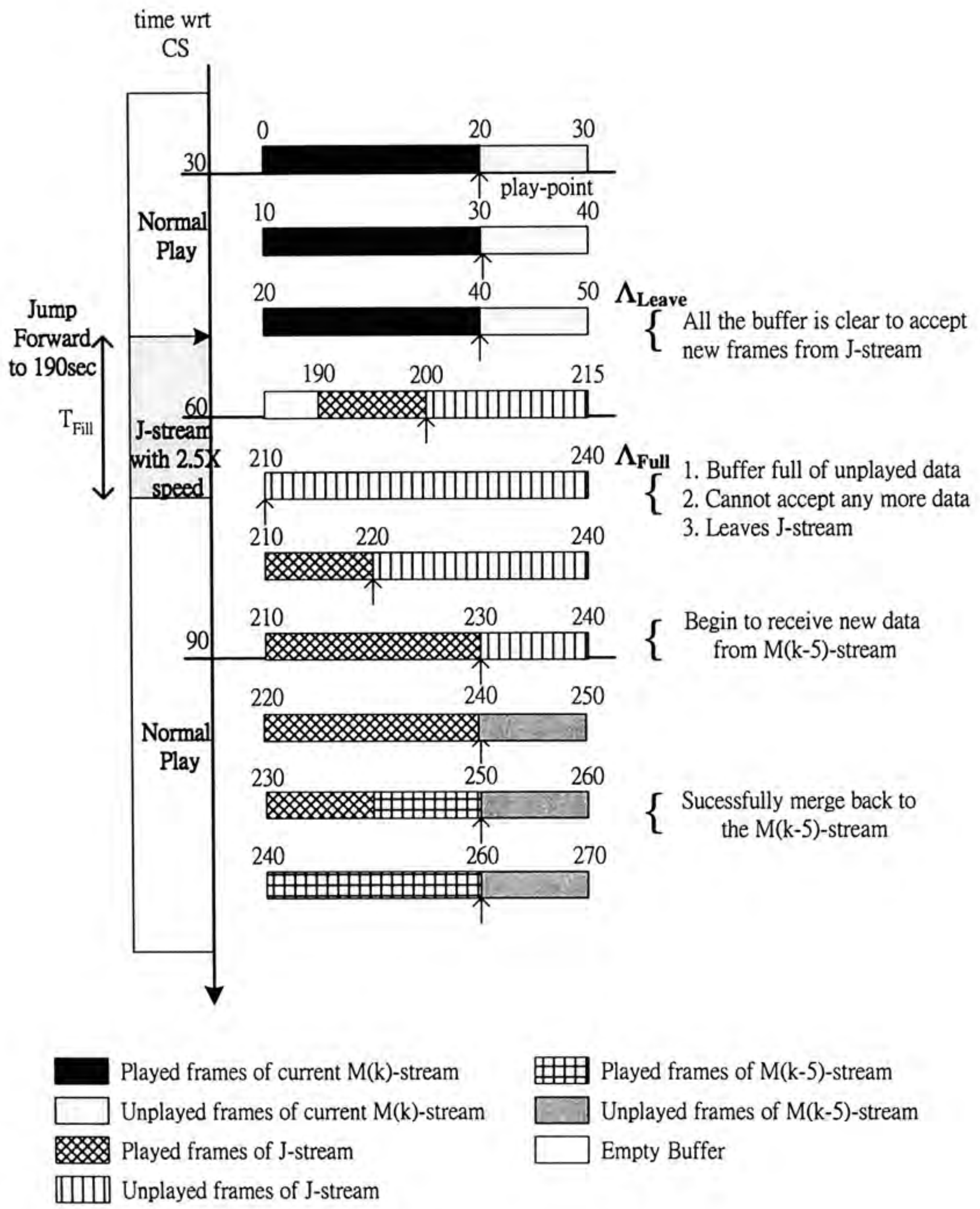

time wrt CS
30
Normal Play
Jump Forward to 190sec
T_Fill
60
J-stream with 2.5X speed
90
Normal Play
0 20 30
play-point
10 30 40
20 40 50
Λ_Leave
All the buffer is clear to accept new frames from J-stream
190 200 215
210 240
Λ_Full
1. Buffer full of unplayed data
2. Cannot accept any more data
3. Leaves J-stream
210 220 240
210 230 240
Begin to receive new data from M(k-5)-stream
220 240 250
230 250 260
Sucessfully merge back to the M(k-5)-stream
240 260 270
Played frames of current M(k)-stream
Unplayed frames of current M(k)-stream
Played frames of J-stream
Unplayed frames of J-stream
Played frames of M(k-5)-stream
Unplayed frames of M(k-5)-stream
Empty Buffer

Figure 17. CS buffer state in Jump Forward

the jump forward operation is satisfied and the buffer is filled up after 20sec. Then, the CS merges back to an appropriate multicast stream. After the jump forward operation and buffer filled up, the play-time advances for $(\Lambda_{Full} - \Lambda_{Leave})$ and T_{Fill} is the total time for this operation. Thus, $[(\Lambda_{Full} - \Lambda_{Leave}) - T_{Fill}]$ is the play-time of the new multicast stream ahead of the play-time of the original multicast stream. Suppose the CS is originally in the M(k)-stream.

The algorithm to find out the appropriate multicast stream is

$$\begin{aligned} &\textit{If } m \times \textit{stream interval} \leq [(\Lambda_{Full} - \Lambda_{Leave}) - T_{Fill}] \\ &\quad < (m+1) \times \textit{stream interval} \\ &\textit{then merge to } M(k-m) \textit{ stream} \end{aligned} \tag{5}$$

where *m* should be non-negative as it requests the later part of the viewing by jumping back to the earlier stream.

In the above example, $\Lambda_{Full} = 240$, $\Lambda_{Leave} = 50$, $T_{Fill} = 20$ and $\textit{stream interval} = 30$. Thus,

$[(\Lambda_{Full} - \Lambda_{Leave}) - T_{Fill}] = 170$.

$5 \times 30 \leq 170 < 6 \times 30$

Then, CS merges to M(k-5)-stream that is the same as the result in the Figure 17.

The jump backward operation is similar to jump forward, except that jump backward will jump to a play-time at the earlier part of the viewing. After the jump backward operation and buffer filled up, the play-point jumps back for $(\Lambda_{Leave} - \Lambda_{Full})$. Thus, $[(\Lambda_{Leave} - \Lambda_{Full}) + T_{Fill}]$ is the play-time of the new multicast stream lags that of the original multicast stream. Suppose the CS is originally in the M(k)-stream.

The algorithm to find out the proper multicast stream is

$$\begin{aligned} &\textit{If } m \times \textit{stream interval} \leq [(\Lambda_{Leave} - \Lambda_{Full}) + T_{Fill}] \\ &\quad < (m+1) \times \textit{stream interval} \\ &\textit{then merge to } M(k+m) \textit{ stream} \end{aligned} \tag{6}$$

where *m* should be non-negative as it requests the earlier part of the viewing by jumping to the later stream.

3.5 Performance Analysis

In this architecture, the MVSC is able to serve nearly unlimited number of normal play clients. Thus, the performance of the DIS in handling interactive functions has a great impact on the system performance that we want to optimize [17]. In Ref. [18], a generic VOD performance model has been proposed. In this model, the user request is modeled as a Poisson process and the service time is exponentially distributed with mean *T*. We will employ this model to evaluate the DIS performance. Only network resource is considered in our analysis. The performance is measured with respect to the blocking probability versus the request arrival rate.

3.5.1 Model

From the interactive functions described in chapter 3.4, we can see that the interactive functions supported in this VOD system can be divided into 3 types based on the resource they required.

1) **Pause and Slow Motion:** They can be satisfied by simply jumping to other multicast streams.
2) **Fast Forward and Backward:** They consume both I-stream and J-stream from DIS.
3) **Jump Forward and Backward:** Only J-stream from DIS is consumed.

Let the data delivery rate of the I-stream be r (say 1.5Mbps for MPEG-1), the data delivery rate of J-stream be $s.r$ where s is the speed-up-ratio (say, 1.5 to 4), the probability for an interactive request to be pause, slow motion, fast forward, fast backward, jump forward and jump backward be P_{PAU}, P_{SM}, P_{FF}, P_{REW}, P_{JF} and P_{JB} respectively where $P_{PAU}+P_{SM}+P_{FF}+P_{REW}+P_{JF}+P_{JB} = 1$.

Since pause and slow motion can be served by jumping of multicast stream and no DIS is involved, no bandwidth is consumed from DIS. For jump forward or backward, it only consumes J-stream and thus bandwidth of $s.r$ is consumed from DIS. For fast forward or backward, two stages are involved. Firstly, I-stream will be opened for the time duration of fast forward or backward $T_{FF/REW}$, so bandwidth of r will be consumed. Then, J-stream will be opened for merging that CS back to a multicast stream and bandwidth of $s.r$ will be consumed during buffer filling up T_{Fill}. Thus the average bandwidth occupied by fast forward or backward operation is,

$$\left(\frac{T_{FF/REW}}{T_{Fill}+T_{FF/REW}} r + \frac{T_{Fill}}{T_{Fill}+T_{FF/REW}} sr \right) Mbps$$

Then, the average data delivery rate from DIS for each user interaction R is,

$$\begin{aligned} R &= (P_{PAU}+P_{SM}) \times 0 + (P_{JF}+P_{JB}) \times sr \\ &+ (P_{FF}+P_{REW}) \times \left(\frac{T_{FF/REW}}{T_{Fill}+T_{FF/REW}} r + \frac{T_{Fill}}{T_{Fill}+T_{FF/REW}} sr \right) Mbps \end{aligned} \quad (7)$$

Let B be the bandwidth of the DIS allocated to the interactive functions, then the maximum number of concurrent streams n DIS can support is,

$$n = \frac{B}{R} \tag{8}$$

For pause and slow motion, DIS does not need to serve them. For jump forward or backward, DIS serves for the time of T_{Fill} until CS buffer is filled up. Fast forward or backward involve two stages. Firstly, I-stream will be opened for the time of $T_{FF/REW}$. Then, J-stream will be opened for the time of T_{Fill}. Thus, the average service time per user T, defined as the average time that the user is connected to the interactive server,

$$T = (P_{PAU} + P_{SM}) \times 0 + (P_{JF} + P_{JB}) \times T_{Fill} + (P_{FF} + P_{REW}) \times (T_{FF/REW} + T_{Fill})\,sec \tag{9}$$

The average service rate μ is,

$$\mu = \frac{1}{T} \tag{10}$$

Since there is no buffering of requests in this system, any request that cannot be served immediately will be blocked. From the M/M/n queuing model, it follows that

$$\begin{gathered} \text{Blocking Probability } P_{BL} = \frac{\rho^n}{n!} p_0 \\ \text{where } p_0 = [\sum_{k=0}^{n} \frac{\rho^k}{k!}]^{-1} \text{ and } \rho = \frac{\lambda}{\mu} \end{gathered} \tag{11}$$

where λ is the average requests arrival rate.

3.5.2 System Parameters

The DIS is assumed to be a high-end server with a 1Gbps network interface. Assume the interactive functions utilize 60% of the total bandwidth. The probability for each interactive function to occur is assumed to be uniformly distributed. The video contents are assumed to be MPEG-1 video with transmission rate 1.5Mbps. The J-stream with speed-up-ratio s=2, 2.5, 3 and 3.5 will be studied. The batched stream interval of 30sec, 45sec and 60sec will also studied. From chapter 3.4, Eq. (7) and (9), among the duration of all the user interaction, we see that only the time duration for the fast forward or backward operation (duration for delivering I-stream) affects the DIS performance. Thus, the fast forward or backward duration $T_{FF/REW}$ with mean time 30sec, 60sec and 90sec has been studied.

3.5.3 Results

Figure 18 shows how the blocking probability changes with increasing request arrival rate under different speed-up-ratios of J-stream and a mean $T_{FF/REW}$ of 30sec. Same as predicted, the blocking probability increases when the rate of the requests increases. Also, it is seen that for a given blocking probability, the DIS can accept higher request rate at higher speed-up-ratio of J-stream.

This behavior can be explained by Eq. (7). As the speed-up-ratio s of J-stream increases, the time for filling the buffer T_{Fill} reduces. Thus, the holding time of

J-stream is also reduced and the DIS streams can be freed sooner to accept new requests. It in turn reduces the blocking probability. However, Eq. (7) also shows that by increasing the speed-up-ratio s, delivery rate R will also increase, and reduce the maximum concurrent stream n that DIS can support. In this case, the reduction of J-stream holding time has a more significant effect than reducing the number of concurrent streams.

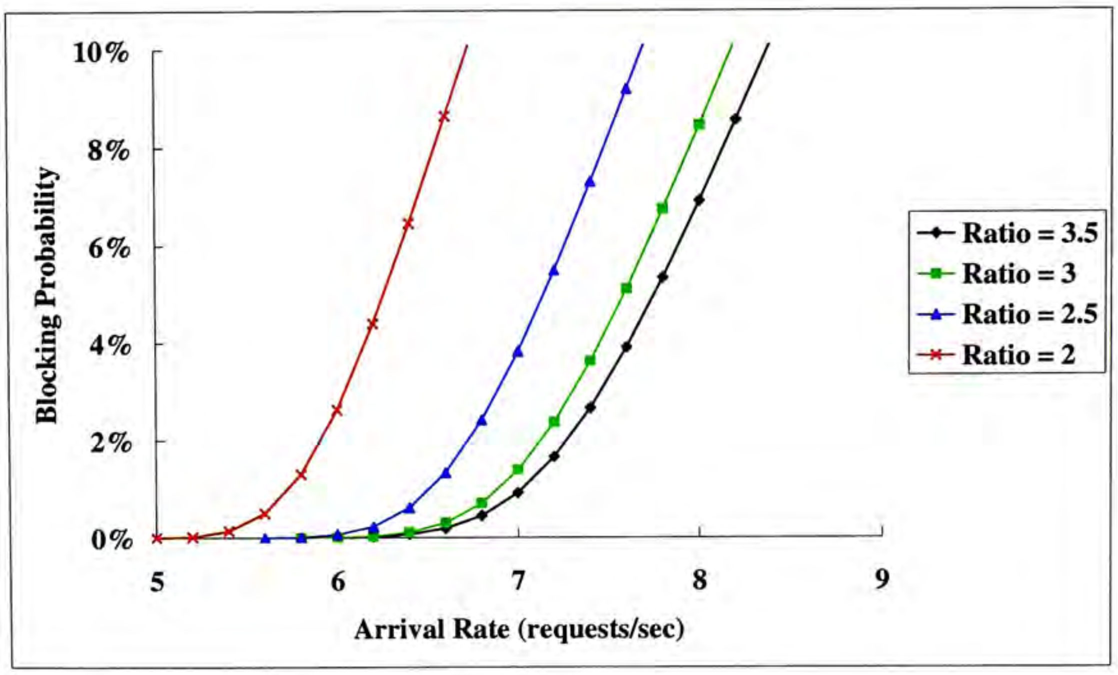


Figure 18. Impact of J-stream speed-up-ratio on the blocking probability with a mean $T_{FF/REW}$ of 30sec

Figure 19 and 20 shows how different speed-up-ratios affect the blocking probability with a mean $T_{FF/REW}$ of 60sec and 90sec respectively. A surprising result occurs. In Figure 19, we see that the J-stream with speed-up-ratio of 3 gives the best performance whereas, in Figure 20, the J-stream with speed-up-ratio of 2.5 has the best performance.

These behaviors can be explained by Eq. (7), (8) and (9). In the descriptions for Figure 18, we have already mentioned the effect of increasing speed-up-ratio in Eq. (7). However, we see from Eq. (9) that the effect of reducing T_{Fill} decreases as $T_{FF/REW}$ increases. The service time is affected significantly by $T_{FF/REW}$ if $T_{FF/REW}$ is large. Thus, the average service time is large and cannot be reduced significantly by simply reducing T_{Fill} at large $T_{FF/REW}$. A large service time means that the DIS stream is occupied longer and freed up slowly. At the same time, we see from Eq. (8) that a lower speed-up-ratio J-stream can provide more concurrent DIS streams. Thus, in the case of a long fast forward or backward duration, a lower speed-up-ratio J-stream could give a better performance.

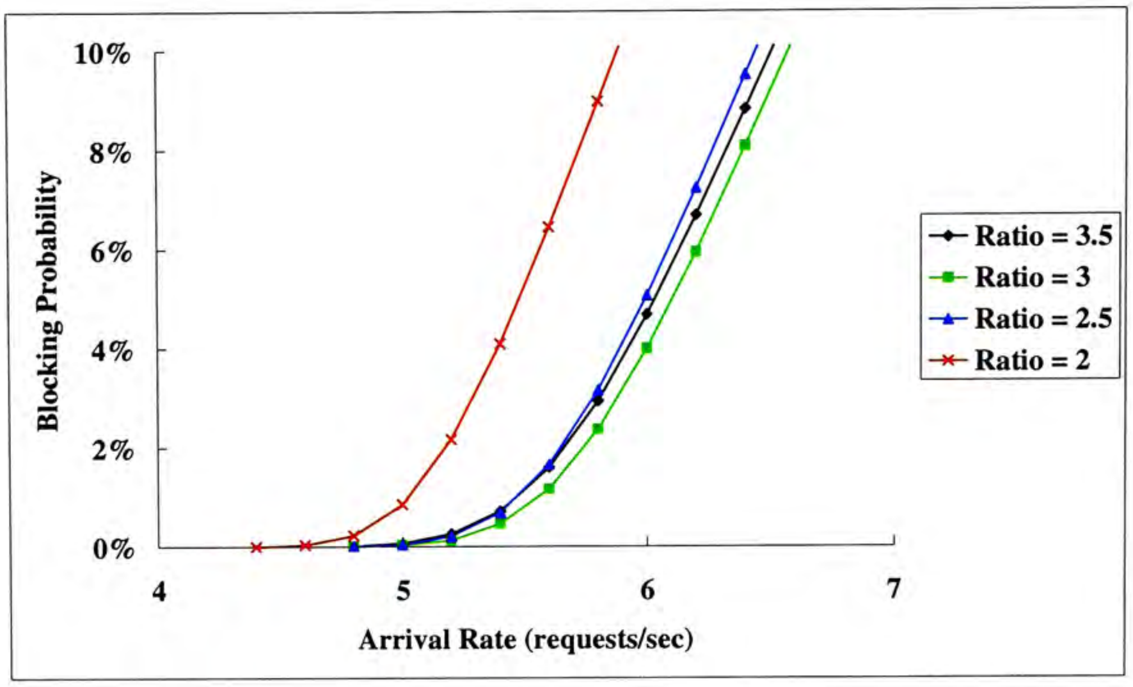


Figure 19. Impact of J-stream speed-up-ratio on the blocking probability with a mean $T_{FF/REW}$ of 60sec

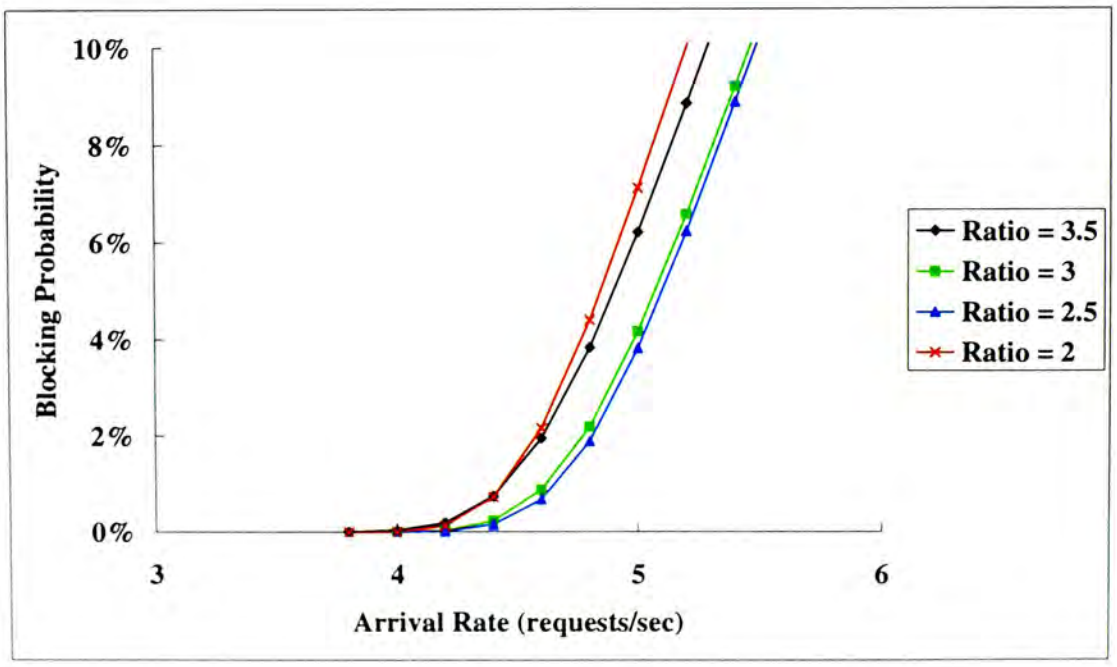


Figure 20. Impact of J-stream speed-up-ratio on the blocking probability with a mean $T_{FF/REW}$ of 90sec

A clearer relationship between the blocking probability and the speed-up-ratio is shown in Figure 21. It can be seen that for a J-stream with a speed-up-ratio of 2.6, it gives the best performance at a mean $T_{FF/REW}$ of 90sec, but gives a worse performance at a mean $T_{FF/REW}$ of 30sec. It can also be seen that at a mean $T_{FF/REW}$ of 30sec, the J-stream with speed-up-ratio 3.5 can reduce the blocking probability by nearly 15% than that with a speed-up-ratio of 2.

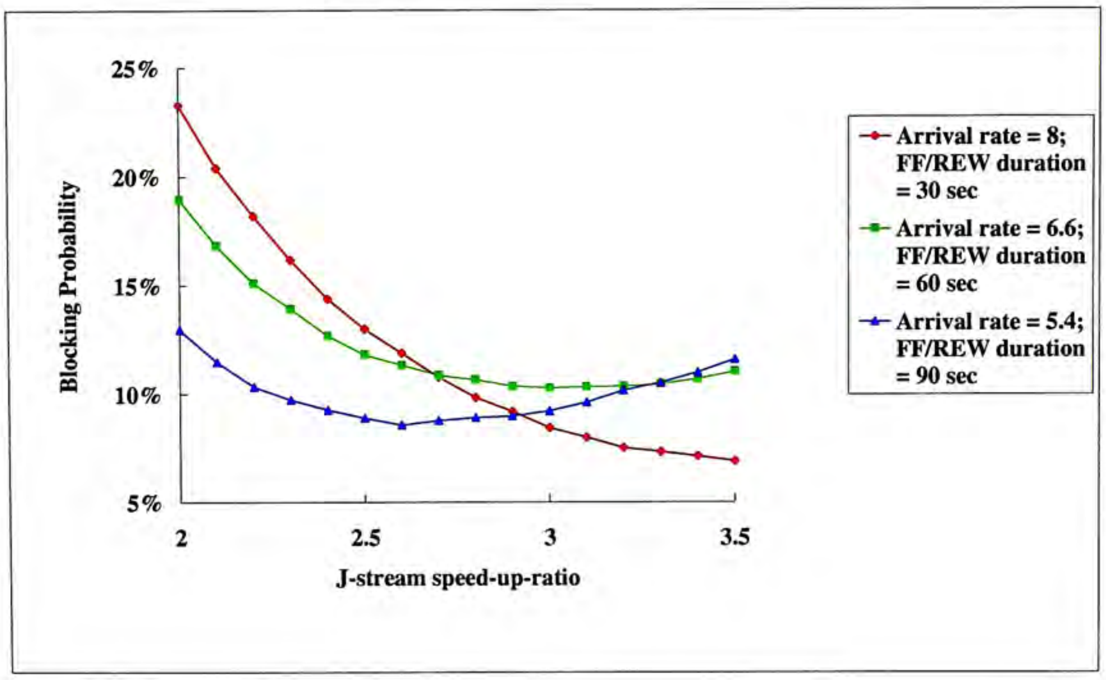


Figure 21. Relationship between blocking probability and J-stream speed-up-ratio

Figure 22 shows how different batched stream intervals affect the blocking probability with a mean $T_{FF/REW}$ of 30sec. We see that the performance drops as the batched stream interval increases. From Eq. (7), T_{Fill} increases as batched stream interval increases. Under the same fast forward or backward duration, increasing the stream interval will increase the service time and free up the DIS stream slowly. This causes a larger blocking probability, and thus reduces the performance. Besides, a longer stream interval requires the CS to have a larger buffer in order to store up all the frames within a stream interval.

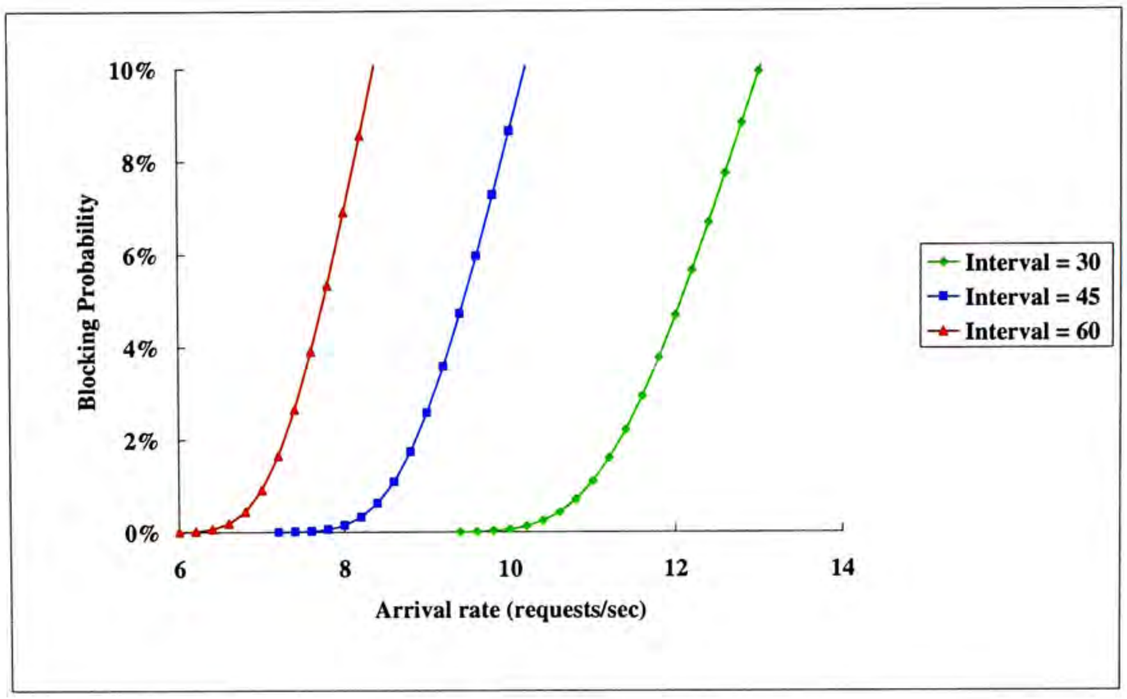


Figure 22. Impact of batched stream interval on the blocking probability with a mean $T_{FF/REW}$ of 30sec

A fast forward operation performed by a faster transmission rate (e.g. 2X) has been suggested in [7]. Figure 23 compares this scheme with the pre-record fast forward scheme in our system. Same as predicted, the pre-record fast forward has a better performance than double rate fast forward. It is because the I-stream consumes data rate *r* instead of *2r* in double rate fast forward. Thus, DIS can support more concurrent streams.

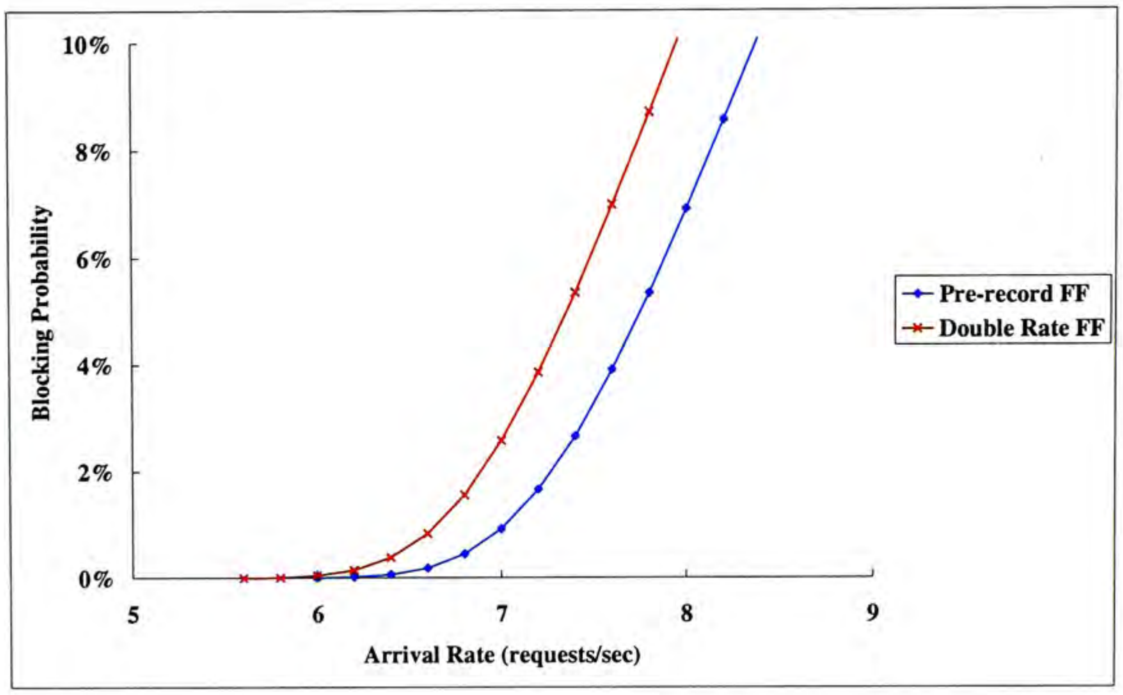


Figure 23. Comparison between Pre-record FF and Double Rate FF with a mean $T_{FF/REW}$ of 30sec

From the above analysis, we see that different combination of the parameters can affect the DIS performance significantly. The most encouraging result is on the effect of the combination of the J-stream speed-up-ratio and the fast forward or backward duration on the performance.

From these results, it is possible to optimize the system by performing some analysis and classification on both the video contents and the viewer behavior. For example, if we find out that a long fast forward or backward duration usually happens for movies of a particular category and the clients within a particular local region, we may provide a lower speed-up-ratio J-stream to those kinds of movies and local region.

From the analysis of the batched stream interval, it shows that a shorter interval have a better performance and lower requirement on the CS. However, shorter interval also means that a larger number of multicast streams are needed. And there will be a heavier loading on the central server and backbone network. Thus, this is a tradeoff to decide the batched stream interval.

Chapter 4

Design of a Video Proxy System

In this chapter, we discuss the design of a video proxy system that caches the popular video objects in the local distribution network. In chapter 4.1, we discuss the advantages of deploying video proxy in a VOD system. Detailed design issues will be described in chapter 4.2. The priority function, two-stage replacement policy and caching policy will be described and explained. To verify the design, chapter 4.3 presents the simulation results of the video proxy system.

4.1 Video Proxy System

In this architecture, the video contents are divided into two types, namely hot and non-hot contents. The hot contents are delivered by the multicast streams because they are expected to be viewed by many users at the same time. Batching [2] is deployed to generate multicast stream at each fixed batch interval in a round-robin manner. Since one multicast stream can support all the user requests within a batching interval, the backbone bandwidth loading can be reduced significantly. Basically all hot contents are cached in the DIS and will not be subject to any replacement policy except for changes issued by the operator. The hot contents in the DIS are used to provide various interactive functions.

For the non-hot contents, they are carried by the unicast streams. The replacement policy will choose the higher priority contents and cache them in the proxy server. These contents are expected to be re-accessed in the near future, so that the DPS can serve the future requests without accessing the UVSC or consuming backbone bandwidth.

In this video proxy system [19], DPS is the main component that is responsible for caching the popular contents. In order to utilize the disk space more efficiently and achieve higher bandwidth reduction, the cached contents in one DPS can be shared with other DPSs.

4.1.1 Priority Function

The priority function is to determine the priority of the cached contents in the proxy server at any time. The accessed frequency and recency were the important parameters in web caching. It is believed that these parameters can still be applied for the continuous streaming media, though they are not enough. Thus, an estimated popularity parameter is introduced to predict the popularity of a video content. The use of video object characteristics as caching parameters has also been suggested in [14]. The estimated popularity can be estimated by its rating, stars, directors, etc of the video content. Actually, a lot of cinemas use these kinds of information to predict the popularity of a movie and decide how many show rooms are allocated to that movie. The priority function of a cached content at any time $P_i(t)$ is defined as follows,

$$P_i(t) = \begin{cases} \left(P_i(t_{last}) + E_i + \alpha \frac{N_i}{N_{total}} \right) e^{-\beta(t-t_{last})} \\ P_i(t_{last}) = 0 \ \textit{for the first access} \end{cases} \tag{12}$$

where E_i is the estimated popularity of a video clip i
N_i is the number of times video clip i is accessed within a period
N_{total} is the total number of video clips i accessed within a period
t_{last} is the time that video clip i was last accessed
α is the bias value to the video clip's accessed frequency
β is the bias value to the video clip's accessed recency

This priority function includes the estimated popularity, access frequency and recency as an indicator to predict the popularity of a video object. Moreover, this function provides flexibility for the system to bias to any of the parameters.

4.1.2 Two-Stage Replacement Policy

In the DPS, the disk space is partitioned into two regions, namely the set and the prefixed region. The set region is responsible for storing an entire video object (like a whole movie), while the prefixed region is for storing the initial portion of a video object.

The replacement policy is performed when the newly requested object is not in the proxy server and there is not enough free space to store that newly requested object. Then, the priority of each cached object in the proxy server will be calculated by the priority function described in Eq. (12). The lowest priority objects will be removed until there is enough free space to store the new object. If there are more than one object that have the same priority value, the less recently accessed object will be removed.

In this caching system, the replaced object will not be removed completely. Instead, the initial portion of the object will be transferred from the set region to the prefix region and remained in the proxy server. The purpose is to minimize the start-up latency for this object when it is re-accessed. The minimum amount of an object S_i needed to be remained is shown in Eq. (13). With this portion of the video object, the client will experience the minimum latency and can view the object as if it is coming from the DPS. It is because the DPS can serve that client immediately without waiting for the arrival of the object from remote site.

$$S_i \geq (T_{DPS} + T_{UVSC}) \times r \tag{13}$$

where T_{DPS} is the transmission delay between DPSs
T_{UVSC} is the transmission delay between the DPS and UVSC
r is the transmission rate of the video object

When there is a need to store a new object's initial portion and there is not enough free space in the prefix region, the above replacement policy will be performed. The lowest priority objects' prefix will be removed until enough space is freed up.

4.1.3 Caching Policy

When there is a new request from the client, the directly attached DPS will search for the requested video object. If there is a match for the whole object, the DPS will deliver the requested object to the clients.

However, if there is only a match for the prefix of the object, the DPS also delivers that object's prefix to the client immediately. And at the same time, the DPS will make a request for the clients to other DPSs. If there is a match in one of the DPSs, the remaining portion of that object will be delivered from that DPS to the requesting DPS and then to the client. However, if no DPS contains that object, the requesting DPS will request the UVSC to get that object's remaining portion.

Meanwhile, the requesting DPS will cache that video object in its disk. That object's prefix will also be transferred from the prefix region to the set region so that it can merge with the remaining portion of the object. If there is not enough free space for that object, the replacement policy will be performed.

If there is no match for the object's prefix, then the clients need to wait until the requested object is obtained from other DPSs or UVSC. DPS will also free up the necessary space to cache this object.

4.2 Performance Evaluation

In our simulation, we have evaluated the performance of our proposed caching algorithm. It will also be compared with two well known caching algorithms, namely LRU and LFU.

4.2.1 Simulation Environment

In our simulation, the UVSC contains 400 video objects and the objects are divided into 10 sets with 10 levels of estimated popularity. The video objects length range from 90 min to 120 min. The objects are assumed to be MPEG-I movie and are transmitted in constant bit rate with 1.5Mbps. The user requests are generated in exponential distribution. Zipf distribution is employed to model the user access pattern. The backbone bandwidth from UVSC to each DPS can support up to 100 channels and the local distribution bandwidth from each DPS to clients can support up to 200 channels. The transmission delay between client and DPS, DPS and DPS, DPS and UVSC is assumed to be 5 ms, 10 ms and 20 ms respectively.

Unless otherwise specified, the disk space in each DPS is 10% of the total space in UVSC and the skewness parameters for Zipf is 0.3.

4.2.2 Performance Metric

Since the ultimate purpose for the video proxy system is to reduce the backbone bandwidth, the backbone bandwidth reduction ratio will serve as the performance metric in our simulation. If there is no video proxy, the central server needs to handle all the client requests. Thus, one unit of bandwidth used by the cached object can reduce one unit of backbone bandwidth. Thus, the bandwidth reduction ratio *BRR* is defined as,

$$BRR = \frac{BW_{cached}}{BW_{cached} + BW_{server}} \times 100\% \qquad (14)$$

where BW_{cached} is the bandwidth consumed by the cached object in DPS
BW_{server} is the bandwidth consumed by the un-cached object that needed to be delivered from UVSC

4.2.3 Results

Figure 24 shows the comparison of our caching algorithm with LFU and LRU under different disk spaces of the DPS. As expected, the bandwidth reduction increase as the amount of proxy space increase. It is because larger proxy space means more video objects can be cached and more client requests can be served from DPS. In addition, it shows that our proposed caching algorithm gives better performance than the other two algorithms. It can reduce nearly 20% and 10% more backbone bandwidth than LRU and LFU respectively.

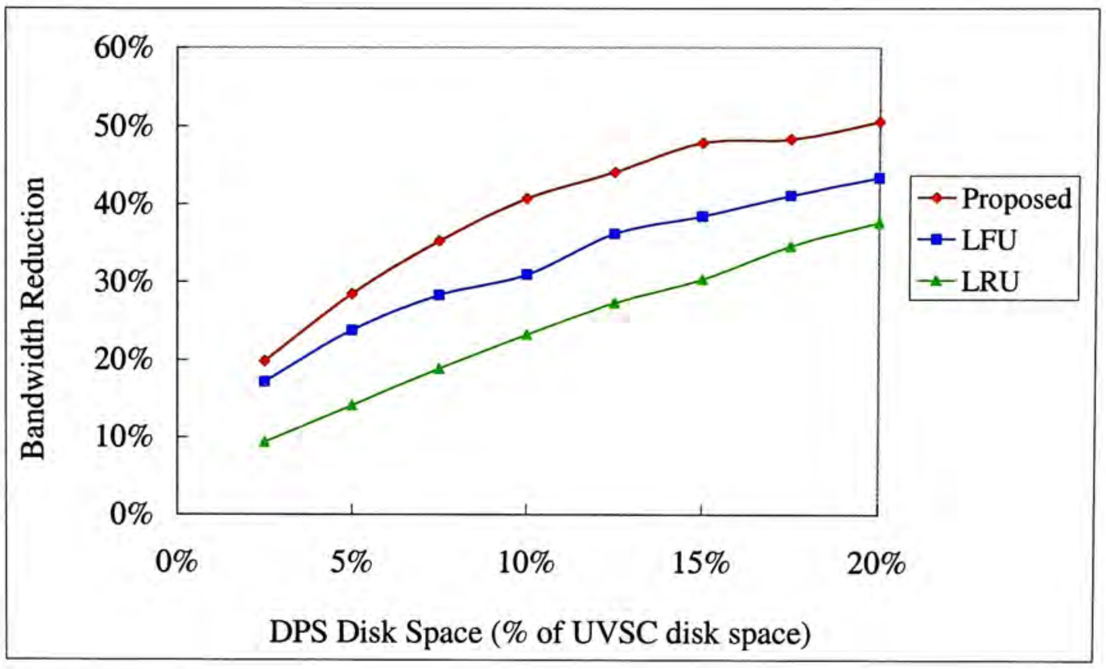


Figure 24. Different algorithm performance under different disk space of the DPS

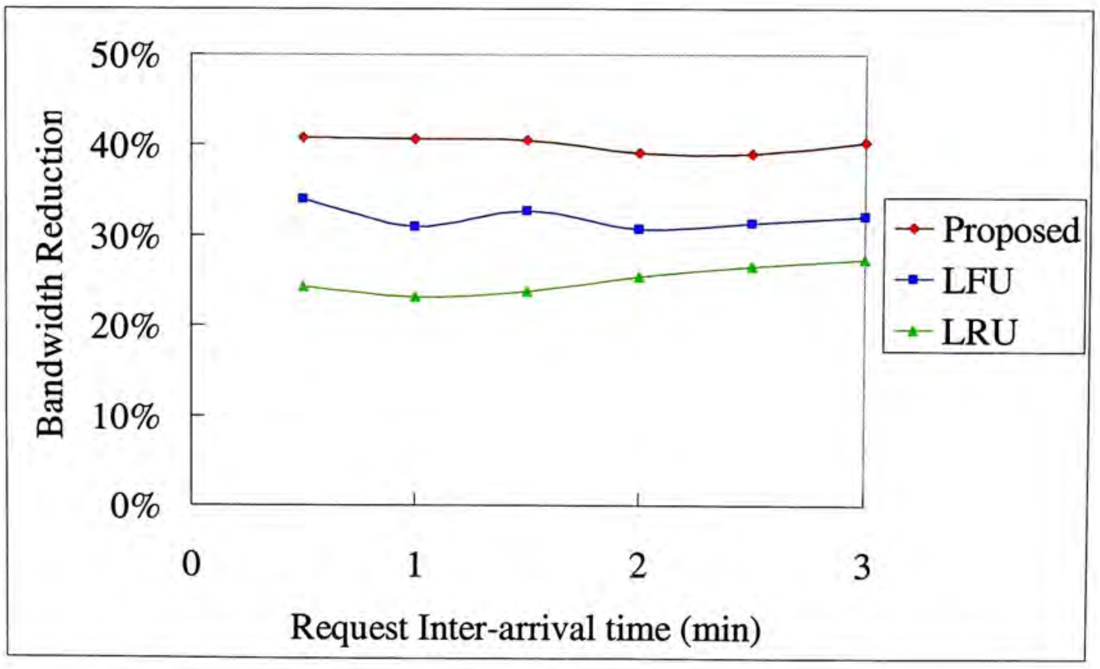


Figure 25. Different algorithm performance under different request inter-arrival time

In Figure 25, it shows that our proposed caching algorithm can also achieve higher bandwidth reduction ratio under different request inter-arrival time. It reduces nearly 20% and 10% more backbone bandwidth than LRU and LFU respectively.

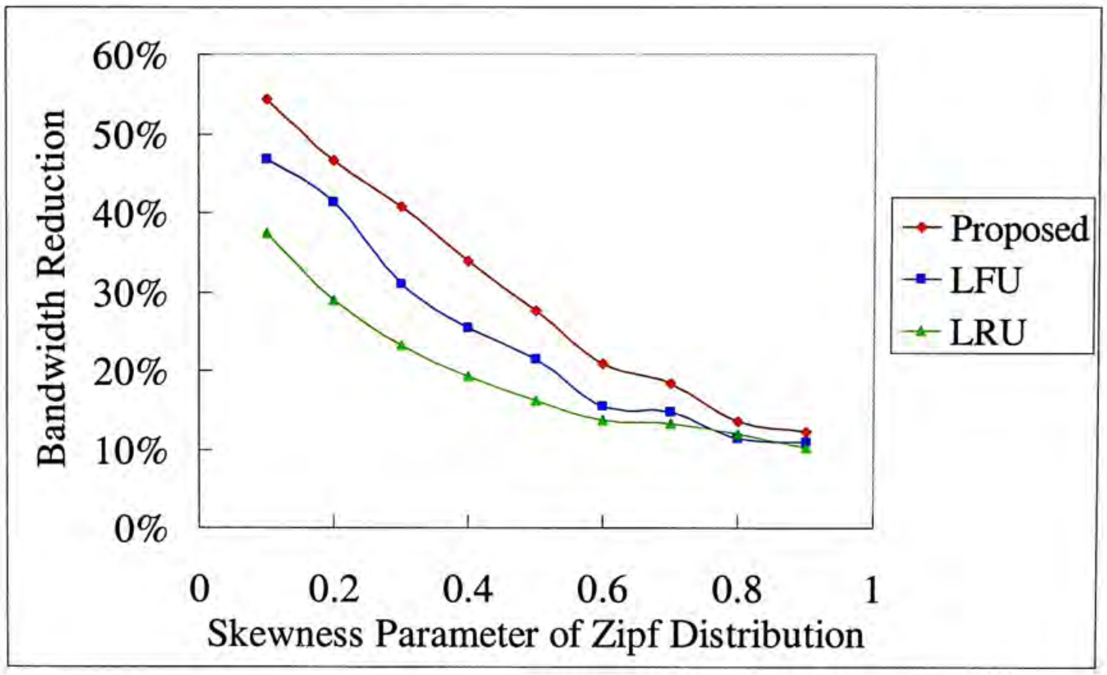


Figure 26. The relationship between the performance of different algorithm and the user access pattern

The relationship between the performance of different algorithms and user access pattern is shown in Figure 26. It can be seen that our algorithm can achieve better performance under different user access pattern. It is also shown that the performance and the difference in performance between the algorithms decrease as the skewness of Zipf distribution increase. It is because at high skewness, the access pattern for the video objects becomes more uniform. It means that the popularity of each object is nearly the same. So the cached objects are less likely to be re-accessed within a short period of time. On the other hand, when the user requests are concentrated on some of the video objects at low skewness, all the algorithms are able to choose and cache the more popular objects. Thus, more backbone bandwidth can be saved by providing these cached objects to the clients.

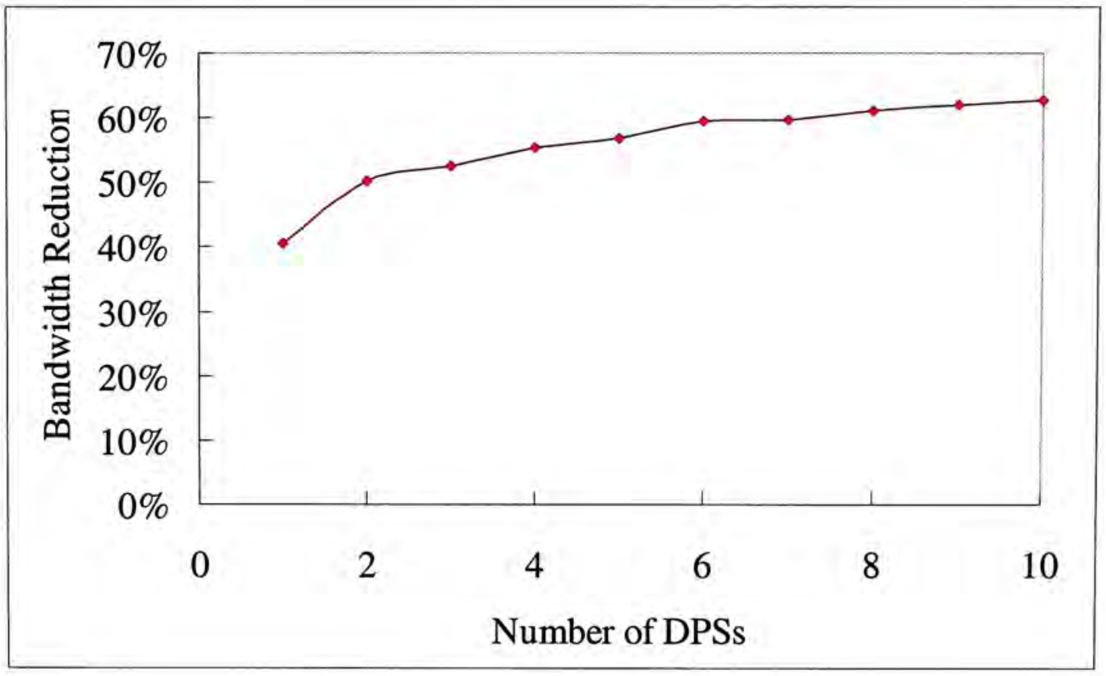


Figure 27. Performance with different number of DPSs

The reduction of the backbone bandwidth versus different number of DPSs is shown in Figure 27. It shows that the bandwidth reduction ratio increases with the increasing number of DPSs. It is because each DPS can share its cached object with other DPSs. Thus, most of the client requests can be satisfied without asking for the UVSC. In addition, it also shows that the increase in reduction ratio becomes saturated when the number of DPSs is large. It is because some of the requests are for the very unpopular objects that are not cached in the DPS for a long time. Each time they are requested, DPS may not be able to find them and need to ask the UVSC for them.

Figure 28 shows that the two-stage replacement policy can effectively reduce the start-up latency. It is because the DPS can use the prefix stored to serve most of the client requests immediately. Thus, it can compensate for the searching time needed for getting the requested object in other DPSs or UVSC.

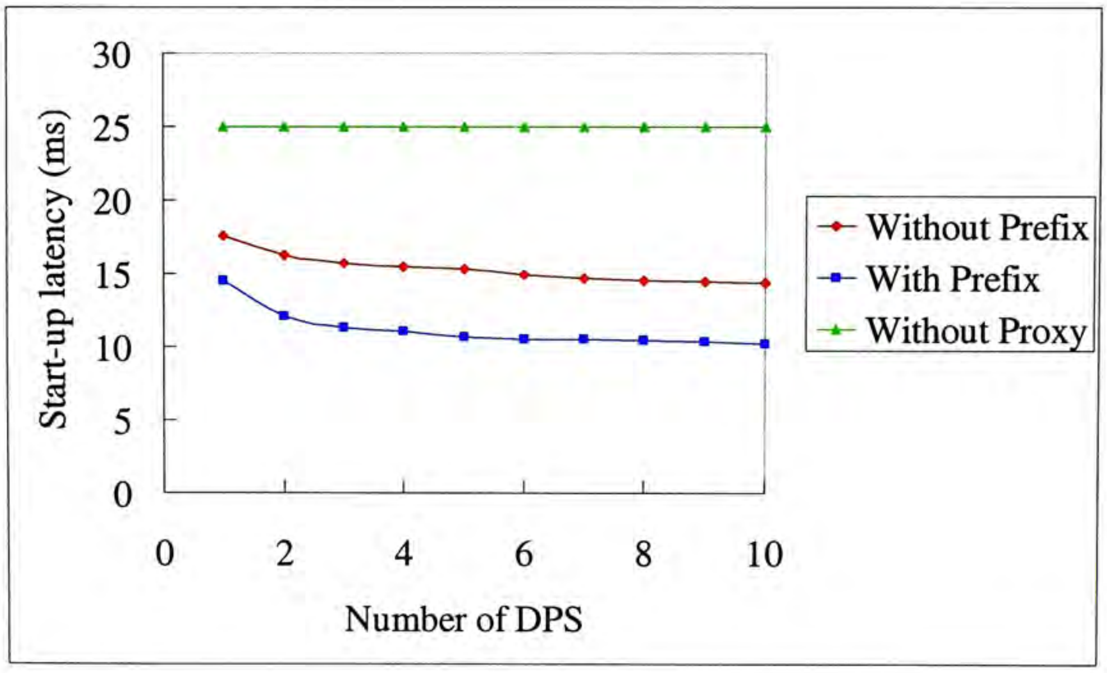


Figure 28. Effect on start-up latency for the Two-Stage Replacement Policy

Chapter 5

Conclusion

We have described a novel architecture together with a detailed description of the mechanisms for providing full interactivity for a highly efficient and scalable VOD system. To achieve this goal, we make use of the following techniques: (1) batched multicast transmission to reduce the cost by sharing the multicast stream with more users, (2) buffer management in the clients to allow smooth interaction, (3) merging a client back to a multicast stream after the interactive function to reduce the number of unicast streams needed which in turn increases the scalability and reduces the cost, (4) employing DIS to provide high quality interactive services to users and improve the scalability and flexibility, and (5) efficient use of network bandwidth to eliminate unnecessary traffic such as higher rate transmission for fast forward.

In this architecture, the MVSC is able to serve nearly unlimited number of normal play clients. Thus, the performance of the DIS in handling interactive functions has a great impact on the system performance. Thus, we have analyzed the performance of the DIS in our VOD system. The relationship between the DIS performance and the various factors – speed-up-ratio, the time duration of interaction, and the stream interval of the multicast streams has been studied. It is seen that an optimal speed-up-ratio for J-stream does exist. It is possible to optimize the system by performing some analysis and classification on both the video contents and the viewer behavior. It is also shown that pre-record fast forward is significantly better than using a double rate fast forward.

In order to further improve the scalability, we have described a Video Proxy System for our VOD system. Our proposed replacement algorithm considers the combination of the estimated popularity, access frequency and recency of a video clips. Moreover, a two-stage replacement policy is introduced to minimize the start-up latency. The simulation results show that our video proxy system can give a better performance in terms of reducing the start-up latency, backbone network and central server loading.

Bibliography

[1] J. Dey, J. Salehi, J. Kurose, and D. Towsley, "Providing VCR capabilities in large-scale video servers", *ACM Multimedia 94*, Oct 1994.

[2] A. Dean, D. Sitaram, and P. Shahabuddin, "Scheduling Batching," *Proc. ACM Multimedia '94* Pages 391-398, 1994.

[3] K. Almeroth, M. Ammar, "The Use of Multicast Delivery to Provide a Scalable and Interactive Video-on-Demand Service", *IEEE Journal on Selected Areas in Communications*, Vol. 14, No. 6 Pages 1110-1122, August 1996.

[4] W.J. Liao and V.O.K.Li, "The Spilt and Merge (SAM) Protocol for Interactive Video-on-Demand Systems," *IEEE Multimedia* Volume: 4 4, Pages 51-62, Oct-Dec 1997,

[5] Brubeck, D.W., Rowe, L.A., "Hierarchical storage management in a distributed VOD system," *IEEE Multimedia,* Volume: 3 3 , Pages 37 –47, Fall 1996

[6] C.H. Ng, K.C. Chan, K.W. Chan, C.H. Chan, C.Y. Tam and K.W. Cheung, "DINA – System for a Large-scale Fully interactive VOD System based on Hybrid Multicast-unicast Streaming", *IEEE-PCM'2000*, Dec 2000

[7] W.F. Poon, K.T. Lo and J Feng "Multicast Video-on-Demand System with VCR Functionality," *Communication Technology Proceedings*, International Conference on Communication Technology '98, Oct 1998, Pages 569-573 vol.1.

[8] Poon, W.-F., Lo, K.-T. and Feng, J, "Design and Analysis of Multicast Delivery to provide VCR functionality in video-on-demand systems", *ICATM '99*, Pages 132 –139H.

[9] J. T. robinson and N. V. Devarkonda, "Data Cache Management Using Frequency-Based Replacement", Proc. of the 1990 ACM SIGMETRICS, Pages 134-142

[10] E. J. O'Neil, P. E. O'Neil, and G.Weikum, "The LRU-k page replacement alogrithm for database disk buffering," Proc. of International conference on Management of Data, May 1993.

[11] Sen, S., Rexford, J. and Towsley, D., "Proxy Prefix Caching for Multimedia Streams", *Proceedings of INFOCOM '99*, Volume: 3 , 1999 , Pages 1310 –1319

[12] Y.W. Park, K.H. Baek and K.D. Chung, "Reducing Network Traffic Using Two-layered Cache Servers for Continuous Media Date on the Internet", *Proceedings of the 24th Computer Software and Applications Conference, 2000,* Pages 389 –394

[13] E.J. Lim, S.H Park, H.O. Hong and K.D. Chung, "A proxy Caching Scheme for Continuous Media Streams on the Internet", *Proceedings of the 15th International Conference on Information Networking, 2001,* Pages. 720 –725

[14] Sonah, B., Ito, M.R. "Cache Transparency in VOD Systems: Taking Advantage of Viewers' Flexibility", *Proc. of the Third International Conference on Computational Intelligence and Multimedia Applications,* 1999, Pages. 420 –425

[15] Rejaie, R., Haobo Yu, Handley, M. and Estrin, D, "Multimedia Proxy Caching Mechanism for Quality Adaptive Streaming Application in the Internet", *Proc. of INFOCOM 2000,* Vol. 2 , 2000 , Pages 980 –989

[16] Y. Wang, Z.L. Zhang, Du, D.H.C. and D. Su "A Network-Conscious Approach to End-to-End Video Delivery over Wide Area Networks Using Proxy Servers" *Proceedings of INFOCOM '98,* Vol. 2 , 1998 , Pages. 660 -667

[17] K.C. Chan, K.W. Cheung, "Performance Analysis on Distributed Interactive Server in a Large-scale Fully Interactive VOD System", IEEE-ICOIN-15, Feb 2001

[18] V.O.K. Li et al., "Performance model of interactive video-on-demand systems, *IEEE Journal on Selected Areas in Communications*, Vol 14, No. 6 pp. 1099-1109, August 1996.

[19] K.C. Chan, K.W. Cheung, "Video Proxy System for a large-scale VOD System", 5th WSES/IEEE CSCC2001, July 2001