DESIGN AND PERFORMANCE ANALYSIS

OF A

SUPER-SCALAR

VIDEO-ON-DEMAND SYSTEM

LEE CHUNG HING

A THESIS SUBMITTED IN PARTIAL FULFILLMENT

OF THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF PHILOSOPHY

IN

INFORMATION ENGINEERING

AUGUST 2001

To my family and Sylvia

Acknowledgements

This thesis is dedicated to my family. I would like to thank them for giving me support and care. Thank them especially for the delicious food and nutriment. I am especially grateful to my supervisor Professor Jack Y. B. Lee for his invaluable advice and guidance on my study. Through his guidance, both of my knowledge and vision broadened.

I also like to thank my colleagues, Ah Mo, Ken, Edward, Rudolf and Raymond for their kind advice and valuable idea. I am so appreciate to know all the people of IE Mphil99 who play and share with me over the years. Without them, I cannot have an enjoyable study period.

Finally, I would like to thank Sylvia who is the most special person in my life. Thank for her endless support especially for her patience on me.

Abstract

Despite the availability of video-on-demand (VoD) services in a handful of cities around the world, large-scale deployment of VoD services in a metropolitan is still economically impractical. A lot of researches have been done in improving the scalability of VoD systems, but to this day, the ultimate capacity of a video server is still finite.

In this thesis, we present a Super-Scalar Video Server (SS-VoD) which is a novel architecture to tackle this capacity problem. By the intelligent use of network multicast and client-side caching, the proposed architecture can vastly reduce server and network resource requirement. More importantly, the resource reduction increases with the load, and the server latency asymptotically approaches a constant when the load is further increased. For example, a small server with hardware capacity, which is equivalent to 50 concurrent streams in traditional video servers, can serve a 120-min video with an average latency no more than 5.6 seconds, regardless of the customer arrival rate. This thesis presents this new architecture, derives an approximate performance model, and evaluates the architecture using numerical results from analytical models, simulations, and benchmarking.

摘要

儘管在世界上很多城市已經可以享受視頻點播服務（VoD），但是在大城市裏大規模地提供視頻點播服務還是不切實際的。人們已經做了很多研究來提高視頻點播系統的容量，但是直到今天，視頻伺服器的最終容量還是有限的。

在本論文中，我們提出了一種新的結構來處理系統容量問題：超容量視頻伺服器（Super-Scalar Video Server）。通過智慧化地應用網路廣播（network multicast）和用戶端緩存（client-side caching），我們提出的構造可以極大地減少對伺服器和網路資源的需求。更重要的是，資源需求的減少和網路的負載成正比，而且當負載進一步增加時，伺服器的延遲接近一個常數。例如，對於一個支援 50 個資料流程的小伺服器，我們的構造可以支援無限多的人欣賞一齣長一百二十分鐘的錄像，並且平均延遲不超過 5.6 秒。本論文的內容包括：提出了這種新的構造，推出了一個近似的性能模型，用理論模型的數值結果分析了這種構造。

Contents

Acknowledgements ..... **ii**

Abstract ..... **iii**

List of Figures ..... **vii**

1. Introduction **1**

1.1 Contributions of This Thesis 3

1.2 Organizations of This Thesis 3

1.3 Publication 4

2. Overview of VoD Systems **5**

2.1 True VoD 6

2.2 Near VoD 7

2.3 Related Works 9

2.3.1 Batching 9

2.3.2 Patching 11

2.3.3 Mcache 11

2.3.4 Unified VoD 12

2.4 Discussions 15

3. Super-Scalar Architecture **17**

3.1 Transmission Scheduling 20

3.2 Admission Control 21

3.3 Channel Merging 26

3.4 Interactive Control 29

4. Performance Modeling **31**

4.1 Waiting Time for Statically-Admitted Clients 32

4.2 Waiting Time for Dynamically-Admitted Clients 33

4.3 Admission Threshold 38

4.4 Channel Partitioning 39
5. Performance Evaluation 40
5.1 Model Validation 40
5.2 Channel Partitioning 42
5.3 Latency Comparisons 44
5.4 Channel Requirement 46
5.5 Performance at Light Loads 47
5.6 Multiplexing Gain 49
6. Implementation and Benchmarking 51
6.1 Implementation Description 51
6.2 Benchmarking 53
6.2.1 Benchmarking Setup 53
6.2.2 Benchmarking Result 55
7. Conclusion 56
Appendix 57
Bibliography 61

List of Figures

FIGURE 2.1: A TYPICAL VIDEO-ON-DEMAND SYSTEM 6

FIGURE 2.2: CHANNEL SCHEDULING OF TVOD 7

FIGURE 2.3: CHANNEL SCHEDULING OF NVOD 8

FIGURE 2.4: ARCHITECTURE OF THE UVOD SYSTEM 12

FIGURE 2.5: SCHEDULING OF MULTICAST CYCLES FOR A MOVIE IN UVOD 13

FIGURE 3.1: SYSTEM ARCHITECTURE. 18

FIGURE 3.2: TRANSMISSION SCHEDULE FOR ONE MOVIE. 20

FIGURE 3.3: STATE-TRANSITION DIAGRAM FOR THE ADMISSION CONTROLLER. 22

FIGURE 3.4: STATE-TRANSITION DIAGRAM FOR THE SERVICE NODES. 24

FIGURE 3.5: STATICALLY-ADMITTED 27

FIGURE 3.6: DYNAMICALLY-ADMITTED 27

FIGURE 3.7: CHANNEL MERGENCE OF MULTI-USER ON SS-VOD 28

FIGURE 4.1: USER CLASSIFICATION IN DYNAMIC MULTICAST CHANNEL 34

FIGURE 5.1: LATENCY COMPARISON OF ANALYTIC AND SIMULATION RESULTS 41

FIGURE 5.2: NORMALIZED LATENCY VERSUS PROPORTIONS OF DYNAMIC MULTICAST CHANNEL 42

FIGURE 5.3: OPTIMUM CHANNEL ALLOCATION OF SS-VOD AND UVOD 44

FIGURE 5.4: LATENCY COMPARISON OF SS-VOD WITH TYPICAL VOD SYSTEMS 45

FIGURE 5.5: CHANNEL REQUIREMENT OF SS-VOD AND TYPICAL VOD SYSTEMS..... 46

FIGURE 5.6: CHANNEL REDUCTION OVER TVOD 48

FIGURE 5.7: MULTIPLEXING GAIN OF SS-VOD SYSTEM 50

FIGURE 6.1: A SS-VOD NETWORK. 52

Chapter 1

Introduction

Video-on-demand (VoD) systems have been commercially available for many years. However, except for a few cities, large-scale deployment of VoD service is still uncommon. One of the reasons is the high cost in provisioning large-scale interactive VoD service. The traditional model of true-video-on-demand (TVoD) calls for a dedicated channel, both at the server and at the network, for each active user during the entire duration of the session (e.g. 1~2 hours for movies). In a city with potentially millions of subscribers, the required infrastructure investment would be immense.

To tackle this problem, a number of researchers have started to investigate various innovative architectures in an attempt to improve the scalability and efficiency of large-scale VoD systems [1-15]. Examples include the periodic broadcasting approach by Chiueh *et al.* [1], the batching approach by Dan *et al.* [2] and Shachnai *et al.* [3], the split and merge protocol by Liao *et al.* [4], the stream tapping scheme by Carter *et al.* [5], the pyramid broadcasting approach by Viswanathan *et al.* [6] and Aggarwal *et al.* [7], the piggybacking approach by

Golubchik *et al.* [8], Aggarwal *et al.* [9], the patching approach by Hua *et al.* [10] and the Mcache approach by Ramesh *et al.* [11], and so on. It is beyond the scope of this thesis to compare these difference approaches and the interested readers are referred to [5] for some comparative discussions.

In this thesis, we present a Super-Scalar Video-on-Demand architecture (SS-VoD) to tackle this capacity problem. By the intelligent use of network multicast and client-side caching, the proposed architecture can vastly reduce server and network resource requirement. Specifically, our goal in this thesis is to design a video-on-demand system that scales in a super-linear manner. Consequently, the average latency as experienced by new customers should asymptotically approach a constant (e.g. a few seconds). For example, our results show that a small SS-VoD server with hardware capacity equivalent to 50 concurrent streams in traditional video server can serve a 120-min video with an average latency no more than 5.6 seconds, regardless of the customer arrival rate.

This super-scalar property will enable the VoD system to serve a huge number of concurrent users – a requirement for metropolitan-scale deployments. Unlike traditional video servers where the server cost increases proportionally for larger user population, the server-cost per customer for the proposed super-scalar architecture decreases for larger user population. Hence this super-scalar architecture can provide a cost-effective solution for deploying large-scale, city-wide video-on-demand services.

1.1 Contributions of This Thesis

The main contribution of this thesis is in the design and performance evaluation of the Super-Scalar Video-on-Demand (SS-VoD) architecture, which provides a solution to the capacity challenge in city-wide VoD service deployments. Specifically, we developed a transmission scheduling algorithm for scheduling multicast transmission, and an admission control algorithm for processing and scheduling client requests. To characterize the performance of the SS-VoD architecture, we derived an approximate performance model for system dimensioning. The performance model is validated against simulation results and is shown to be a reasonable approximation. To prove the feasibility of the SS-VoD architecture, we implement a SS-VoD prototype with off-the-shelf computer hardware and software. Benchmarking results obtained from the prototype show that the SS-VoD architecture is indeed super-scalar, and can be scaled up to millions of users.

1.2 Organizations of This Thesis

This thesis is organized as follows. Chapter 1 presents an introduction to this study. Chapter 2 presents an overview of video-on-demand systems and related works. Chapter 3 presents the SS-VoD architecture. Chapter 4 presents an approximate performance model of SS-VoD. Chapter 5 evaluates the performance of SS-VoD via numerical and simulation results. Chapter 6 presents the implementation

of the SS-VoD prototype and benchmarking results. Finally, Chapter 7 concludes this thesis.

1.3 Publication

V.C.H. Lee, J.Y.B. Lee, "Improving UVoD System Efficiency With Batching," *Proc. International Conference on Software, Telecommunications and Computer Networks – SoftCOM*, Croatia, October 2000.

Chapter 2

Overview of VoD Systems

Many video-on-demand (VoD) systems have been developed in the last decade. Figure 2.1 depicts a typical VoD system, consisting of a video server, an interconnection network, and multiple video clients. The video server usually is a high-performance computer that stores compressed digital video in harddisks for transmission to clients over the network. A video client receives video data from the video server and decodes them for playback. Client device can be a single personal computer (PC) or a set-top box (STB) connected to a television. The interconnection network can be implemented using Ethernet, ATM or other network technologies. The following sections review the existing VoD architectures.

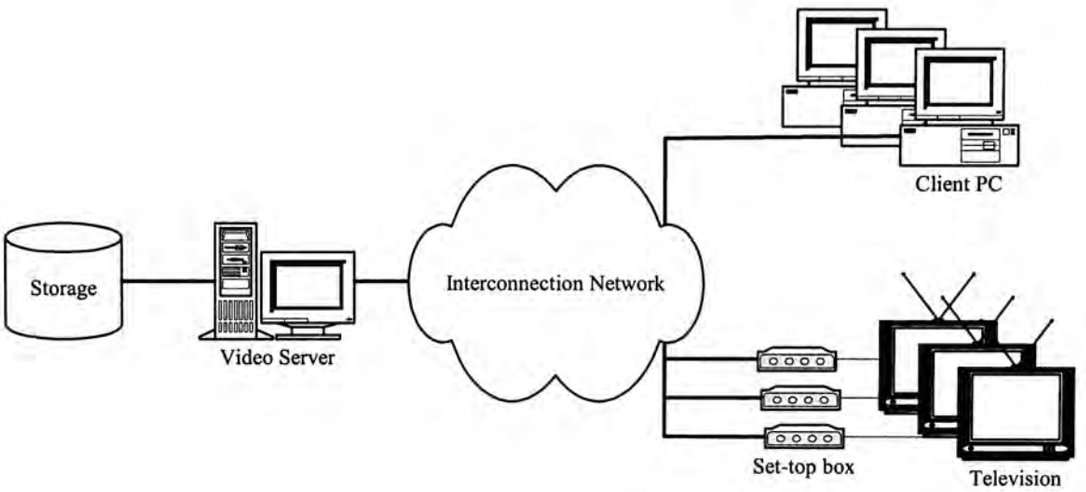

Figure 2.1: A typical video-on-demand system

2.1 True VoD

The most common type of VoD systems is called true video-on-demand (TVoD), depicted in Figure 2.2. To initiate a new video session, a client first sends a request to the video server for a specific video. The video server processes the request by retrieving video data from the storage and streaming the video data over the network to the client device. The client device typically first buffers a small amount of video data and then starts video playback while concurrently receiving video data from the server.

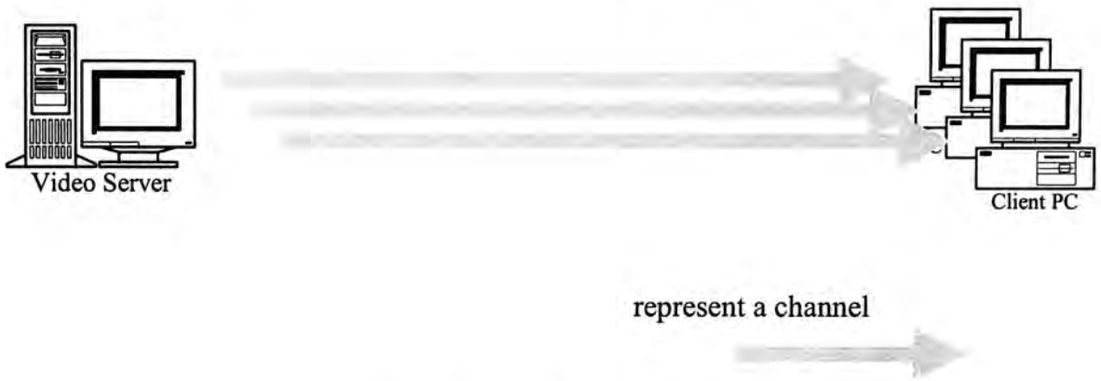

Figure 2.2: Channel Scheduling of TVoD

In a TVoD system, each client has its own dedicated channel for video streaming where a channel refers to network and server resources allocated for a video session. As each user has a dedicated channel, one can perform interactive VCR-like controls such as pause/resume, fast-forward and rewind at any time. However, as the video server has a limited number of channels for serving users, arriving users will be denied service once all these channels are occupied. Therefore the resource requirement of a TVoD system is proportional to the desired system capacity and this limits the scalability of VoD systems designed using the TVoD architecture.

2.2 Near VoD

Another type of VoD system, commonly called near video-on-demand (NVoD) [1], repeatedly transmits a video stream over multicast or broadcast channels as shown in Figure 2.3. The channels are scheduled in advance and independent of the

client request. For example, suppose there are 10 channels assigned for a video, and the movie length is 120 minutes. Then the movie will be restarted every 12 minutes as shown in Figure 2.3.

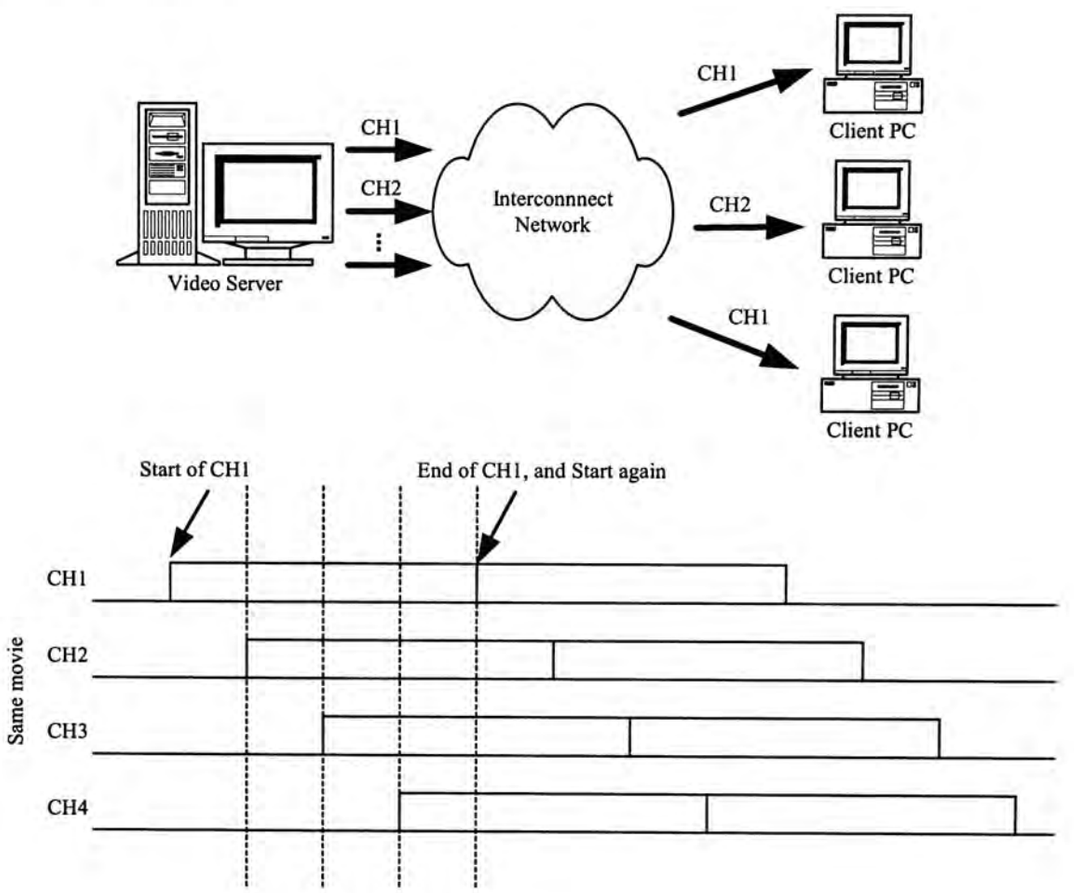

Figure 2.3: Channel Scheduling of NVoD

To start a new video session, a client simply waits for the next upcoming transmission cycle to begin playback. Given that the video is restarted once every 12 minutes, the average and worst-case waiting time will be equal to 6 minutes and 12 minutes respectively.

An important characteristic of NVoD is that when a new transmission cycle starts, all the waiting clients can share the multicast/broadcast channel and start video playback at the same time. This eliminates the need for allocating a dedicated channel to each video session and hence the resource requirement of a NVoD system is independent of the desired system capacity. In theory, a NVoD system can serve any number of users, albeit at the expense of longer start-up latency and limited interactivity.

2.3 Related Works

Apart from the TVoD and NVoD systems, researchers have recently investigated various new approaches to improve the efficient of VoD systems. In the following sections, we review some of these previous works that employ novel way of multicasting to achieve scalability but without the long startup latency commonly found in NVoD systems.

2.3.1 Batching

Batching is first proposed by Dan *et al.* [3] and later also investigated by Aggarwal *et al.* [5], Almeroth *et al.* [6] and Shachnai *et al.* [7]. The principle of batching is to group users waiting for the same movie at a video server and serve them using a single multicast channel. Unlike NVoD system, the schedule of multicast channels is not fixed but dynamically determined depending on the user

request pattern. Clearly this batching technique can reduce resource requirement both at the server and in the network. However, as multiple users share a multicast channel, individual user then cannot perform interactive VCR controls such as pause-resume, fast-forward, and fast-backward. To tackle this limitation, one can set aside some contingency channels to serve those users performing interactive controls as proposed and studied in Dan *et al.* [3], Almeroth *et al.* [6], and Li *et al.* [8].

There are several algorithms in which waiting users are scheduled for service in batching. For example, in FCFS batching [3], arriving users all join a single queue. Once a free channel becomes available in the server, the user at the head of queue will be served. Moreover, other queued users with the same movie selection will also be served together by the same multicast channel.

Another algorithm called Maximum Queue Length (MQL) [3], maintains a separate queue for each movie for the arriving users. Once a free channel becomes available, the movie with the maximum number of waiting users will be selected for service. This algorithm can improve batching efficiency at the expense of fairness as users waiting for unpopular movies are likely to experience longer waiting time than users waiting for popular movies. There are other more sophisticated batching algorithms and the interested readers are referred to the study by Shachnai *et al.* [7].

Their simulation results showed that resource reductions of up to 70% could be realized in large system serving around 5000 concurrent users, with an average waiting time of around one minute.

2.3.2 Patching

Hua *et al.* [10] proposed a *patching* approach where client caching and channel hopping are used to merge channels with near starting time to reduce resource requirement. Specifically, all channels in patching are multicast channels and a channel can function as either a regular channel or a patching channel. Depending on the arrival time and other system parameters, an arriving user may start the video session with a regular channel. Or it may start the video session using a patching channel while concurrently cache data from another regular channel. After a short time period, the user can then release the patching channel by continuing video playback via the cached data and the regular channel. Their simulation results show that at a latency of zero, patching can provide 300% system capacity compared to an equivalent TVoD system.

2.3.3 Mcache

Ramesh *et al.* recently proposed the multicast with cache (*Mcache*) approach [11] to further increase the performance of patching by using regional cache server. The regional cache server stores the first few minutes or even seconds, called *prefix*, of each video. Using the prefix from the cache server, a client can start video playback immediately while waiting for a free regular or patching channel. Hence by introducing the cache server, the latency of user request is reduced compared to pure patching. Simulation results [11] showed that at a latency of zero, Mcache with one-

minute prefix can provide 1300% system capacity compared to an equivalent TVoD system. The tradeoff is the additional costs for the cache server.

2.3.4 Unified VoD

Lee [15] recently proposed a UVoD architecture where both unicast and multicast channels are employed for video delivery. By using intelligent client buffering and channel switching, a UVoD system can achieve latency similar to TVoD while at the same time dramatically reduces the resource requirement.

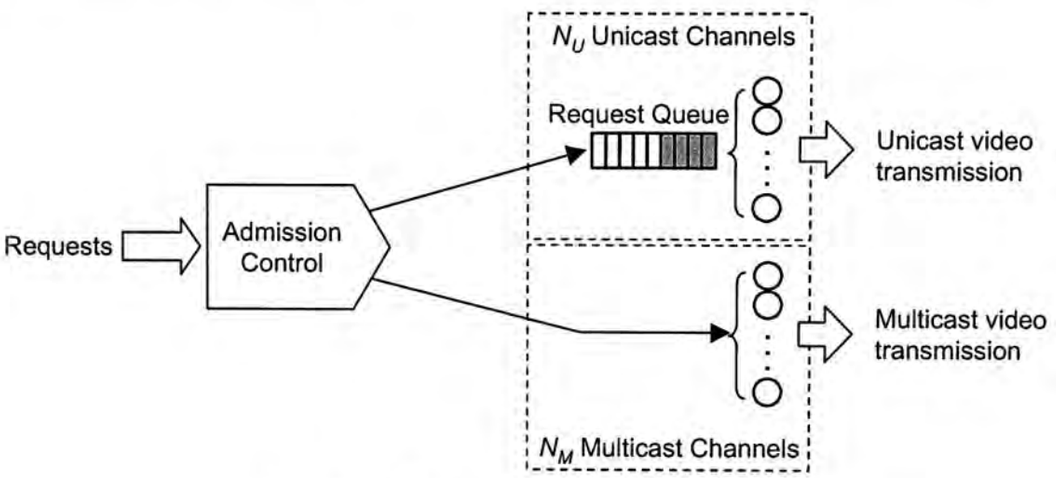

Figure 2.4: Architecture of the UVoD System

Figure 2.4 depicts the UVoD architecture. Let N be the total number of available channels, where N_u of them are unicast channels, $N_m = N - N_u$ are multicast channels. Let M be the number of movies of length L seconds each and movies are assumed to have the same length for simplicity. The multicast channels are evenly assigned to all movies and the architecture requires $N_m > M$ such that each movie can

be allocated with at least one multicast channel. Therefore, there are $\lfloor N_m/M \rfloor$ multicast channels allocated for each movie. The movie is then transmitted over all allocated multicast channels repeatedly as shown in Figure 2.5.

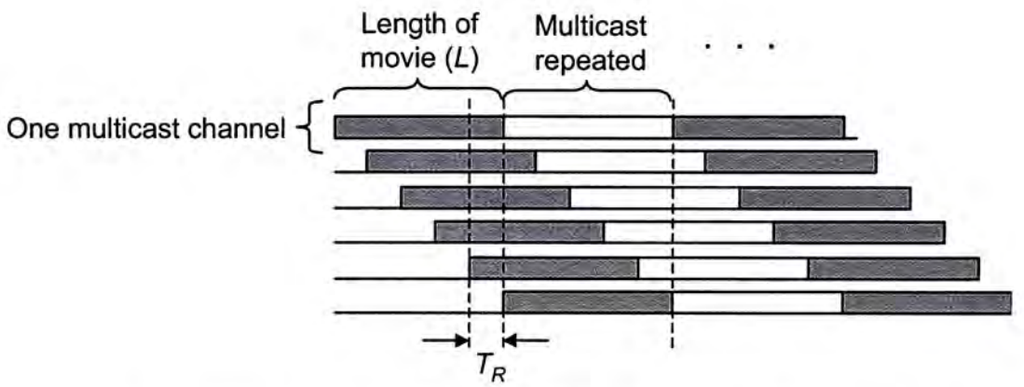

Figure 2.5: Scheduling of multicast cycles for a movie in UVoD

Note that transmission cycles are offset by

$$T_R = \frac{L}{\lfloor N_m/M \rfloor} \tag{2.1}$$

seconds between adjacent multicast channels allocated to the same movie.

When a user arrives at the system, the admission controller (Figure 2.4) will assign the user to wait for either a unicast channel (admit-via-unicast) or a multicast channel (admit-via-multicast) to begin playback. The purpose of the admission controller is to equalize the average waiting time for users served initially by unicast channel and multicast channel [15]. Specifically, a parameter called admission threshold, denoted by δ, is introduced for admission control. Let t be the time a user arrives at the system requesting movie i, and let t_m be the start time of the next multicast cycle for movie i. The system will assign the user to wait for the upcoming

multicast cycle if the resultant waiting time will be smaller than the admission threshold:

$$(t_m - t) \leq \delta \tag{2.2}$$

The admitted user continues to receive video data from this multicast channel for the entire video playback as in a NVoD system.

If the resultant waiting time is longer than the admission threshold, the admission controller will assign the user to wait for a free unicast channel to begin video playback. All unicast channels share a single input queue as shown in Figure 2.4 and waiting users are served according to the first-come-first-serve (FCFS) queueing discipline. By adjusting the admission threshold, the system can maintain similar latency for both admit-via-multicast and admit-via-unicast users.

For admit-via-unicast users, the client device first starts caching video data from the previous multicast of the requested movie. Then it waits for a free unicast channel to start playback. For example, assuming that the user arrives at time t, and let t_{m-1} and t_m be the nearest epoch times of multicast channel $m-1$ and channel m, for which $t_{m-1} < t < (t_m - \delta)$. Then at time t, the client starts caching video data from channel $m-1$ into the client's local storage. At the same time, the client enters the request queue and starts video playback using unicast once a free unicast channel becomes available. Therefore UVoD assumes that the client devices can receive two video channels simultaneously and have local storage to cache up to T_R seconds of video data.

The admission process is not yet completed as the client still occupies one unicast channel. As the client concurrently caches multicasted video data for the movie starting from movie time ($t-t_{m-1}$), the unicast channel can be released after a time ($t-t_{m-1}$) and the client can continue video playback using the local cache. Hence similar to Liao *et al.* [4], the local cache is used to add time delay to the multicast video stream so that it can be synchronized with the client playback. Since $0<(t-t_{m-1})<(T_R-\delta)<<L$, we can see that the unicast channels are occupied for much shorter duration when compared to TVoD. This reduction in service time allows more requests to be served by the unicast channels.

The analytic show that the 500% performance gain of UVoD to TVoD architecture can be achieved at a latency requirement of one second. In the meanwhile, this architecture provides *pause/resume* interactive control without incurs any additional system load. The optimum channel allocation on UVoD for multicast and unicast channel is depends on the system load, and the number of multicast channel increase as the increment in the system load.

2.4 Discussions

Reconsidering the UVoD architecture, we observe that the unicast channels within UVoD operate in the same way as a TVoD system. The difference is that a unicast channel in UVoD is not occupied for the entire duration of the movie. The channel will be released once the user is merged back to a stagger multicast channel.

This motivates us to investigate applying algorithm similar to batching to these unicast channels to further improve the system performance. This new Super-Scalar Video-on-Demand (SS-VoD) architecture will be presented in the next chapter.

Chapter 3

Super-Scalar Architecture

In this section we present the architecture and the motivation for the proposed Super-Scalar Video-on-Demand System (SS-VoD). The overall SS-VoD architecture is depicted in Figure 3.1. The system comprises a number of service nodes connected via a multicast-ready network to the clients. The clients form clusters according to their geographical proximity. An admission controller in each cluster performs authentication as well as scheduling requests forwarding to the service nodes.

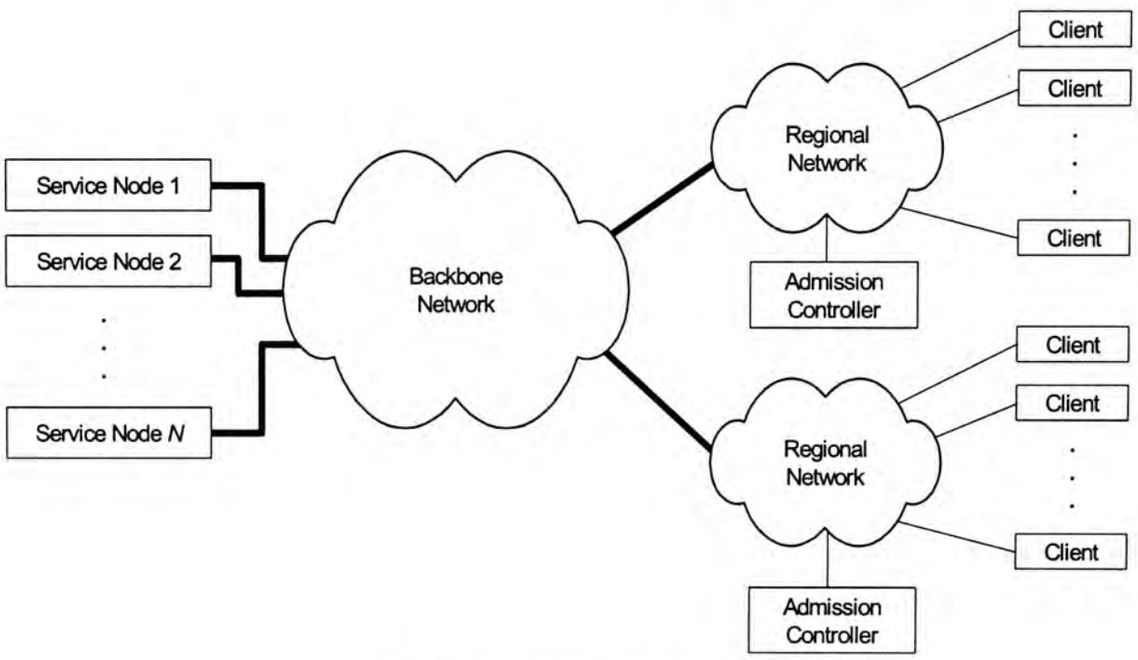

Figure 3.1: System architecture.

Each service node operates independently from each other, having its own disk storage, memory, CPU, and network interface. Hence a service node is effectively a mini video server, albeit serving a small number of video titles to the *entire* user population. This modular architecture can simplify the deployment and management of the system. For example, since the configuration of each node is decoupled from the scale of the system and each server node carries just a few movies, a service provider simply deploys the right number of server nodes according to the desired video selections. Additional server nodes can be added when more movie selections are needed, with the existing nodes remain unchanged.

To improve reliability, one can use disk mirroring for each server node. While parity-based schemes [16-18] have lower redundancy overhead, the number of disks involved present too much storage capacity for use in a server node, where only a

few movie is served. Additionally, mirroring greatly simplifies recovery from a disk failure as a failed disk can easily be replaced without the need to shutdown the server node (e.g. with hot-swap disks). Server performance can also be maintained despite a disk failure and the failed disk can be rebuilt off-line simply by reloading movie data from backup storage.

In case of a complete node failure, the service provider can simply pull the disks from the failed node and install then into a spare node. The recovery time can be made very short and only users currently viewing movies served by the failed node will be affected.

SS-VoD achieves scalability and bandwidth efficiency with two techniques. The first technique is through the use of multicast to serve multiple clients using a single multicast channel. However, simple multicast such as those used in a near-VoD (NVoD) system limits the time for which a client may start a new video session. Depending on the number of multicast channels allocated for a video title, this startup delay can range from a few minutes to tens of minutes. To tackle this initial delay problem, we make use of the second technique: the use of client-side caching together with channel merging, to allow a client to start video playback at any time using a *bridging* channel until it can be merged back to an existing multicast channel. The following sections present these techniques in detail.

3.1 Transmission Scheduling

Each service node in the system streams video data into multiple multicast channels. Let M be the number of video titles served by each service node and let N be the total number of multicast channels available to a service node. For simplicity, we assume N is divisible by M and hence each video title is served by the same number of multicast channels, denoted by $N_M=N/M$. These multicast channels are then divided into two groups of N_S static multicast channels and $N_D=N_M-N_S$ dynamic multicast channels.

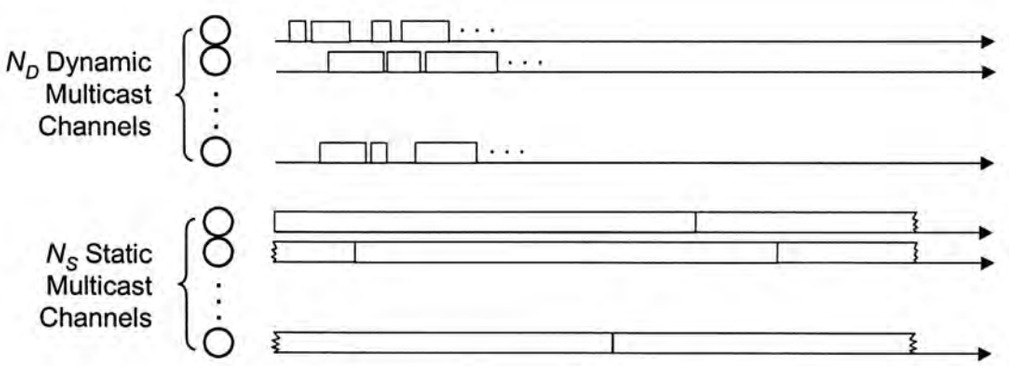

Figure 3.2: Transmission schedule for one movie.

The video title is repeatedly multicasted over all N_S static multicast channels in a time-staggered manner as shown in Figure 3.2. Specifically, adjacent channels are offset by

$$T_R = \frac{L}{N_S} \tag{3.1}$$

seconds, where L is the length of the video title in seconds. Transmissions are continuously repeated, i.e. restart from the beginning of a video title after transmission completes, regardless of the load of the server or how many users are active. These static multicast channels are used as the main channel for delivering video data to the clients. A client may start out with a dynamic multicast channel but it will shortly be merged back to one of these static multicast channels to continue the video session until completion. The next section presents the admission procedure for starting a new video session and we explain in Chapter 3.3 how the client is merged back to one of the static multicast channels.

3.2 Admission Control

To reduce the response time while still leveraging the bandwidth efficiency of multicast, SS-VoD allocates a portion of the multicast channels and schedules them dynamically according to the requests arrival patterns.

Specifically, a new request first goes to the admission controller, which first performs authentication of the client. Armed with complete knowledge of the transmission schedules for the static multicast channels, the admission controller then determines if the new user should wait for the next upcoming multicast transmission from the static multicast channels, or start playback with a dynamic multicast channel. In the former case, the client just waits for the next multicast cycle to begin, without incurring any additional load to the backend service nodes. In the latter case,

the admission controller then performs additional processing to determine if a new request needs to be sent to the appropriate service node to start a new dynamic multicast stream.

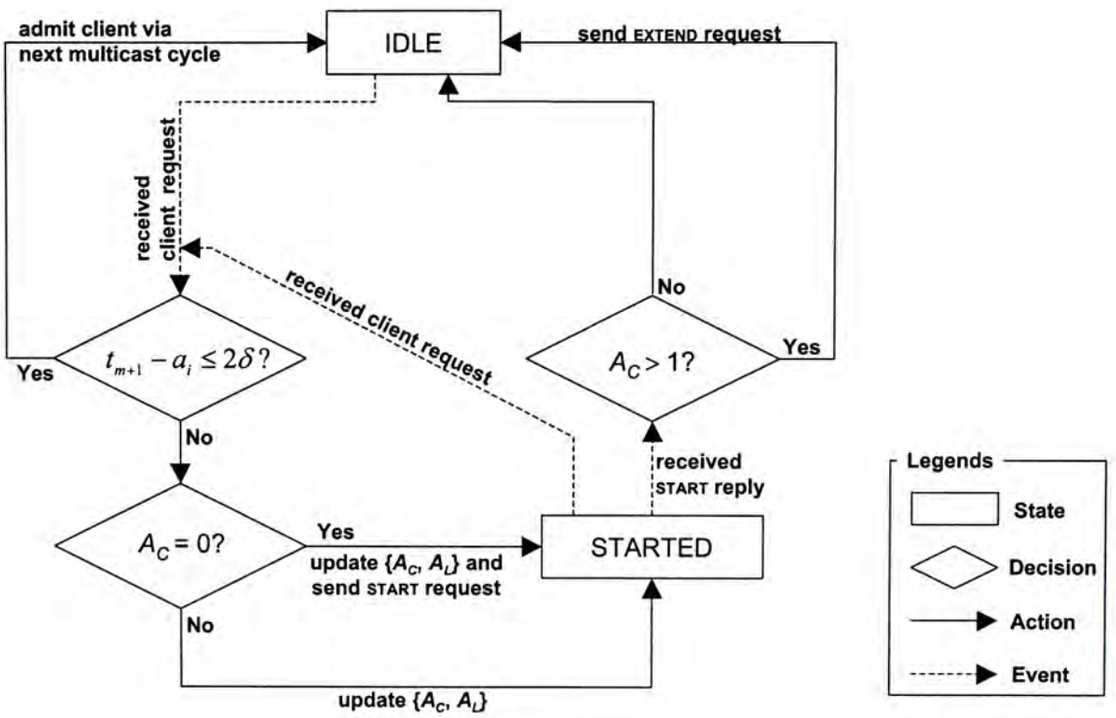

Figure 3.3: State-transition diagram for the admission controller.

Figure 3.3 depicts the state-transition diagram defining the admission procedure. Beginning from the IDLE state, suppose that a new request arrives at time a_i, which is between the start time of the previous multicast cycle, denoted by t_m, and the start time of the next multicast cycle, denoted by t_{m+1}. Now a predefined admission threshold, denoted by δ, determines the first admission decision made by the admission controller: the new request will be assigned to wait for the next multicast

cycle to start playback if the waiting time, denoted by w_i, is equal to or smaller than 2δ, i.e.

$$w_i = t_{m+1} - a_i \leq 2\delta \tag{3.2}$$

We call these requests *statically-admitted* and the admission controller returns to the IDLE state afterwards. For a randomly arrived request (e.g. Poisson arrivals) that is statically-admitted, the waiting time is uniformly distributed between zero and 2δ, with a mean waiting time of δ. This admission threshold is introduced to reduce the amount of load going to the dynamic multicast channels. Configuration of this admission threshold will be presented in Chapter 4.

If (2) does not hold, then the admission controller will proceed to determine if a request needs to be sent to the appropriate service node to start a new dynamic multicast stream – *dynamically-admitted.* The service nodes and admission controllers each keeps a counter and a length tuple: $\{A_C, A_L\}$, where A_C=0,1,…, and $0 \leq A_L \leq (T_R - 2\delta)$, for each video title being served. Therefore each service node will have M such admission tuples and each admission controller will have MK such admission tuples, where K is the total number of service nodes in the system. Both the counter and the length fields are initially set to zero.

Now with the admission tuples, the admission procedure proceeds as follows. For requests that cannot be statically-admitted, the admission controller will first check the counter in the admission tuple for the requested video title. If the counter A_C is zero, then the counter is increased by one, and the length field is set according to

$$A_L = a_i - t_m \tag{3.3}$$

which is essentially the length of time passed since the beginning of the last multicast. At the same time a START request will be sent to the service node with the requested video title and the admission controller enters the STARTED state.

If another request for the same video title arrives during the STARTED state, say at time a_{i+1}, the admission controller will not send another request to the service node, but just update the length field according to

$$A_L = a_{i+1} - t_m \tag{3.4}$$

This process repeats for any additional requests arrived during the STARTED state.

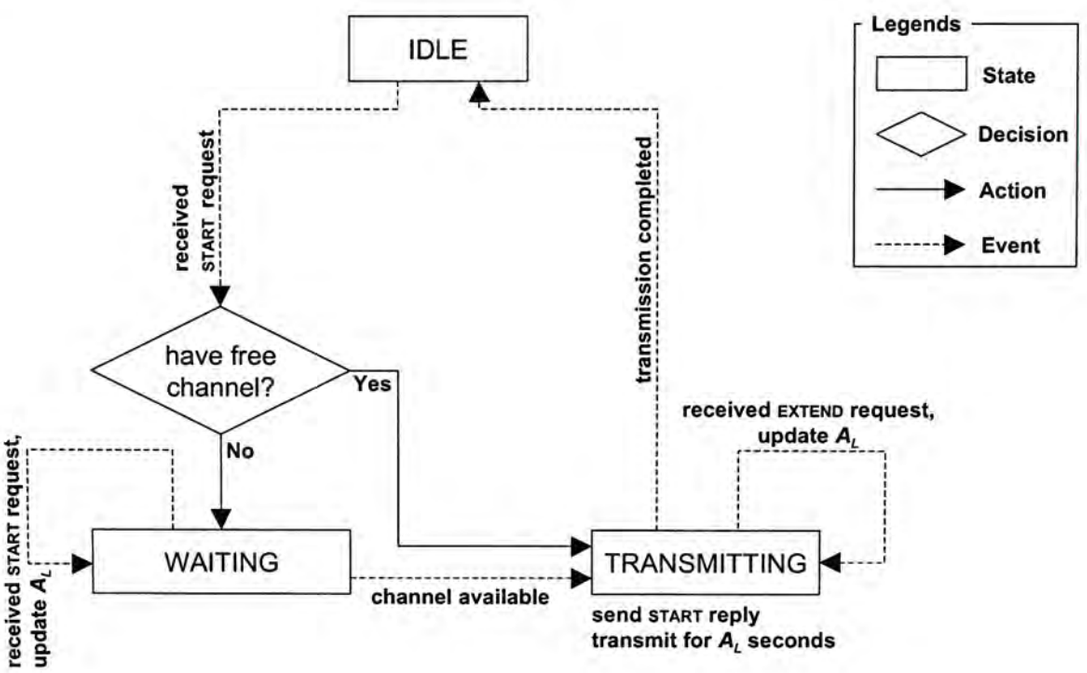

Figure 3.4: State-transition diagram for the service nodes.

At the service node side, upon receiving a START request from the admission controller, the service node will attempt to allocate a channel from the N_D dynamic multicast channels to start transmitting the video title for a duration of A_L seconds as shown in Figure 3.4. If the allocation is successful, i.e. free channels are available, then the counter and the length fields are zeroed and a START reply sent back to all admission controllers to announce the commencement of the new transmission. Otherwise it will wait for a free channel to be released.

The admission controllers, upon receiving the START reply, will do one of two things. If the local counter value is one, then both the counter and the length fields are zeroed and the admission process completes. Otherwise, i.e. the counter is larger than one, the admission controller will send an EXTEND request to the service node to extend the transmission duration according to the value of the local length A_L. Note that in this case, the length field at the admission controller will be larger than the length field at the service node because only the length field at the admission controller is updated for subsequent requests for the same video title. The length field at the service node is always the one for the first request. Upon receiving the EXTEND requests, the service node will update the interval transmission duration to the largest one among all EXTEND requests. Transmission will stop after the specified transmission duration expires.

It may appear that the previous admission procedure is unnecessarily complex and the clients can better-off send requests directly to the service nodes. However, this direct approach suffers from poor scalability. In particular, recall that each

service node serves a few video titles to the entire user population. Therefore as the user population grows, the volume of requests directed at a service node will increase linearly and eventually exceed the service node's processing capability.

By contrast, the admission controller generates at most two requests, one START request and one EXTEND request, for each dynamically-started multicast transmission, irrespective of the actual number of client requests. Since the numbers of admission controllers are orders of magnitude smaller than the user population, the processing requirement at the service nodes can then be substantially reduced. For extremely-large user populations where even requests from admission controllers can become overwhelming, one can extend this request-consolidation strategy in a hierarchical manner by introducing additional layers of admission controllers to further consolidate the requests until the volume becomes manageable by the service nodes.

3.3 Channel Merging

According to the previous admission control policy, a statically-admitted client starts receiving streaming video data from a static multicast channel for playback which is depicted in Figure 3.5. For dynamically-admitted clients, video playback starts with video data received from a dynamically-allocated multicast channel. To prepare merging the client back to an existing static multicast channel, the client concurrently receives and caches video data from a nearby (in time) static multicast channel as illustrated in the timing diagram in Figure 3.6.

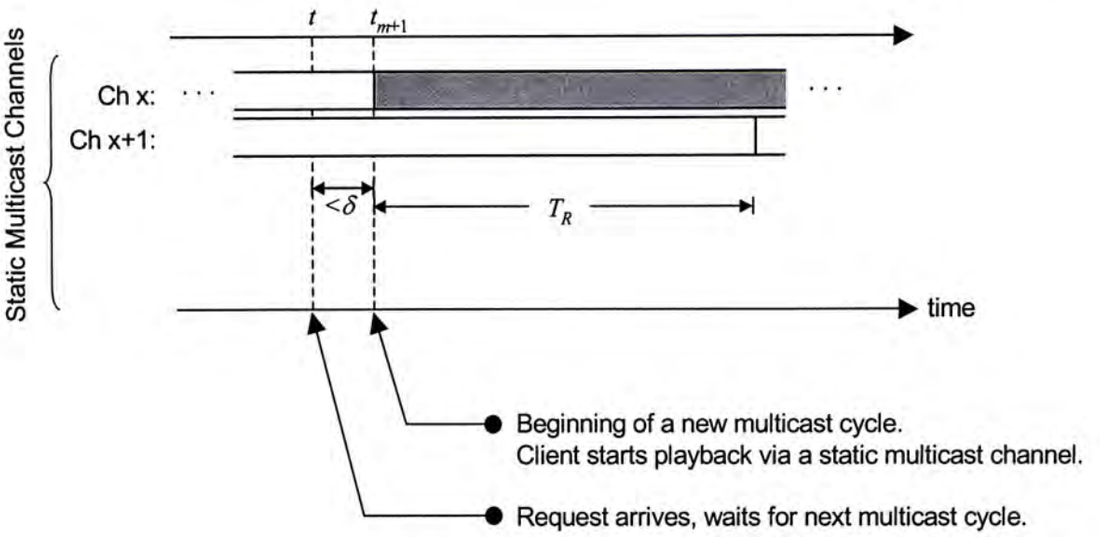

Figure 3.5: Statically-admitted

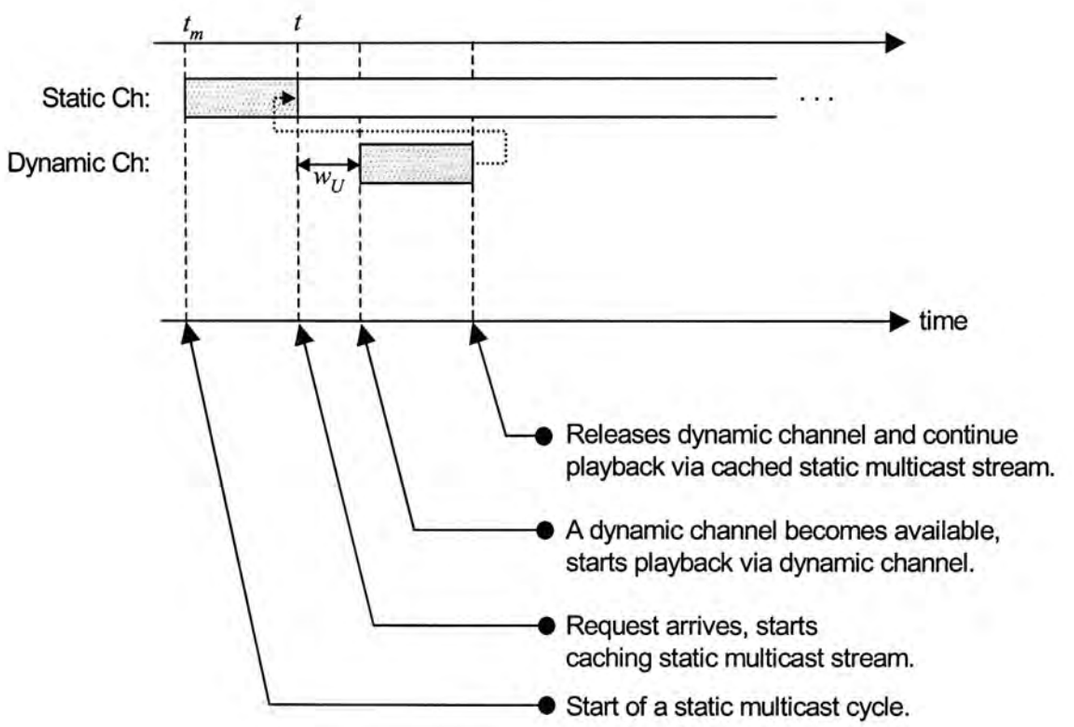

Figure 3.6: Dynamically-admitted

Since the dynamic multicast channel will cease transmission after a time A_L, a dynamically-admitted client will concurrently receive streaming video data from another static multicast channel and store them locally either in memory or in the harddisk. The goal here is to use the cached video data to continue playback after the dynamic multicast channel is released.

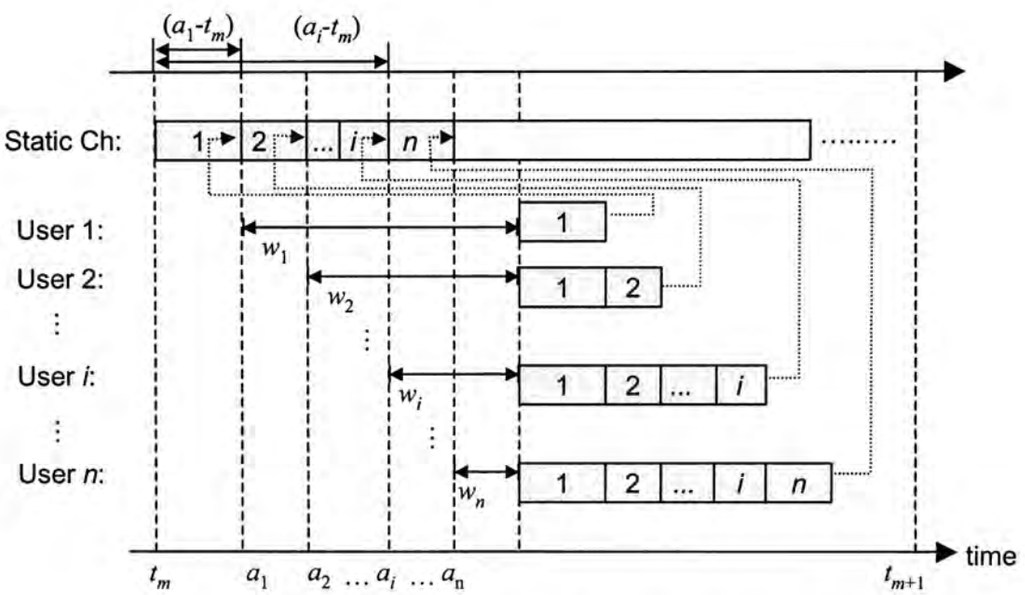

Figure 3.7: Channel Mergence of multi-user on SS-VoD

As an illustration, consider a dynamic multicast channel serving n dynamically-admitted clients. Let a_i be the time client i arrives at the system and the nearest multicast cycle starts at t_m and t_{m+1} respectively, where $t_m < a_1 < a_2 \ldots < a_n < (t_{m+1} - \delta)$. Client i will cache the data from a_i at the proper static multicast channel while waiting for the upcoming dynamic channel. Then client i will leave the dynamic channel to merge back to the static multicast channel after a service time of $(a_i - t_m)$ as shown in Figure 3.7. The dynamic multicast channel can be released once

all n clients are merged back to the static multicast channel. Therefore the holding time of the dynamic channel is simply the maximum of $(a_i - t_m)$, i=1,2...n. As a consequence, the holding time is simply equal to the latest client joined to the system.

3.4 Interactive Control

In a conventional VoD system, the major types of interactive control are fast forward/backward, pause/resume, and stop. Among them, pause/resume is the most common control performed in movie-on-demand applications.

Intuitively, performing an interactive control essentially breaks the client away from the current static multicast video stream, and then restarts it at the same point within the video stream. Under this view, interactive control is no different from a new request and hence can be served the same way as for a new-video request. Obviously this approach will increase loads at the dynamic multicast channel, which could increase waiting time for both new and interactive request. As there is no generally accepted user-activity model, we do not attempt to quantify the performance impact of this approach.

Due to the static channel allocation employed in SS-VoD, we can devise a channel hopping algorithm to support pause-resume control without incurring additional load at the unicast channels. Specifically, each movie is multicasted every T_R seconds and the client has a buffer large enough to cache T_R seconds of video. When a user pauses, say at a movie time t_p, the client just continue to buffer the

incoming video data. If the user resumes playback before buffer overflows, then nothing needs to be done. Otherwise, the client just stops buffering and enters and idle state once the buffer is full (ie. storing the movie segment from t_p to (t_p+T_R)). When the user later resumes playback, the client can resume playback immediately and at the same time determine the nearest multicast channel that is currently multicasting the movie at the movie time $t_m \geq t_p$. Since a movie is repeated multicasted every T_R seconds, we have $(t_m - t_p) \leq T_R$. Hence the client just needs to start buffering again after the selected channel reaches movie time (t_p+T_R). This channel-hopping algorithm is unique in the sense that no additional resource is required at the server. Pause-resume is simply supported by buffering and switching of multicast channel at the appropriate time. Hence, SS-VoD is particularly suitable for movie-on-demand applications where pause-resume is the primary interactive control needed.

Chapter 4

Performance Modeling

In this section we present an approximate performance model for the SS-VoD architecture. While exact analytical solution does not appear to be tractable, conventional numerical methods can be applied to obtain performance results based on the approximated model. The purpose of this performance model is to assist system designers to quickly evaluate various design options and to perform preliminary system dimensioning. Once the approximate system parameters are known, one can resort to a more detailed simulation to obtain more accurate performance results.

The primary performance metric we use in this thesis is average waiting time, defined as the time a client submitted a request to the admission controller to the time the beginning of the requested video starts streaming. For simplicity, we ignore network delay, transmission loss, and processing time at the admission controller. We further assume that there is a single movie stored in a service node. We will investigate multiple-movie cases in Chapter 5.6.

In the following sections, we will first derive the average waiting time for statically-admitted clients and dynamically-admitted clients, and then investigate the channel partitioning problem. We will compare results computed using this approximate performance model with the simulation results in Chapter 5.1.

4.1 Waiting Time for Statically-Admitted Clients

As described in Chapter 3.2, there are two ways where a client can be admitted to the system to start a video session. The first way is admission through a static multicast channel as shown in Figure 3.5. Given that any clients arriving within the time window of 2δ seconds will be admitted this way, it is easy to see that the average waiting time for statically-admitted clients, denoted by $W_S(\delta)$, is equal to half of the admission threshold:

$$W_S(\delta) = \delta \tag{4.1}$$

assuming it is equally probable for a request to arrive at any time within the time window.

4.2 Waiting Time for Dynamically-Admitted Clients

The second way to admit a new client is through a dynamic multicast channel as shown in Figure 3.6. Unlike static multicast channels, dynamic multicast channels are allocated in an on-demand basis according to the admission procedure described in Chapter 3.2. Specifically, if there are one or more free channels available at the time a request arrives, a free channel will be allocated to start transmitting video data to the client immediately and the resultant waiting time will be zero.

On the other hand, if there is no channel available at the time a request arrives, then the resultant waiting time will depend on when a request arrive and when a free dynamic multicast channel becomes available. Specifically, requests arriving at the admission controller will be consolidated using the procedure described in Chapter 3.2 where the admission controller will send a consolidated START request to a service node to initiate video transmission.

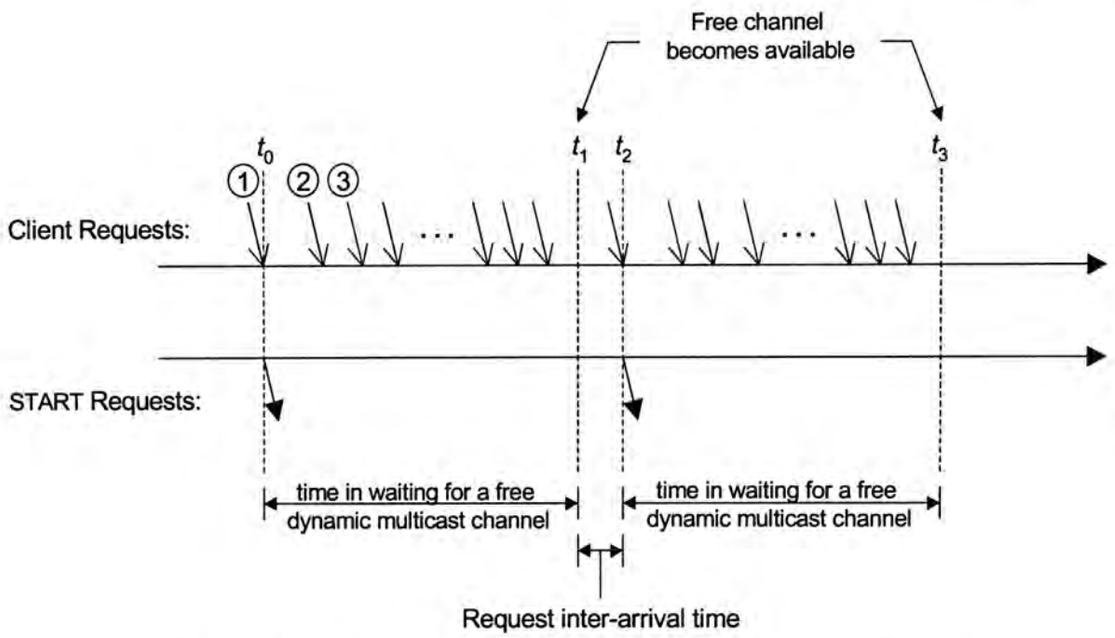

Figure 4.1: User classification in dynamic multicast channel

Figure 4.1 illustrates this admission process. This example assumes that there is no request waiting and all dynamic multicast channels are occupied before client request 1 arrives. After receiving request 1, the admission controller sends a START request to a service node to initiate a new multicast transmission for this request. However as all channels are occupied, the transmission will not start until a later time t_1 when a free channel becomes available. During this waiting time, additional client requests such as request 2, 3, and so on arrives but the admission controller will not send additional START request to the service node. This process repeats when a new request arrives at time t_2.

Based on this mode, we first derive the average waiting time experienced by a START request at the service node. For the arrival process, we assume that user

requests form a Poisson arrival process with rate λ. The proportion of client requests falls within the admission threshold δ is given by

$$P_S = \frac{2\delta}{T_R} \tag{4.2}$$

and these clients will be statically-admitted.

Correspondingly, the proportion of dynamically-admitted clients is equal to (1–P_S). We assume the resultant arrival process at the admission controller is also Poisson, with a rate equal to

$$\lambda_D = (1 - P_S)\lambda \tag{4.3}$$

Referring to Figure 4.1, we observe that the time between two adjacent START requests is composed of two parts. The first part is the waiting time for a free dynamic multicast channel; and the second part is the inter-arrival time for dynamically-admitted client requests. Let $W_C(\delta)$ be the average waiting time for a free dynamic multicast given δ. Then the inter-arrival time for START requests will be given by

$$\frac{1}{\lambda_S} = W_C(\delta) + \frac{1}{\lambda_D} \tag{4.4}$$

where λ_S is the arrival rate for START requests. For simplicity, we assume that the arrival process is Poisson.

For the service time of START request, it depends on the last user joined to the system as shown in Figure 3.7. In particular, the service time of the last user equals to the arrival time a_n minus the time t_{m-1} for the previous multicast of the requested

movie. The service time, denoted by s, can range from 0 to $(T_R-2\delta)$. We assume the service time s is uniformly distributed between

$$0 < s < T_R - 2\delta \tag{4.5}$$

Therefore the dynamic multicast channels form a multiserver queueing system with Poisson arrival and uniformly-distributed service time. As there is no close-form solution for such queueing model, we resort to the approximation by Allen and Cunneen [19] for G/G/m queues to obtain the average waiting time for a dynamic multicast channel:

$$W_C(\delta) = \frac{E_C(N_D,u)}{N_D(1-\rho)}\left(\frac{C_A^2 + C_S^2}{2}\right)T_S \tag{4.6}$$

where $C_A^2 = 1$ is the coefficient of variation of Poisson process,

$$C_S^2 = \frac{(T_R - 2\delta)^2}{12}\left(\frac{2}{T_R - 2\delta}\right)^2 = \frac{1}{3} \tag{4.7}$$

is the coefficient of variation for uniformly-distributed service time, and T_S is the average service time, given by

$$T_S = \frac{T_R - 2\delta}{2} \tag{4.8}$$

Additionally, $u = \lambda_S T_S$ is the traffic intensity, $\rho = u/N_D$ is the server utilization, and $E_C(N_D,u)$ is the Erlang-C function:

$$E_C(N_D,u) = \frac{u^{N_D}/N_D!}{u^{N_D}/N_D! + (1-\rho)\sum_{k=0}^{N_D-1}\frac{u^k}{k!}} \tag{4.9}$$

Since the traffic intensity depends on the average waiting time, and the traffic intensity is needed to compute the average waiting time, Equation (4.6) is in fact defined recursively. Due to (4.9), Equation (4.6) does not appear to be analytically solvable. Therefore, we use numerical methods in solving for $W_C(\delta)$ in computing the numerical results in Chapter 5.

Now that we have obtained the waiting time for a START request, we can proceed to compute the average waiting time for dynamically-admitted client requests. Specifically, we assume the waiting time for START request is exponentially distributed with mean $W_C(\delta)$. We classify client requests into two types. A Type-1 request is the first request that arrives at the beginning of the admission cycle. Type-2 requests are the other requests that arrive after a Type-1 request. For example, request 1 in Figure 4.1 is a Type-1 request, and request 2 and 3 are Type-2 requests.

We first derive the average waiting time for Type-2 requests. Let $W_2(\delta)$ be the average waiting time for Type-2 requests which can be found to be (see Appendix):

$$W_2(\delta) = W_C(\delta)\left(1 - \frac{1 + \dfrac{(T_R - 2\delta)}{2W_C(\delta)}}{1 - e^{\frac{-(T_R-2\delta)}{W_C(\delta)}}} \frac{(T_R - 2\delta)}{W_C(\delta)} e^{\frac{-(T_R-2\delta)}{W_C(\delta)}}\right) \tag{4.10}$$

Next for Type-1 requests, the average waiting time, denoted by $W_1(\delta)$, is simply equal to $W_C(\delta)$. Therefore the overall average waiting time can be computed from a weighted average of both Type-1 and Type-2 requests. Specifically, the average

number of Type-2 requests arriving in an admission cycle, denoted by $M_2(\delta)$, can be computed from

$$M_2(\delta) = W_C(\delta)\lambda_D \tag{4.11}$$

Let $W_D(\delta)$ be the average waiting time for all Type-1 and Type-2 requests. We can then compute it from the weighted average of both Type-1 and Type-2 average waiting times:

$$\begin{aligned} W_D(\delta) &= \frac{W_1(\delta) + M_2(\delta)W_2(\delta)}{1 + M_2(\delta)} \\ &= \frac{W_1(\delta) + W_C(\delta)\lambda_D W_2(\delta)}{1 + W_C(\delta)\lambda_D} \end{aligned} \tag{4.12}$$

4.3 Admission Threshold

In the previous derivations, we have assumed that the admission threshold value is given *a priori*. Consequently, the resultant average waiting time for statically-admitted and dynamically-admitted users may differ. To maintain a uniform average waiting time in both cases, we can adjust the admission threshold according to the average waiting time at the unicast channels:

$$\delta = \min\{x \mid (W_S(x) - W_D(x)) \le \varepsilon, T_R \ge x \ge 0\} \tag{4.13}$$

so that the waiting-time differences are less than some small value ε.

As adjusting the admission threshold does not affect existing users, the adjustment can be done dynamically while the system is online. In particular, the system can maintain a moving average of previous users' waiting time as the

reference for threshold adjustment. This enables the system to maintain a uniform waiting time, referred to as latency thereafter, for both statically-admitted and dynamically-admitted users.

4.4 Channel Partitioning

An important configuration of SS-VoD is partitioning of available channels for use as dynamic and static multicast channels. Intuitively, having too many dynamic multicast channels will increase the traffic intensity at the dynamic multicast channels due to increases in the service time (c.f. Equations (3.1) and (4.4)). On the other hand, having too few dynamic multicast channels may also result in higher load at the dynamic multicast channels.

Similar to the study by Lee [15] on UVoD, an optimal channel partitioning policy can be obtained by enumerating all possibilities, which in this case is of O(N). Unlike UVoD, we found that the optimal channel partitioning policy is relatively independent of the user arrival rate. See Chapter 5.2 for more details.

Chapter 5

Performance Evaluation

In this section, we present simulation and numerical results to evaluate the SS-VoD architecture studied in this thesis.

5.1 Model Validation

To verify the accuracy of the performance model derived in Section IV, we developed a simulation program using CNCL [20] to obtain simulation results for comparison. A set of simulations is run to obtain the average waiting over a range of arrival rates. Each run simulates a duration of 1440 hours (60 days), with the first 24 hours of data skipped to reduce initial condition effects. There is one movie in the system, with a length of 120 minutes. We divide available multicast channels equally into static-multicast and dynamic-multicast channels. We do not simulate user interactions and assume all users playback the entire movie from start to finish.

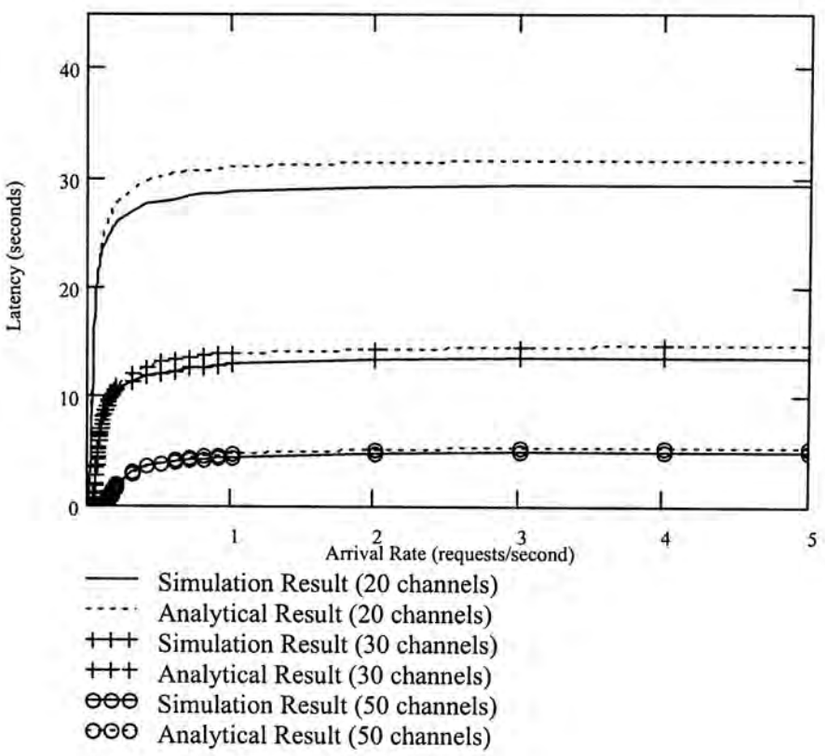

Figure 5.1: Latency Comparison of analytic and simulation results

Figure 5.1 shows the average waiting time versus arrival rate ranging from 1×10^{-3} to 5.0 requests per second. We observe that the analytical results are reasonable approximations for the simulation results. At high arrival rates (e.g. over 1 requests per second), the analytical results over-estimate the simulation results by up to 5%.

As discussed in the beginning of Chapter 4, the analytical model is primary used for preliminary system dimensioning. Detailed simulation, while lengthy (e.g. hours), is still required to obtain accurate performance results.

5.2 Channel Partitioning

To investigate the performance impact of different channel allocations, we conducted simulations with proportion of dynamic multicast channels, denoted by r, ranging from 0.3 to 0.7. The results are plotted in Figure 5.2. Note that we use a normalized latency instead of actual latency for the y-axis to facilitate comparison. Normalized latency is defined as

$$\frac{w(r)}{\min\{w(r), \forall r\}} \tag{5.1}$$

where $w(r)$ is the latency with r_N dynamic multicast channels.

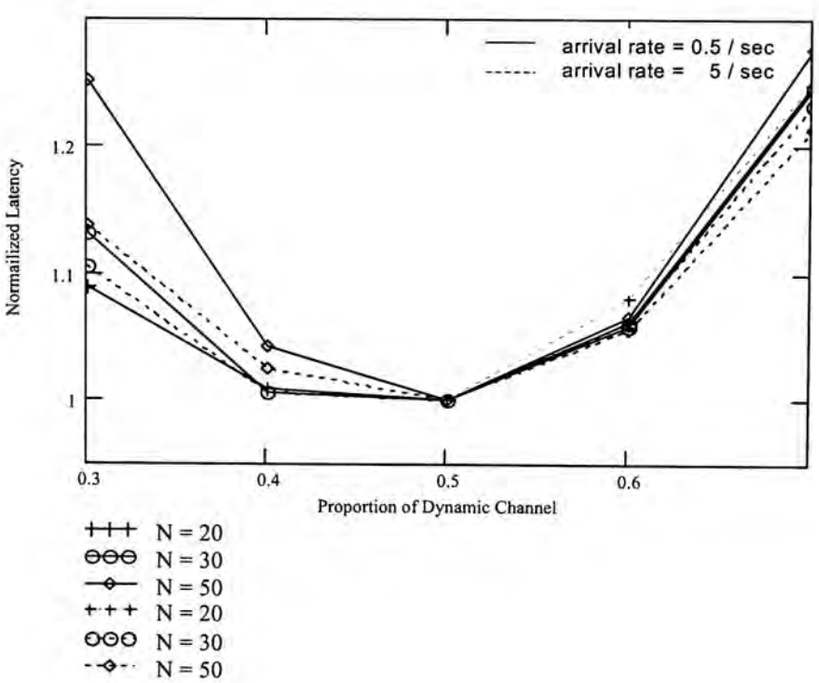

Figure 5.2: Normalized latency versus proportions of dynamic multicast channel

We simulated three sets of parameters with N=20, 30, and 50 for two arrival rates, namely heavy load at 5 requests/second and light load at 0.5 requests/second. Surprisingly, the results show that the latency is minimized by assigning half of channels to dynamic multicast and the other half to static multicast. By contrast, UVoD [15] exhibits a different behavior and requires more channels allocated to static multicast channels to minimize latency at high loads. Figure 5.3 compares the optimal channel allocation for UVoD and SS-VoD for a 50-channel configuration. For UVoD, the optimal proportion of static multicast channel increases with the arrival rate. For example, the optimal proportion of static multicast channel is 86% (43 static multicast channels) at an arrival rate of one user per second. For comparison, the optimum proportion of static multicast channel for SS-VoD remains 50% for the entire range of loads in this example.

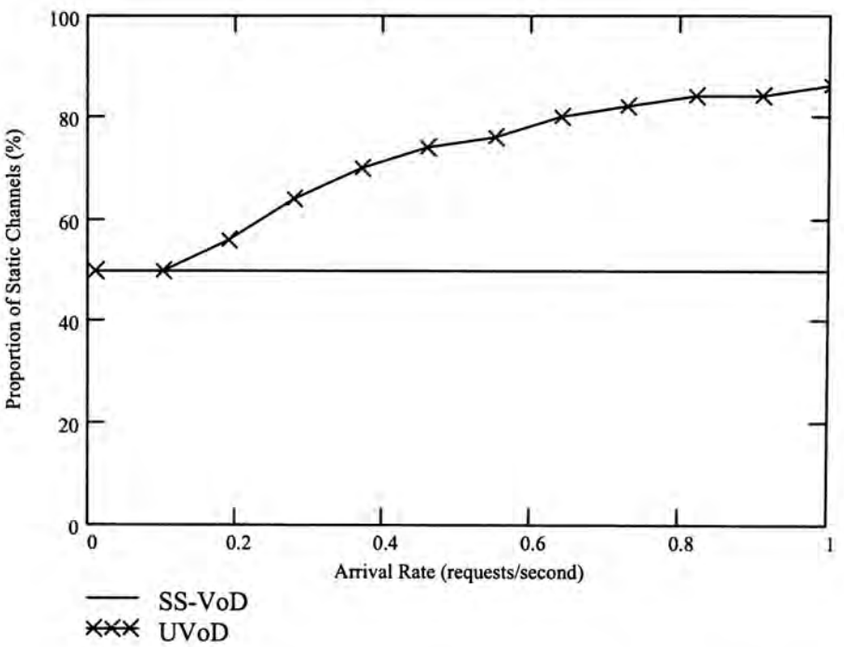

Figure 5.3: Optimum Channel Allocation of SS-VoD and UVoD

The channel reallocation scheme affects the behavior of admitted users so the complexity of practical system implementation increases in UVoD. In contract, the optimal channel allocation of SS-VoD is simply independent with the system load for a wide range of loads. This property greatly enhances the practicability of SS-VoD system.

5.3 Latency Comparisons

Figure 5.4 plots the latency for SS-VoD, UVoD, TVoD, and NVoD for arrival rates up to 5 requests per second. The service node (or video server for TVoD/NVoD) has 50 channels and serves a single movie of length 120 minutes. The first

observation is that except for NVoD, which has a constant latency of 72 seconds, the latency generally increases with higher arrival rates as expected. For TVoD, the server overloads for arrival rates larger than 1.16×10^{-4} requests per second. UVoD performs significantly better with the latency asymptotically approaches that of NVoD. SS-VoD performs even better with the latency level off and approaches 5.6 seconds, or a 92% reduction compared to UVoD.

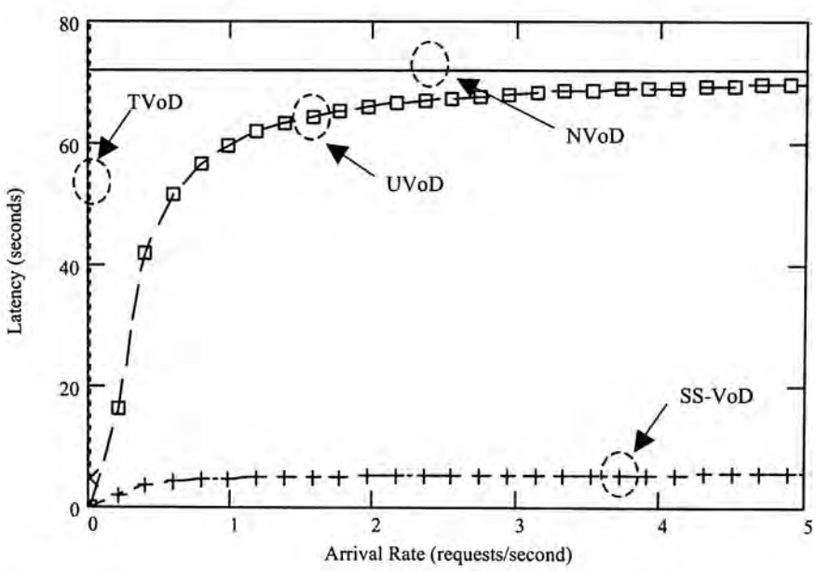

Figure 5.4: Latency comparison of SS-VoD with typical VoD Systems

It is worth noting that the performance gain of SS-VoD over UVoD does not incur any tradeoff at the client side. Specifically, the buffer requirement and bandwidth requirement are the same for both SS-VoD and UVoD. The only differences are the replacement of the unicast channels in UVoD with multicast channel; and the more complex admission procedure in the admission controller.

5.4 Channel Requirement

To investigate the channel requirement for a range of arrival rate, Figure 5.5 plots the channel requirement of SS-VoD, UVoD, TVoD, and NVoD in log-scale versus the arrival rates from 0.01 to 5 requests per second. There is a single movie of length 120 minutes. The latency constraint is equal to or shorter than one second. The channel requirement in the y-axis, denoted by C, is computed from

$$C = \min\{n \mid W \leq 1, \forall n = 0,1,\ldots\} \tag{5.2}$$

where W is the waiting time for the systems at the given arrival rate.

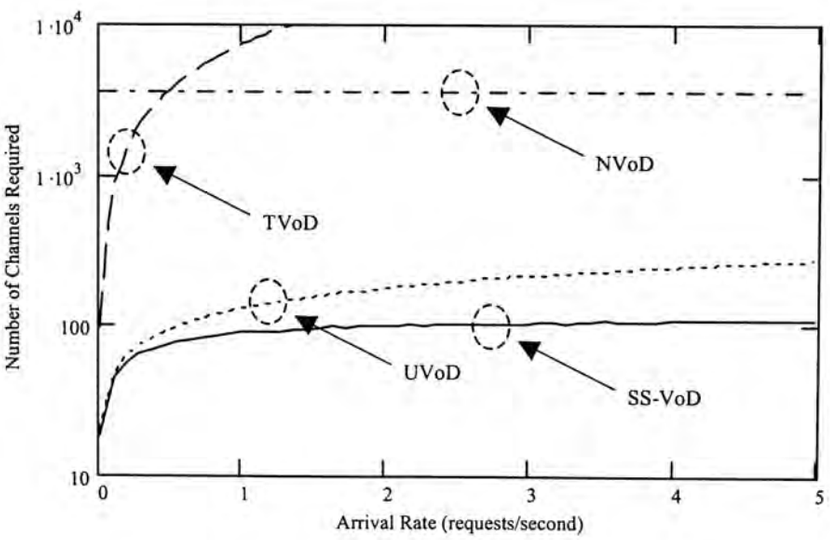

Figure 5.5: Channel requirement of SS-VoD and typical VoD systems

As expected, number of channel required for NVoD is a constant value and equal to 3600. The channel requirement of TVoD rapidly increases with the arrival rate and the number of channels required is larger than that of NVoD for the arrival

rate of 0.485 request per second. The channel requirement of SS-VoD and UVoD is much lower than TVoD and NVoD in all the arrival rates. The channel requirement of UVoD increases gracefully with the arrival rate. For example, the channel requirement of UVoD is equal to 130 at the arrival rate of one request per second, and it increases to 274 at the arrival rate of five requests per second. The channel requirement of SS-VoD increases relatively slower compare to UVoD for arrival rate up to one request per second. For arrival rate higher than one request per second, the channel requirement of SS-VoD increases insignificantly. Specifically, the number of channel required is equal to 90 and 108 at the arrival rate of one and five requests per second respectively. Note that the channel requirement only increases 20% for the five times increase in the arrival rate.

It is worth noting that the channel requirement of SS-VoD is relatively constant for arrival rate from one request per second. Having this nearly constant channel requirement property, SS-VoD will never experience overflow, and the overall user latency will not be increased significantly for a sudden increase in the arrival rate.

5.5 Performance at Light Loads

The previous results are computed using relatively high arrival rates. Intuitively, the performance gains will decrease at lower arrival rates as fewer requests will be served by a dynamic multicast channel. To investigate this issue, we compute the number of channels required at a given arrival rate so that the latency is equal to or

shorter than one second. Figure 5.6 shows the channel reduction over TVoD in percentage versus the arrival rate from 1×10^{-4} to 0.01 for SS-VoD and UVoD. The channel reduction percentage in the y-axis, denoted by G, is calculated from

$$G = \frac{\min\{n \mid W_{TVoD} \le 1, \forall n = 0,1,\ldots\} - \min\{n \mid W \le 1, \forall n = 0,1,\ldots\}}{\min\{n \mid W_{TVoD} \le 1, \forall n = 0,1,\ldots\}} \times 100\% \quad (5.3)$$

where W is the average waiting time for SS-VoD/UVoD at the given arrival rate.

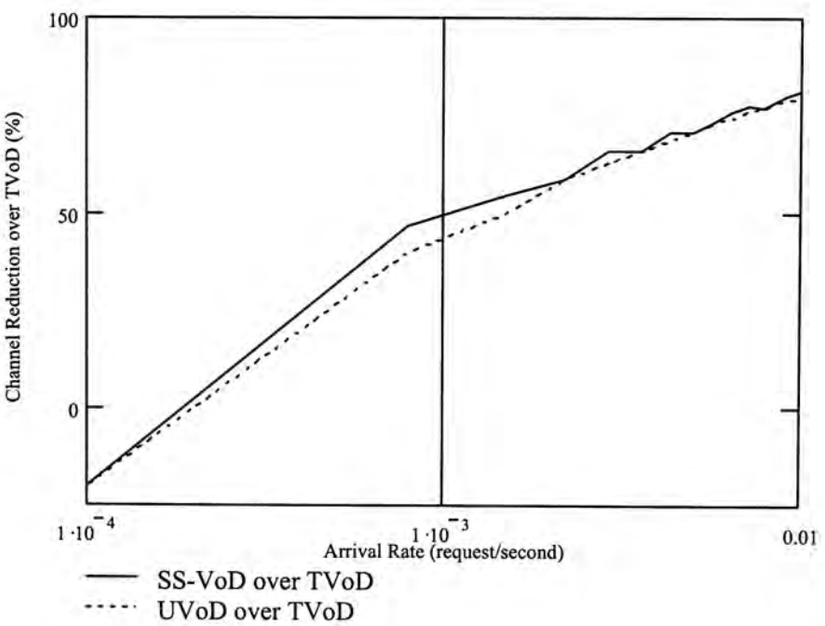

Figure 5.6: Channel Reduction over TVoD

As expected, the results show that SS-VoD requires fewer channels for arrival rates greater than 1.8×10^{-4} request per second. At this arrival rate, both TVoD and SS-VoD require only six channels. Note that the minimum number of channels required under SS-VoD is two and for arrival rates lower than 1×10^{-9} request per

second, TVoD will require only one channel. This suggests that SS-VoD is likely to outperform TVoD in practice.

5.6 Multiplexing Gain

In deriving the performance model in Chapter 4, we assumed there is one movie in the service node. To support more than one movie, one can treat each movie independently and assign channels according to the expected arrival rate and latency constraint. We call this partitioned SS-VoD in light of the fact that channels are partitioned (i.e. not shared) between different movies.

On the other hand, we can also pool the dynamic multicast channels together and share them among all movies in a first-come-first-serve manner. We call this multiplexed SS-VoD. Intuitively, partitioned SS-VoD is less efficient because a request for a movie can be blocked even if there are free dynamic multicast channels assigned to other movies. By contrast, multiplexed SS-VoD avoids this problem by pooling and sharing dynamic multicast channels and hence can achieve better performance.

To investigate the effect of this multiplexing gain, we conducted simulations for partitioned SS-VoD, and multiplexed SS-VoD with 2 movies, 8 movies and 32 movies respectively. For all cases, we assign 50 channels to each movie and assume all movies to be equally popular. Under these assumptions, the latency for partitioned SS-VoD is independent of the number of movies in the system.

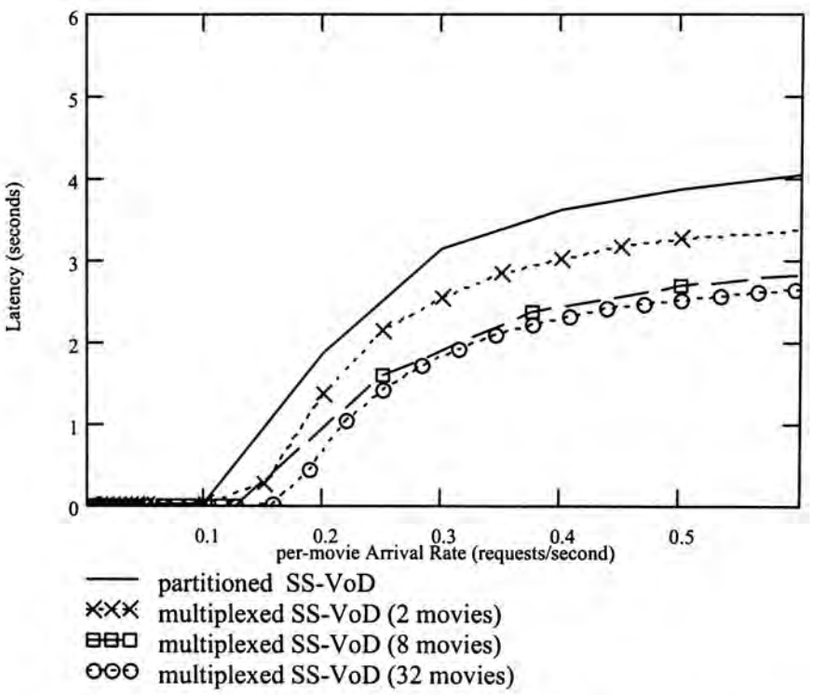

Figure 5.7: Multiplexing gain of SS-VoD system

Figure 5.7 shows the latency for partitioned SS-VoD and multiplexed SS-VoD for per-movie arrival rates up to 0.6 requests per second. As expected, the results show that multiplexed SS-VoD outperforms partitioned SS-VoD and the multiplexing gain increases with more movies. For example, at a per-movie arrival rate of 0.6 request per second, multiplexed SS-VoD with 2 movies outperforms partitioned SS-VoD by 15%. If we increase the number of movies to 32, multiplexed SS-VoD will outperform partitioned SS-VoD by as much as 32%. This suggests that the multiplexing gain is significant and hence it is worthwhile to adopt the multiplexed SS-VoD instead of partitioned SS-VoD in practice.

Chapter 6

Implementation and Benchmarking

In this chapter we present the implementation detail as well as the benchmarking results. The simulation and analytic result shows the designed architecture substantial increase the system capacity. To prove the feasibility of this system, we implement a SS-VoD prototype with up-to-date hardware and software. To further validate the analytic model, we run a set of benchmarking to compare with the analytic results.

6.1 Implementation Description

The SS-VoD prototype is implemented using off-the-shelf software and hardware. The hardware configuration is shown in Figure 6.1.

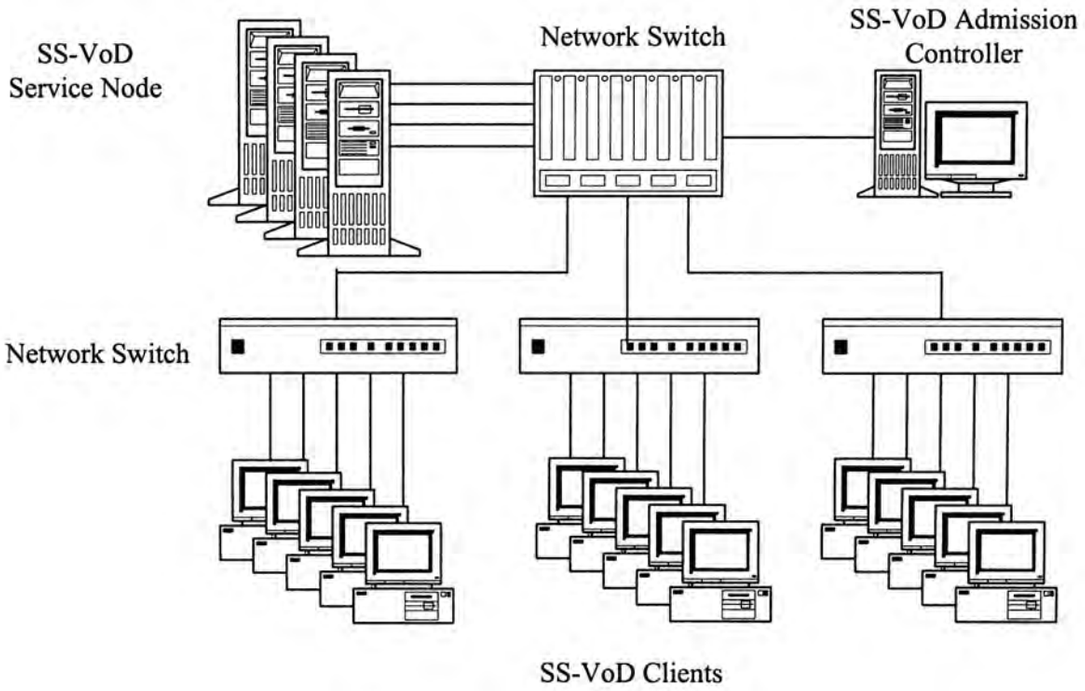

Figure 6.1: A SS-VoD Network.

There are three components in the prototype, which are SS-VoD service node, SS-VoD admission controller, and SS-VoD clients. Both the SS-VoD service node and the SS-VoD admission controller are implemented using the C++ programming language on the Red Hat Linux 6.2 [21] operating system platform. The SS-VoD client application is implemented using the Java programming language and the Java Media Framework (JMF) 2.1 [22].

The operation of the system is as follows. First, digitized and compressed video data for each movie is stored in a service node, and each service node is running the SS-VoD service node software. Each SS-VoD service node connects to the SS-VoD admission controller for registering the movie. Client stations running the SS-VoD client software can then send request to the SS-VoD admission controller. SS-VoD

admission controller will transmit the proper information (e.g. multicast address of the video channels, detail movie information, latency for available channels) to the client. The client then receives video data from the SS-VoD service node directly and starts the movie playback.

6.2 Benchmarking

We have discussed the system performance of SS-VoD in the previous chapters. To provide a more realistic figure for this architecture, benchmarking is a necessary and importance procedure. As there are limited hardware resources, we setup up a test-bed for a signal movie system. The benchmarking experiment consists of three major components: SS-VoD service node, admission controller, and client generator. The role of service node and admission controller remains unchanged. We have developed a java application called client generator, and the purpose of the client generator is to generate the desirable client request rate for the benchmarking. The detail benchmarking setup and result are presented in the following sub-sections.

6.2.1 Benchmarking Setup

The hardware configurations for different system components in the test-bed are listed in Tables 6.1 to 6.4 in the following:

Component	Model and Configuration
Motherboard	Compaq server-grade mainboard (2 PCI slots, 4GMB DRAM)
CPU	Intel Pentium III 800Mhz
Disk Controller	Compaq Ultra3 SCSI
Disk	2 x Fujitsu MAJ3182MC (Ultra3 SCSI)
Network	Intel PRO/1000T (1000Mbps)

Table 6.1: Service Node Configuration (Compaq Proliant DL360)

Component	Model and Configuration
Motherboard	Compaq server-grade mainboard (6 PCI slots, 256MB DRAM)
CPU	Intel Pentium III 500Mhz
Disk Controller	Compaq Ultra2 SCSI
Disk	3 x Fujitsu MAG3182LC (Ultra2 SCSI)
Network	Intel PRO/1000T (1000Mbps)

Table 6.2: Admission Controller Configuration (Compaq Proliant 1600)

Component	Model and Configuration
CPU	Intel Pentium III 500Mhz
Memory	256M SDRAM
Network	Intel PRO/1000T (1000Mbps)

Table 6.3: Client Generator Configuration

Component	Model and Configuration
Switch	Extreme Networks Summit24

Table 6.4: Interconnection Network Configuration

In conducting the benchmarking tests, the service node only serves one movie and the length of movie is 120 minutes. We run the benchmarking tests for 30

channels for arrival rate from one to five requests per second. For each configuration, we run the benchmarking tests for a length of 6 hours with the first hour of data skipped to reduce initial condition effects. In the implementation, the latency is known for users when they join to the system. Therefore, the latency is captured in the client generator.

6.2.2 Benchmarking Result

Table 6.5 compares the latencies obtained from the analytic performance model, simulation, and benchmarking respectively. We observe that the benchmarking results agree with the analytical results and simulation results. The maximum difference between benchmarking and analytic result is less than 4.5% in this range of load. Therefore, the benchmarking results serves as a proof for the feasibility and correctness of the SS-VoD architecture, and verifies the performance model derived in Chapter 4.

Request per second	Analytic Result	Simulation	Benchmarking
1	13.90s	12.95s	13.86s
2	14.39s	13.34s	13.78s
3	14.52s	13.59s	14.20s
4	14.57s	13.61s	14.70s
5	14.67s	13.68s	14.58s

Table 6.5: Latency comparison of analytic, simulation and benchmarking results

Chapter 7

Conclusion

In this study, we present and analyze a Super-Scalar Video-on-Demand (SS-VoD) architecture that can achieve super-linear scalability by utilizing network multicast together with client-side caching. This SS-VoD architecture is particularly suitable for metropolitan-scale deployment as the resource savings increase exponentially with higher arrival rates. In fact, there is no inherent scalability limit to this SS-VoD architecture provided that the network is multicast-ready, and has sufficient bandwidth to connect all customers. With more and more existing residential broadband networks being upgraded to support multicast, the presented SS-VoD architecture could provide a cost-effective solution to the scalability challenge.

Appendix

To compute the average waiting time for Type-2 request, denoted by $W_2(\delta)$, we first compute the apparent waiting time distribution of the dynamic multicast channels for Type-2 user request, denoted by $f_{C^*}(t)$. We then can compute the average waiting time of Type-2 user request.

The apparent waiting time distribution of dynamic multicast channel for Type-2 user is given by [23]:

$$f_{C^*}(t) = \frac{tf_C(t)}{M_C} \tag{A.1}$$

where $f_{C^*}(t)$ be the apparent waiting time distribution of dynamic multicast channels for Type-2 user, $f_C(t)$ be the waiting time distribution of dynamic multicast channels and M_C be the mean waiting time of dynamic multicast channel.

Let $W_{C^*}(\delta)$ be the average waiting time of the apparent distribution, and it can be calculated as following:

$$W_{C^*}(\delta) = \int_{-\infty}^{\infty} tf_{C^*}(t)dt \tag{A.2}$$

Therefore from equation (A.1) and (A.2),

$$W_{C^*}(\delta) = \int_{-\infty}^{\infty} \frac{t^2 f_C(t)}{M_C} dt \tag{A.3}$$

From the definition, the minimum waiting time should be a non-negative value, and the maximum waiting time of user in dynamic channel is equal to (T_R–2δ). We can simplify the above equation to:

$$W_{C^*}(\delta) = \int_{0}^{T_R-2\delta} \frac{t^2 f_C(t)}{M_C} dt \tag{A.4}$$

We assume $f_C(t)$ is truncated exponential distributed with mean equal to $W_C(\delta)$. The range of t is from zero to (T_R–2δ). The distribution is given by:

$$f_C(t) = \left((1 - e^{\frac{-(T_R-2\delta)}{W_C(\delta)}}) W_C(\delta) \right)^{-1} e^{\frac{-t}{W_C(\delta)}} \tag{A.5}$$

From equation (A.4) and (A.5),

$$W_{C^*}(\delta) = \int_{0}^{T_R-2\delta} \frac{t^2 e^{\frac{-t}{W_C(\delta)}}}{(1 - e^{\frac{-(T_R-2\delta)}{W_C(\delta)}}) W_C(\delta)^2} dt \tag{A.6}$$

Here, we then computer the average apparent waiting time of dynamic multicast channels by integration,

$$W_{C^*}(\delta) = -\frac{1}{(1 - e^{\frac{-(T_R-2\delta)}{W_C(\delta)}}) W_C(\delta)} t^2 e^{\frac{-t}{W_C(\delta)}} \Bigg|_0^{T_R-2\delta} + \int_{0}^{T_R-2\delta} \frac{2te^{\frac{-t}{W_C(\delta)}}}{(1 - e^{\frac{-(T_R-2\delta)}{W_C(\delta)}}) W_C(\delta)} dt \tag{A.7}$$

First, we calculate the left part, it becomes

$$W_{C^*}(\delta) = -\frac{(T_R - 2\delta)^2 e^{\frac{-(T_R-2\delta)}{W_C(\delta)}}}{(1-e^{\frac{-(T_R-2\delta)}{W_C(\delta)}})W_C(\delta)} + \int_0^{T_R-2\delta} \frac{2te^{\frac{-t}{W_C(\delta)}}}{(1-e^{\frac{-(T_R-2\delta)}{W_C(\delta)}})W_C(\delta)} dt$$

(A.8)

We simply the equation by *integration by parts* for the right part,

$$W_{C^*}(\delta) = -\frac{(T_R - 2\delta)^2 e^{\frac{-(T_R-2\delta)}{W_C(\delta)}}}{(1-e^{\frac{-(T_R-2\delta)}{W_C(\delta)}})W_C(\delta)} - \frac{2te^{\frac{-t}{W_C(\delta)}}}{(1-e^{\frac{-(T_R-2\delta)}{W_C(\delta)}})}\Bigg|_0^{T_R-2\delta} + \int_0^{T_R-2\delta} \frac{2e^{\frac{-t}{W_C(\delta)}}}{(1-e^{\frac{-(T_R-2\delta)}{W_C(\delta)}})} dt$$

(A.9)

After calculating the second term, it becomes

$$W_{C^*}(\delta) = -\frac{(T_R - 2\delta)^2 e^{\frac{-(T_R-2\delta)}{W_C(\delta)}}}{(1-e^{\frac{-(T_R-2\delta)}{W_C(\delta)}})W_C(\delta)} - \frac{2(T_R - 2\delta)e^{\frac{-(T_R-2\delta)}{W_C(\delta)}}}{(1-e^{\frac{-(T_R-2\delta)}{W_C(\delta)}})} + \int_0^{T_R-2\delta} \frac{2e^{\frac{-t}{W_C(\delta)}}}{(1-e^{\frac{-(T_R-2\delta)}{W_C(\delta)}})} dt$$

(A.10)

Grouping the first and second parts of (A.10), then,

$$W_{C^*}(\delta) = -\frac{(T_R - 2\delta)e^{\frac{-(T_R-2\delta)}{W_C(\delta)}}}{(1-e^{\frac{-(T_R-2\delta)}{W_C(\delta)}})}\left(\frac{(T_R - 2\delta)}{W_C(\delta)} + 2\right) + \int_0^{T_R-2\delta} \frac{2e^{\frac{-t}{W_C(\delta)}}}{(1-e^{\frac{-(T_R-2\delta)}{W_C(\delta)}})} dt$$

(A.11)

After integrate the right-most part and simplify the equation, it gives

$$W_{C^*}(\delta) = \frac{2W_C(\delta)}{(1-e^{\frac{-(T_R-2\delta)}{W_C(\delta)}})}\left(1-e^{\frac{-(T_R-2\delta)}{W_C(\delta)}} - \left((1+\frac{(T_R-2\delta)}{2W_C(\delta)})\right)\frac{(T_R-2\delta)e^{\frac{-(T_R-2\delta)}{W_C(\delta)}}}{W_C(\delta)}\right)$$

$$= 2W_C(\delta)\left(1-\frac{1+\frac{(T_R-2\delta)}{2W_C(\delta)}}{1-e^{\frac{-(T_R-2\delta)}{W_C(\delta)}}}\frac{(T_R-2\delta)}{W_C(\delta)}e^{\frac{-(T_R-2\delta)}{W_C(\delta)}}\right) \quad \text{(A.12)}$$

For Poisson process, the user randomly arrives to the dynamic multicast channel. Hence, the average waiting time of Type-2 user request should be equal to the half of the average apparent waiting time of dynamic multicast channel by Type-2 user requests. Therefore,

$$W_2(\delta) = \frac{W_{C^*}(\delta)}{2}$$

$$= W_C(\delta)\left(1-\frac{1+\frac{(T_R-2\delta)}{2W_C(\delta)}}{1-e^{\frac{-(T_R-2\delta)}{W_C(\delta)}}}\frac{(T_R-2\delta)}{W_C(\delta)}e^{\frac{-(T_R-2\delta)}{W_C(\delta)}}\right) \quad \text{(A.13)}$$

■

Bibliography

[1] T.C. Chiueh and C.H. Lu, "A Periodic Broadcasting Approach to Video-on-demand Service," *Proc. of SPIE*, 1996, pp.162-169.

[2] A. Dan, D. Sitaram, and P. Shahabuddin, "Scheduling Policies for an On-Demand Video Server with Batching," *Proc. 2nd ACM International Conference on Multimedia*, 1994, pp.15-23.

[3] H. Shachnai and P.S. Yu, "Exploring Waiting Tolerance in Effective Batching for Video-on-Demand Scheduling," *Proc. 8th Israeli Conference on Computer Systems and Software Engineering*, Jun 1997, pp.67-76.

[4] W. Liao and V.O.K. Li, "The Split and Merge protocol for interactive video-on-demand," *IEEE Multimedia*, vol.4(4), 1997, pp.51-62.

[5] S.W. Carter, D.D.E. Long, K. Makki, L.M. Ni, M. Singhal, and N. Pissinou, "Improving Video-on-Demand Server Efficiency Through Stream Tapping," *Proc. 6th International Conference on Computer Communications and Networks*, Sep 1997, pp.200-207.

[6] S. Viswanathan and T. Imielinski, "Metropolitan area video-on-demand service using pyramid broadcasting," *ACM Multimedia Systems*, vol.4(4), 1996, pp.197-208.

[7] C.C. Aggarwal, J.L. Wolf, and P.S. Yu, "A permutation-based pyramid broadcasting scheme for video-on-demand systems," *Proc. International Conference on Multimedia Computing and Systems*, June 1996, pp.118-26.

[8] L. Golubchik, J.C.S. Lui, and R.R. Muntz, "Adaptive piggybacking: a novel technique for data sharing in video-on-demand storage servers," *ACM Multimedia Systems*, vol.4(30), 1996, pp.14-55.

[9] C.C. Aggarwal, J.L. Wolf, and P.S. Yu, "On optimal piggyback merging policies for video-on-demand systems," *Proc. International Conference on Multimedia Systems*, June 1996, pp.253-258.

[10] Kien A. Hua, Ying Cai and Simon Sheu, "Patching: a multicast technique for true video-on-demand services," *Proc. 6th international conference on Multimedia*, Sept 1998, pp 191 – 200.

[11] Ramesh, S., Rhee, I., Guo, K, "Multicast with cache (Mcache): an adaptive zero-delay video-on-demand service," *IEEE Transactions on Circuits and Systems for Video Technology*, vol.11(3), March 2001, pp. 440 -456

[12] K.C. Almeroth and M.H. Ammar, "The Use of Multicast Delivery to Provide a Scalable and Interactive Video-on-Demand Service," *IEEE Journal of Selected Areas in Communications*, vol.14(6), Aug 1996, pp.1110-1122.

[13] H.K. Park, and H.B. Ryou, "Multicast Delivery for Interactive Video-on-Demand Service," *Proc. 12th International Conference on Information Networking*, Jan 1998, pp.46-50.

[14] E.L. Abram-Profeta and K.G. Shin, "Providing Unrestricted VCR Functions in Multicast Video-on-Demand Servers," *Proc. IEEE International Conference on Multimedia Computing and Systems*, July 1998, pp.66-75.

[15] J.Y.B. Lee, "UVoD – A Unified Architecture for Video-on-Demand Services," *IEEE Communications Letters*, vol.3(9), September 1999, pp.277-279.

[16] P.M. Chen, E.K. Lee, G.A. Gibson, R.H. Katz, and D.A. Patteson, "Raid: High-performance, reliable secondary storage," *ACM Computing Surveys*, vol. 26, pp. 145-185, June 1994.

[17] S. Berson, L. Golubchik, and R. R. Muntz, "Fault tolerant design of multimedia servers," *Proceedings of SIGMOD'95*, pp. 364-375, May 1995.

[18] R. Tewari, D.M. Dias, W. Kish, and H.Vin, "Design and performance tradeoffs in clustered video servers," *Proceedings IEEE International Conference on Multimedia Computing and Systems (ICMCS'96)*, pp. 144-150, June 1996.

[19] A.O. Allen, *Probability, Statistics, and Queueing Theory with Computer Science Applications*, 2nd Ed. Academic Press, New York, 1990.

[20] ComNets Class Library and Tools:
http://www.comnets.rwth-aachen.de/doc/cncl.html

[21] Red Hat Linux: http://www.redhat.com

[22] Java Media Framework 2.1:
http://java.sun.com/products/java-media/jmf/index.html

[23] L. Kleinrock, *Queueing Systems Vol I: Theory*, Wiley-Interscience, 1975.