Turbo-Slice-and-Patch: An Algorithm for Metropolitan Scale VBR Video Streaming

KONG CHUN WAI

A Thesis Submitted in Partial Fulfillment
of the Requirements for the Degree of
Master of Philosophy
In
Information Engineering

July 2003

ACKNOWLEDGEMENT

First of all, I would like to thank my supervisor, Prof. Jack Y. B. Lee, for his invaluable advices, guidance and patience. During my research study, he fully supported my work and provided guidance on my research direction.

I would also like to thank my friends Raymond, Rudolf, Wai, Patton and Nera for their kindness in discussing with me problems and solutions during the last two years.

Finally, I would like to express my gratitude towards my family for their support.

ABSTRACT

In recent years, a number of sophisticated architectures have been proposed to provide video-on-demand (VoD) service using multicast transmissions. Compared to their unicast counterparts, these multicast VoD systems are highly scalable and can potentially serve millions of concurrent users. Nevertheless, these systems are designed for streaming constant-bit-rate (CBR) encoded videos and thus cannot benefit from the improved visual quality obtainable from variable-bit-rate (VBR) encoding techniques. To tackle this challenge, this thesis presents a Turbo-Slice-and-Patch (TSP) algorithm to support VBR video streaming in a multicast VoD system. Results obtained from trace-driven simulation of 300 VBR videos show that serving VBR videos with the TSP algorithm increases the average latency by only 9% compared to the CBR case with the same average video bit-rate. Moreover, in 165 out of the 300 video titles, the TSP algorithm actually outperforms the CBR equivalent by shortening the latency by 0.04% to 99%. Given that we can achieve similar visual quality by encoding VBR video at half the average rate of CBR video, this TSP algorithm can potentially serve VBR videos with more consistent visual quality and with less resource compare to CBR-based video streaming systems.

摘要

近年有很多學者以多重播送技術(Multicast)爲基礎，設計出能提供視頻點播(VoD)服務的系統。與使用單點傳送技術(Unicast)比較，使用多重播送技術的視頻點播系統擁有十分高的可升級性，將能夠同時服務過百萬位使用者。但是，此類系統是針對以常數比特率(constant-bit-rate)壓縮而成的視頻而設計。故此，不能藉著使用動態比特率(variable-bit-rate)壓縮技術以提升影象質素。因此，本論文將提供一個支援動態比特率壓制視頻並能使用於多重播送技術的視頻點播系統的演算法，名爲快速切割整合法 (Turbo-Slice-and-Patch)。就用戶的平均等候時間而言，由使用 300 套動態比特率壓制視頻的模擬顯示，以快速切割整合法提供視頻點播服務比以相同平均比特率、但使用常數比特率壓制視頻的系統僅高出 9%。而且，於其中的 165 套視頻，以快速切割整合法比使用常數比特率的系統能縮短等候時間 0.04% 至 99% 不等。由於使用動態比特率壓制的視頻只需要使用常數比特率壓制的平均比特率的一半，便能達到同樣的影象質素，快速切割整合法將能夠以比爲常數比特率設計的系統更少的資源提供穩定的影象質素。

Contents

Chapter 1

INTRODUCTION

In a true-video-on-demand (TVoD) system, the video server has to reserve a dedicated video channel for each user for the entire duration of the video session (e.g. two hours for a movie). Consequently, the server and network resources required increase linearly with the number of concurrent users to be supported. Although current PC servers are already very powerful and capable to serve up to hundreds of concurrent video streams, scaling up a system to thousands and even millions of concurrent video streams is still prohibitively expensive.

One promising solution to this scalability challenge is through the intelligent use of network multicast. Network multicast enables a server to send a few streams of video data for reception by a large number of clients, thereby significantly reducing the amount of resources required. A number of pioneering studies have investigated such architectures, such as batching [2-4], patching [5-8], and periodic broadcasting [9-12].

A common assumption among these multicast VoD architectures is that the videos are constant-bit-rate (CBR) encoded. This significantly simplifies system design and analysis, and enables one to study the system performance independent of video encoding variations. Nevertheless, the visual quality of CBR video is not

constant and tends to vary according to the video content. For example, complex video scenes with a lot of motions will typically result in lower visual quality than simple video scenes with little movement.

By contrast, videos encoded with constant-quality encoding algorithms will have more consistent visual quality, albeit at the expense of bit-rate variations. However, a study by Tan *et al.* [14] has shown that VBR-encoded video can achieve visual quality similar to CBR-encoded video using only half the bit-rate. This result suggests that VBR encoding is not only desirable for providing high-quality VoD services, but also has the potential to reduce resource requirements as well. The challenge is the complex resource allocation and scheduling problems resulting from the video bit-rate variations.

This study addresses this challenge by investigating a new turbo-slice-and-patch (TSP) algorithm for serving VBR-encoded video streams in a metropolitan-scale video streaming service using network multicast. Unlike previous work on VBR video streaming focused on unicast network transmission, the TSP algorithm investigated in this study employs network multicast to significantly increase the system's scalability to cope with the immense workload in a metropolitan scale streaming service. Results obtained from trace-driven simulation of 300 VBR videos show that serving VBR videos with the TSP algorithm increases the average latency by only 9% compared to the CBR case with the same average video bit-rate. Moreover, in 165 out of the 300 video titles, the TSP algorithm actually outperforms the CBR equivalent by shortening the latency by 0.04% to 99%. Given that we can achieve similar visual quality by encoding VBR video at half the average rate of CBR video, this TSP algorithm can potentially serve VBR videos with more

consistent visual quality and with less resources compare to CBR-based video streaming systems.

The rest of the thesis is organized as follows. Chapter 2 reviews some related work and compares them with this study; Chapter 3 reviews the multicast VoD architecture; Chapter 4 presents two priority scheduling algorithms; Chapter 5 presents the Turbo-Slice-and-Patch algorithm; Chapter 6 presents the proof of smooth playback guarantee. Chapter 7 evaluates and compares the three algorithms using simulation results; and Chapter 8 concludes the thesis.

Chapter 2

RELATED WORKS

The problem of VBR video delivery in unicast VoD systems has been studied extensively. We review some of the more relevant previous work in Section 2.1 and compare them with this study in Section 2.2.

2.1 Previous Work

One of the best known solutions for VBR video delivery is temporal smoothing [15-18]. Smoothing makes use of a client-side buffer to receive data in advance of playback. This work-ahead technique enables the server to transmit video data in a piecewise linear schedule that can be optimized to minimize rate variability [16] or to minimize the number of rate changes [17]. The schedule can be computed offline and with proper resource reservation, deterministic performance can be guaranteed. Interested readers are referred to Feng *et al.* [18] for a thorough comparison of various smoothing algorithms.

In another study by Lee and Yeom [19], a data prefetch technique is proposed to improve video server performance in serving VBR videos. Unlike smoothing, where all video data are retrieved from the disk sequentially, data prefetching preloads video data corresponding to a video's bit-rate peaks into the server's memory during system initialization. During operation, the server then only needs to

retrieve the remaining video data from the disk to combine with the prefetched data for transmission to the clients. As the remaining video stream has a lower peak bit-rate, disk utilization is increased. Their simulation results show that up to 81% more streams can be served using this prefetch technique. The tradeoffs are increased server buffer requirement and additional offline preprocessing of the video data.

A third approach proposed, by Saparilla *et al.* [9], schedules video data transmission using a priority scheduler (Join-the-Shortest Queue). In particular, the server schedules video data transmission according to the demand of data of each channel. A channel with the greatest demand of data (the clients listening to this channel are most likely to run out of data) will have the highest priority in the next round of transmission. However, while server efficiency is improved, this priority scheduler does not guarantee a client can receive all data in time. In particular, a channel will simply be skipped (i.e. not transmitted) if the data cannot be transmitted in time for playback. Their simulation results show that with their Join-the-Shortest Queue priority scheduling and allowing the client to retrieve data from seven channels synchronously, the start-up latency can be limited to around 100 seconds with a loss probability of 10^{-6}.

2.2 Comparison

Compared to the TSP algorithm investigated in this study, both temporal smoothing and the data prefetch techniques discussed previously are orthogonal and complementary. For temporal smoothing, a smoothed VBR video stream can be considered as just another VBR video stream, albeit one requiring additional client buffer for proper playback. For the data prefetch technique, the focus is on improving disk retrieval efficiency by intelligently preloading some video data into

the server memory. Obviously, this technique does not affect the transmission schedule at all and can thus be integrated with any transmission scheduling algorithms including S&P.

Compared to the study by Saparilla *et al.* [9], TSP differs in two major ways. First, the TSP algorithm guarantees that no video data will be skipped, thus ensuring visual quality. Second, TSP is targeted at clients with limited access bandwidth (twice the average bit-rate of the video). By contrast, the algorithm proposed by Saparilla *et al.* assumes the client have sufficient bandwidth to receive data from many channels simultaneously, which may not be practical for some applications.

In a previous work [1], we investigat an early version of the TSP algorithm, called slice-and-patch (S&P), which shares some of the design principles of the TSP algorithm. There are, however, two important differences. First, in S&P the VBR video is first smoothed using temporal smoothing (e.g. optimal smoothing [16]) before being subjected to the slicing operation. In TSP we divide the video into two sections that are independently smoothed. Second, TSP has a different algorithm in Phase 2 of the patching process (c.f. Section 5.3) where video data are transmitted at the maximum client access bandwidth rather than the original video bit-rate in S&P (hence the name turbo-slice-and-patch versus slice-and-patch [1]). Last but not least, we have conducted more extensive trace-driven simulation using 300 VBR video traces in this study (versus 50 in the previous study [1]) to evaluate the proposed algorithms in a much broader context to obtain more complete and reliable results.

Chapter 3

SYSTEM ARCHITECTURE

In this section, we briefly review the Super-Scalar VoD (SSVoD) architecture proposed by Lee and Lee [13]. SSVoD is designed for streaming CBR videos over a combination of static and dynamically-scheduled multicast transmission channels. Its primary advantages are the super-linear scalability achieved by multicast transmission, and the ability to support interactive playback control such as pause-resume and slow-motion playback without additional server resources. Interested readers are referred to the study by Lee and Lee [13] for more details. We discuss in Section 3.3 the challenges in streaming VBR video over the SSVoD architecture.

3.1 Transmission Scheduling

The SSVoD architecture comprises a number of service nodes delivering video data over multicast channels to the clients. SSVoD achieves scalability and bandwidth efficiency by sending video data to a large number of clients using a few multicast channels. However, simple periodic multicast schemes such as those used in a near-video-on-demand (NVoD) system limit 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, SSVoD employs patching to enable a client to start video

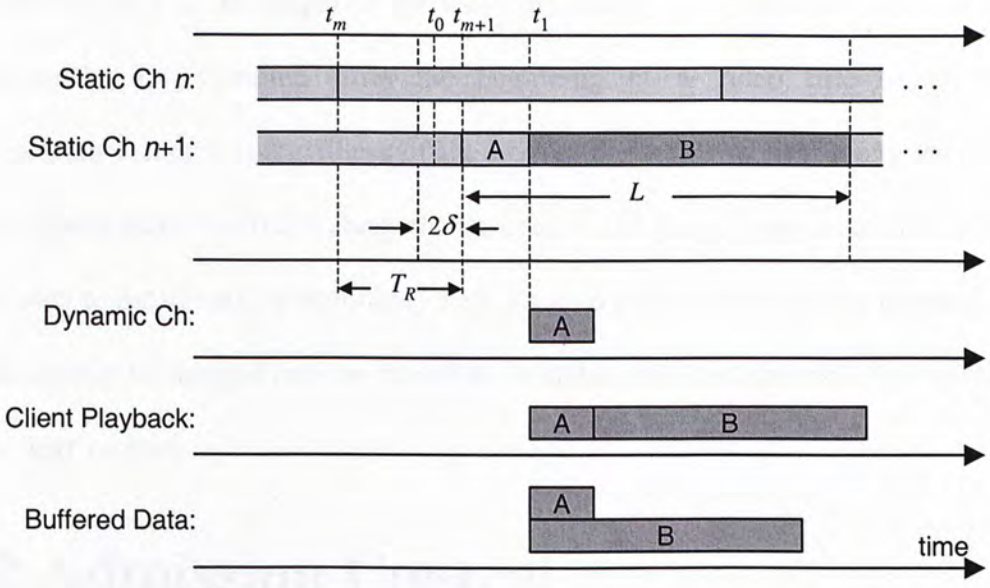

Figure 3.1. The patching process in the super-scalar video-on-demand system supporting CBR video.

playback at any time using a dynamic multicast channel until it can be merged back onto an existing multicast channel. The following sections present these techniques in more detail.

Each service node in the system streams video data using 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 N_S static multicast channels and $N_D=N_M-N_S$ dynamic multicast channels. The video title is multicast repeatedly over all N_S static multicast channels in a time-staggered manner as shown in Figure 3.1. Specifically, adjacent channels are offset by

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

seconds, where L is the length of the video in seconds. Transmissions are repeated continuously, i.e. restarted from the beginning of a video title every time transmission completes, regardless of the load of the server or how many users are active. These static multicast channels are used as the main channels 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 as explained in the next section.

3.2 Admission Control

To reduce the response time while still leveraging the bandwidth efficiency of multicast, SSVoD allocates a portion of the multicast channels and schedules them dynamically according to the request arrival pattern. A new user either waits for the next upcoming multicast transmission from a static multicast channel, or starts playback with a dynamic multicast channel.

Suppose a new request arrives at time t_0, 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} (see Figure 3.1). 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 a predefined admission threshold 2δ, i.e., $w_i = t_{m+1} - t_0 \leq 2\delta$. We call these requests *statically admitted.* This admission threshold is introduced to reduce the amount of load going to the dynamic multicast channels.

On the other hand, if the waiting time is longer than the threshold, then the client will request a dynamic multicast channel to begin playback (*dynamically admitted*), while at the same time caches video data from the multicast channel with the multicast cycle started at time t_m. Note that the client may need to queue up and

wait for a dynamic multicast channel to become available. If additional clients requesting the same video arrive during the wait, they will be batched and served by the same dynamic multicast channel once it becomes available. Eventually, the client playback will reach the point where the cached data began and the client can then release the dynamic multicast channel and continue playback using data received from the static multicast channel. This integration of batching with patching significantly increases the system's efficiency at heavy loads.

Compared to TVoD systems, an SSVoD client must have the capability to receive two multicast channels concurrently and have a local buffer to hold up to T_R seconds of video data. Given a video bit-rate of 3Mbps (e.g. high-quality MPEG-4 video), a total of 6Mbps downstream bandwidth is required during the initial patching phase of the video session. For a two-hour movie served using 25 static multicast channels, the buffer requirement is 108MB. This can easily be accommodated using a small harddisk at the client, and in the near future simply using memory as technology improves.

3.3 Challenges in Supporting VBR-encoded Video

The SSVoD architecture is designed for CBR videos and thus problems will arise if we want to stream VBR videos using the architecture. The first problem is in channel allocation. SSVoD partitions the server and network bandwidth into fixed-bandwidth network channels for allocation purpose. This allocation model is clearly undesirable for streaming VBR videos as it requires each channel to have sufficient bandwidth to accommodate the peak rate of the video, which is typically many times the average video bit-rate.

To tackle this problem, we need to abandon the fixed-rate channel allocation model altogether and resort to allocating bandwidth according to the exact video bit-rate profile. Specifically, instead of reserving half the channels for static multicast channels, we reserve half the server and network bandwidth for multicasting VBR streams in a time-staggered manner. Let $v(t)$ be the video playback bit-rate function defining the bit-rate at which video data are being consumed t seconds after playback has begun. To multicast a video title in n independent time-staggered streams, the aggregate video bit-rate of the ensemble, denoted by $V_S(t,n)$ will be given by

$$V_S(t,n) = \sum_{i=0}^{n-1} v\left(\left(t \mod \frac{L}{n}\right) + \frac{iL}{n}\right) \tag{3.2}$$

Thus we can determine the maximum number of time-staggered VBR video streams that can fit within the system capacity from

$$N_S = \max\left\{n \,|\, V_S(t,n) \le 0.5C, \forall t, n\right\} \tag{3.3}$$

where C is the total server and network bandwidth available.

The second problem is in the client access network where the client has an access network bandwidth equal to twice the video bit-rate. While this is sufficient for receiving two CBR video streams, it is likely to run into congestion when VBR video is streamed due to the inherent video bit-rate variations. The use of temporal smoothing can alleviate this problem but cannot solve it completely without adding excessive start-up delay. In the next section, we address this problem by presenting two streaming algorithms based on priority scheduling.

Chapter 4

PRIORITY SCHEDULING

The primary problem in streaming VBR video in SSVoD is that dynamically-admitted clients may not have sufficient access bandwidth to accommodate both the dynamic and the static multicast channel. For example, let R_V be the average video bit-rate, then the client has an access bandwidth of $2R_V$. However, a VBR video of average bit-rate R_V will likely have bit-rate peaks substantially higher than R_V even after smoothing is applied (e.g. some of the videos in our collection have average bit-rate 4Mbps but have peak bit-rate exceeding 12Mbps). It is easy to see that the access channel will become seriously congested whenever peaks from both dynamic channel and static channel overlap.

Obviously we can increase the access network bandwidth to accommodate the overlapping bit-rate peaks. However this trivial solution suffers from two limitations. First, the access network bandwidth is often limited by the access network technology employed. For example, if Ethernet is employed as the access network infrastructure, then the access bandwidth can never exceed 10Mbps (lower in practice due to frame/packet header overheads). Thus in this case the access network cannot even accommodate one single stream of the VBR video (which has peaks over 10Mbps), let alone two streams. Second, the precise access bandwidth

required is dependent on the particular video being streamed, thus rendering it impossible to fix the access network bandwidth during system design.

Therefore we present in the following two priority-scheduling algorithms that can operate with a given access network bandwidth (say two times the average video bit-rate) and yet are able to support the caching and patching operations in SSVoD.

4.1 Static Channel Priority (SCP)

In the static channel priority algorithm, we schedule the static channels to transmit at the original video bit-rate. The dynamic channel will simply use the remaining access network bandwidth to transmit video data for patching so that the aggregate bit-rate does not exceed the access bandwidth limit. Before streaming can start, we will process the video offline by collecting all the data above the bit-rate R_{max} for $0 \leq \tau \leq L$ to form the prefetch block P, which is of bit-rate $v_p(\tau)$ given by:

$$v_p(\tau) = \begin{cases} v_0(\tau) - R_{max} & \text{if } v_0(\tau) > R_{max} \\ 0 & \text{otherwise} \end{cases} \tag{4.1}$$

where $v_0(\tau)$ is the original video bit-rate for any playback points $0 \leq \tau \leq L$. We can thus guarantee that the client's access bandwidth will be sufficient for caching data from the static channel after the dynamic channel is released. This block P will be multicast periodically by a static channel at the bit-rate R_{max}. A client arriving at the system will first prefetch data from this channel for a period of $\frac{1}{R_{max}} \int_0^L v_p(t)dt$ seconds.

Assume the client finishes prefetching at time t_0 and the immediate previous multicast cycle begins at time t_m. The client will immediately begin caching video data from the static multicast channel starting from a playback point of $t_0 - t_m$ and

request a dynamic channel to stream the missed video data from playback point 0 to t_0-t_m. Then the amount of residual access bandwidth left for the dynamic channel at time t is equal to $u(t) = R_{max} - v(t-t_m)$ for $t \geq t_1$ where $v(\tau)=v_0(\tau)-v_p(\tau)$ for all $0 \leq \tau \leq L$.

As the client has already missed the first t_0-t_m seconds of the video, a dynamic channel will be allocated to stream video data from the beginning of the video to the playback point t_0-t_m. The transmission duration, denoted by d_m, can then be obtained by solving the following equation:

$$\int_{t_1}^{t_1+d_m} u(\tau)d\tau = \int_{t_1}^{t_1+(t_0-t_m)} v(\tau-t_1)d\tau \tag{4.2}$$

However, since the residual bandwidth available to the dynamic channel may not be sufficient to sustain continuous playback, the client may need to introduce an extra delay before playback can begin. Specifically, if the following inequality is satisfied:

$$\int_{t_1}^{t} u(\tau)d\tau \geq \int_{t_1}^{t} v(\tau-t_1)d\tau, \text{ for } t_1 \leq t \leq (t_1+d_m) \tag{4.3}$$

Then it implies that the amount of video data received from the dynamic channel always exceeds the amount required for continuous playback. In this case the client can begin playback as soon as the dynamic channel becomes available. Otherwise, the client will have to delay playback by say d_s seconds so that the continuity condition is satisfied:

$$d_s = \min\left\{ d \middle| \int_{t_1}^{t} u(\tau)d\tau \geq \int_{t_1+d}^{t} v(\tau-t_1-d)d\tau, \text{ for } (t_1+d) \leq t \leq (t_1+d_m) \right\} \tag{4.4}$$

As we will show in Section 7.1, d_s can become very large for certain videos.

Next we derive the client buffer requirement for SCP. Specifically, the client will need to buffer video data from both the dynamic channel and the static channel. First, we derive the amount of data received at any time t. From the time the client

has finished prefetching at time t_0, data will be cached from the static channel up to the end of the video section at time (t_m+L). The amount of data received from the static channel at any time t where $t_0 \le t \le (t_m + L)$ is equal to $\int_{t_0}^{t} v(\tau - t_m)d\tau$. To simplify notations, we set $v(\tau)=0$ for all $\tau>L$ and $\tau<0$. From time t_1 to time t_1+d_m, the client will receive video data from the dynamic channel at the rate $u(t)$. Thus the accumulated amount of video data received by time t is equal to $\int_{t_1}^{t} u(\tau)d\tau$. Note that $u(t)=0$ for all $t < t_1$ (i.e., before dynamic channel becomes available) and $t > (t_1 + d_m)$ (i.e., patching is completed).

Therefore the total amount of data received at any time t is simply given by

$$\int_0^L v_p(\tau)d\tau + \int_{t_0}^{t} v(\tau - t_m)d\tau + \int_{t_1}^{t} u(\tau)d\tau \tag{4.5}$$

Now as the client begins playback from time t_1+d_s, the accumulated amount of data consumed by the time t for $t_1 + d_s \le t \le (t_1 + d_s + L)$ is then given by

$$\begin{aligned} &\int_{t_1+d_s}^{t} \left[v\left(\tau - (t_1 + d_s)\right) + v_p\left(\tau - (t_1 + d_s)\right) \right] d\tau \\ &= \int_{t_1+d_s}^{t} v\left(\tau - (t_1 + d_s)\right) d\tau + \int_0^{t-(t_1+d_s)} v_p(\tau)\, d\tau \end{aligned} \tag{4.6}$$

Finally, we can compute the amount of excess data received but not yet played back at any time t from

$$\begin{aligned} U_{scp}(t_0, t_1, t) &= \left(\int_0^L v_p(\tau)d\tau + \int_{t_0}^{t} v(\tau - t_m)d\tau + \int_{t_1}^{t} u(\tau)d\tau \right) \\ &\quad - \left(\int_{t_1+d_s}^{t} v\left(\tau - (t_1 + d_s)\right) d\tau + \int_0^{t-(t_1+d_s)} v_p(\tau)\, d\tau \right) \\ &= \int_{t-(t_1+d_s)}^{L} v_p(\tau)d\tau + \int_{t_0}^{t} v(\tau - t_m)d\tau + \int_{t_1}^{t} u(\tau)d\tau - \int_{t_1+d_s}^{t} v\left(\tau - (t_1 + d_s)\right) d\tau \end{aligned} \tag{4.7}$$

The maximum of (4.7) thus determines the client buffer requirement

$$U_{scp}^{\max} = \max\{U_{scp}(t_0,t_1,t), \forall t_0,t_1,t \\ | t_m \le t_0 \le (t_m + T_R), t_0 \le t_1 \le (t_m + T_R),\ t_1 \le t \le (t_1 + d_s + L)\} \qquad (4.8)$$

4.2 Dynamic Channel Priority (DCP)

To avoid the startup delay in the previous static channel priority algorithm, we can alternatively give priority to the dynamic channel. Unlike the previous algorithm, the static channel cannot simply transmit video data using the leftover access bandwidth because the static channels are periodically multicast in a fixed schedule to a large number of clients. Therefore, once a dynamic channel becomes available at time $t_1=t_0+w$, the server will transmit video data from the beginning of the video at the maximum rate R_{max} until it catches up with the playback point, say s, currently being multicast by the static channel at time $t_m + s$. At that instant, the client can then release the dynamic channel and continue receiving data from the static channel for the rest of the session. Similar to the SCP algorithm, the video is processed offline to extract video data exceeding the client access bandwidth into a prefetch block P, which is then multicast periodically at the rate R_{max}. The client will prefetch block P before requesting for a dynamic channel to begin playback.

Unlike SCP however, the client in DCP can always begin playback once a dynamic channel is available. The client does not cache video data from a static channel until the dynamic channel catches up with the playback point currently being broadcast by the static channel. When the dynamic channel becomes available at time t_1 and releases at time (t_m+s), the client would have received video data of size $R_{max} \cdot (t_m + s - t_1)$, while missed video data from playback point 0 to s of size

given by $\int_0^s v(\tau)d\tau$. So, to determine this switchover point, we need to find s that satisfies the following equation:

$$R_{max} \cdot (t_m + s - t_1) = \int_0^s v(\tau)d\tau, \text{ where } t_m + s \geq t_1 \tag{4.9}$$

The dynamic channel in DCP will consume more resource than its SCP counterpart. In particular, the dynamic channel itself is streamed at the maximum access bit-rate (i.e., R_{max}). Also, the client cannot cache video data from the static channel while the dynamic channel is streaming, thus increasing the time it takes to catch up with the static channel. Both factors increase the dynamic channel's bandwidth consumption.

To determine the client buffer requirement, we first consider the amount of data being received from the dynamic channel. At time t_1, the dynamic channel starts streaming data to the client at the rate R_{max} up to the time (t_m+s) when the dynamic channel is released. The accumulated amount of data received from the dynamic channel by time t where $t_1 \leq t \leq (t_m + s)$ is given by

$$r(t) \cdot (t - t_1) \tag{4.10}$$

where $r(t)$ equals R_{max} for $t_1 \leq t \leq (t_m + s)$ or 0 otherwise. After the dynamic channel finishes, i.e. time $t \geq (t_m + s)$, the client will cache data from the static channel and thus the accumulated amount of data received by time t where $(t_m + s) \leq t \leq (t_m + L)$ is given by $\int_{t_m+s}^{t} v_r(\tau - t_m)d\tau$, where $v_r(\tau)$=$v(\tau)$ for $s \leq \tau \leq L$ or 0 otherwise because video data of playback point before s will not be received from the static channel. Therefore the total amount of data received by time t where $t_1 \leq t \leq (t_m + L)$ is then given by

$$\int_0^L v_p(\tau)d\tau + r(t)\cdot(t-t_1) + \int_{t_m+s}^{t} v_r(\tau-t_m)d\tau \tag{4.11}$$

Now as the client begins playback at time t_1, the accumulated amount of data consumed by time t is given by $\int_{t_1}^{t}\left[v(\tau-t_1)+v_p(\tau-t_1)\right]d\tau$. Thus the excess amount of video data received but not yet played back at time t is given by

$$\begin{aligned} U_{dcp}(t_1,t) &= \left(\int_0^L v_p(\tau)d\tau + r(t)\cdot(t-t_1) + \int_{t_m+s}^{t} v_r(\tau-t_m)d\tau \right) \\ &\quad - \left(\int_{t_1}^{t} v(\tau-t_1)d\tau + \int_0^{t-t_1} v_p(\tau)d\tau \right) \\ &= \int_{t_m+s}^{t} v_r(\tau-t_m)d\tau + \int_{t_1}^{t}\left[r(t)-v(\tau-t_1)\right]d\tau + \int_{t-t_1}^{L} v_p(\tau)d\tau \end{aligned} \tag{4.12}$$

The maximum of (4.12) thus determines the client buffer requirement

$$U_{dcp}^{\max} = \max\left\{U_{dcp}(t_1,t),\ \forall t_1,t\ |t_m \le t_1 \le (t_m+T_R),\ t_1 \le t \le (t_1+d_s+L)\right\} \tag{4.13}$$

Chapter 5

TURBO-SLICE-AND-PATCH

The two priority scheduling algorithms presented in the previous section have their pros and cons. In this section, we present the turbo-slice-and-patch (TSP) algorithm that combines the virtues of the static channel priority and the dynamic channel priority algorithms. In TSP, we divide the video stream into three portions (i.e. slicing) and admit clients using a three-phase patching process (i.e. patching). The following sections present the algorithm in detail.

5.1 Video Pre-processing

Before a video is put online for streaming, two offline processing steps are performed, namely temporal smoothing and slicing of the video. First, we apply temporal smoothing [16] to reduce the video's peak bit-rate. However, experiments show that temporal smoothing may also increase bandwidth consumption during the patching process. This is because temporal smoothing employs work-ahead to aggressively stream video data to the client as long as buffer allows. Consequently, this work-ahead mechanism will substantially increase the transmission rate of the video's initial portion, thus increasing the time to complete the patching process.

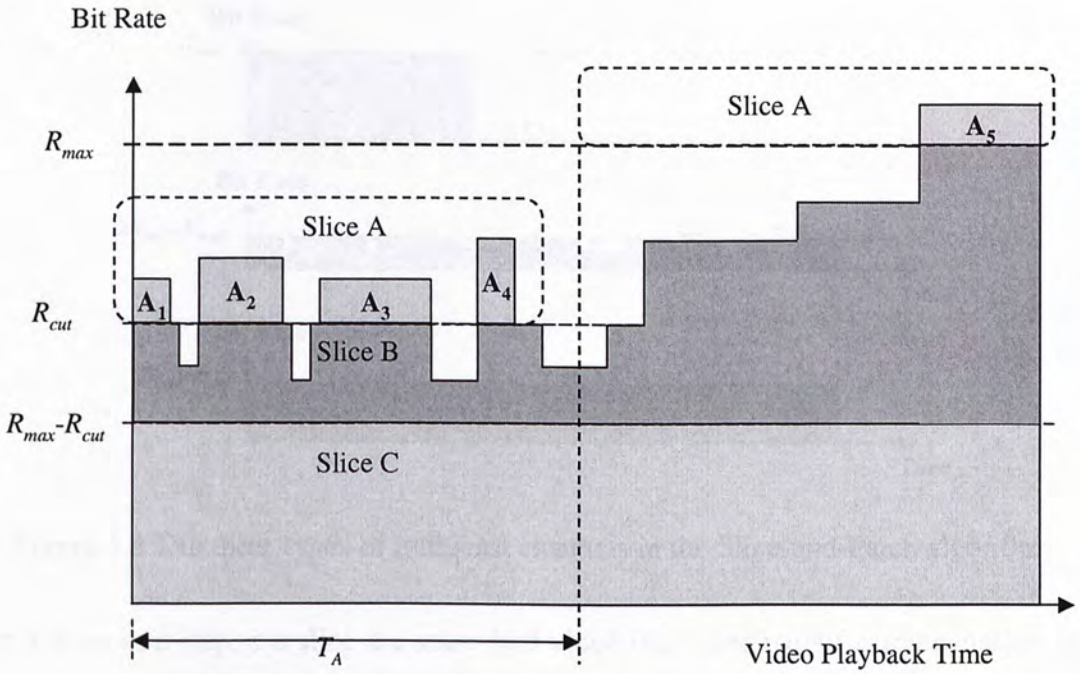

Figure 5.1 Video slicing in the Slice-and-Patch algorithm.

To tackle this problem, we divide the video into two segments and then perform temporal smoothing to these two segments independently. The first video segment comprises video data from the beginning to the playback point T_A given by

$$T_A = \left(\frac{R_{max}}{R_{max} - R_{cut}} \right) T_R \tag{5.1}$$

where T_R is the repeating interval for the static multicast channels. The physical meaning of T_A is the latest possible playback point when the three-phase patching process will end. We will derive T_A in Section 5.3 after we have presented the three-phase patching process.

The rest of the video data then form the second video segment. This two-segment smoothing process can substantially reduce the initial transmission bit-rate as the work-ahead algorithm will not transmit ahead of time video data beyond the playback point T_A. To simplify discussions, we will refer to the smoothed video bit-rate simply as the video bit-rate.

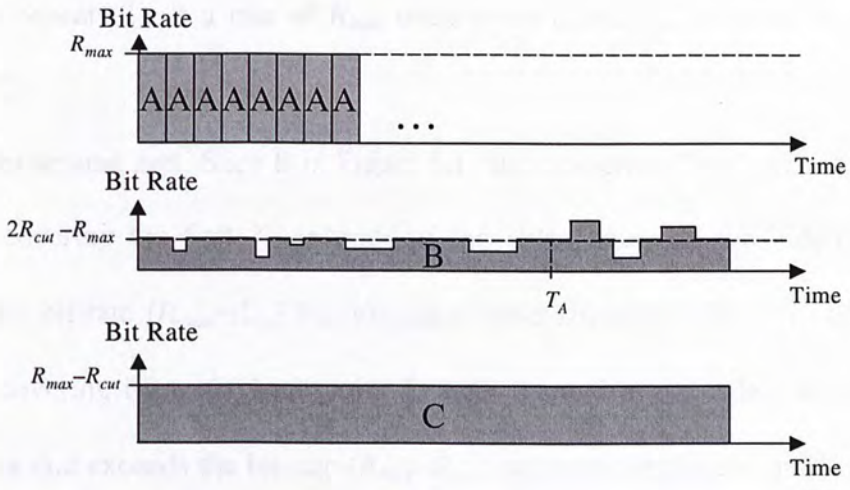

Figure 5.2 The three types of multicast channels in the Slice-and-Patch algorithm.

In the second step, we slice the smoothed video into three parts for transmission in three separate multicast channels. As depicted in Figure 5.1, the video data stream is sliced at two bit-rate thresholds: R_{cut} and $(R_{max}-R_{cut})$, where R_{max} is the maximum access bandwidth of the client and R_{cut} is a system parameter configurable from R_V to $(2/3)R_{max}$.

The first part, Slice A, comprises two portions. The first portion includes video data exceeding the bit-rate R_{cut} (e.g. A_1, A_2, etc., in Figure 5.1) from the beginning of the video until the playback point T_A given by (5.1). The purpose of this slicing is to reduce the peak rate of the video stream to prevent congesting the client's access channel during patching. The second portion includes video data exceeding the client access bandwidth R_{max} from the playback point T_A to the end of the video.

This portion is similar to the prefetch block in the SCP/DCP algorithms and the purpose is to keep the video streaming bit-rate within the client access bandwidth limit. Let the size of this first video data block be A Mb. It will be

multicast repeatedly at a rate of R_{max} once every $d_1=A/R_{max}$ seconds as shown in Figure 5.2.

The second part, Slice B in Figure 5.1, also comprises two portions. The first portion, covering the first T_A seconds of the video, includes the video data that exceed the bit-rate (R_{max}–R_{cut}) but excludes those already in Slice A. The second portion, covering from playback point T_A until the end of the video, comprises *all* video data that exceeds the bit-rate (R_{max}–R_{cut}) except those already in Slice A. This slice will be multicast repeatedly over a separate multicast channel following the *actual* video bit-rate (as opposed to the constant transmission rate for Slice A) as shown in Figure 5.2.

Lastly, the third part, Slice C in Figure 5.1, comprises the rest of the video data not included in Slice A and Slice B. This slice will be multicast repeatedly over a third multicast channel following the *actual* video bit-rate as shown in Figure 5.2.

5.2 Bandwidth Allocation

Let B_{max} be the total server (or network, whichever is smaller) bandwidth allocated for a video of average bit-rate R_V bps and length L seconds. First, a bandwidth of R_{max} will be allocated for multicasting Slice A. Then the remaining bandwidth will be equally divided between the static multicast channels and dynamic multicast channels. Simulation results have shown that this equal allocation results in the best performance (c.f. Section 7.6).

There are two types of static multicast channels, one type transmitting Slice B and the other transmitting Slice C. As the numbers of these channels are equal, we will refer to a pair of such channels as a static multicast channel. Unlike the case of CBR videos, a static multicast channel in TSP does not occupy a constant amount of

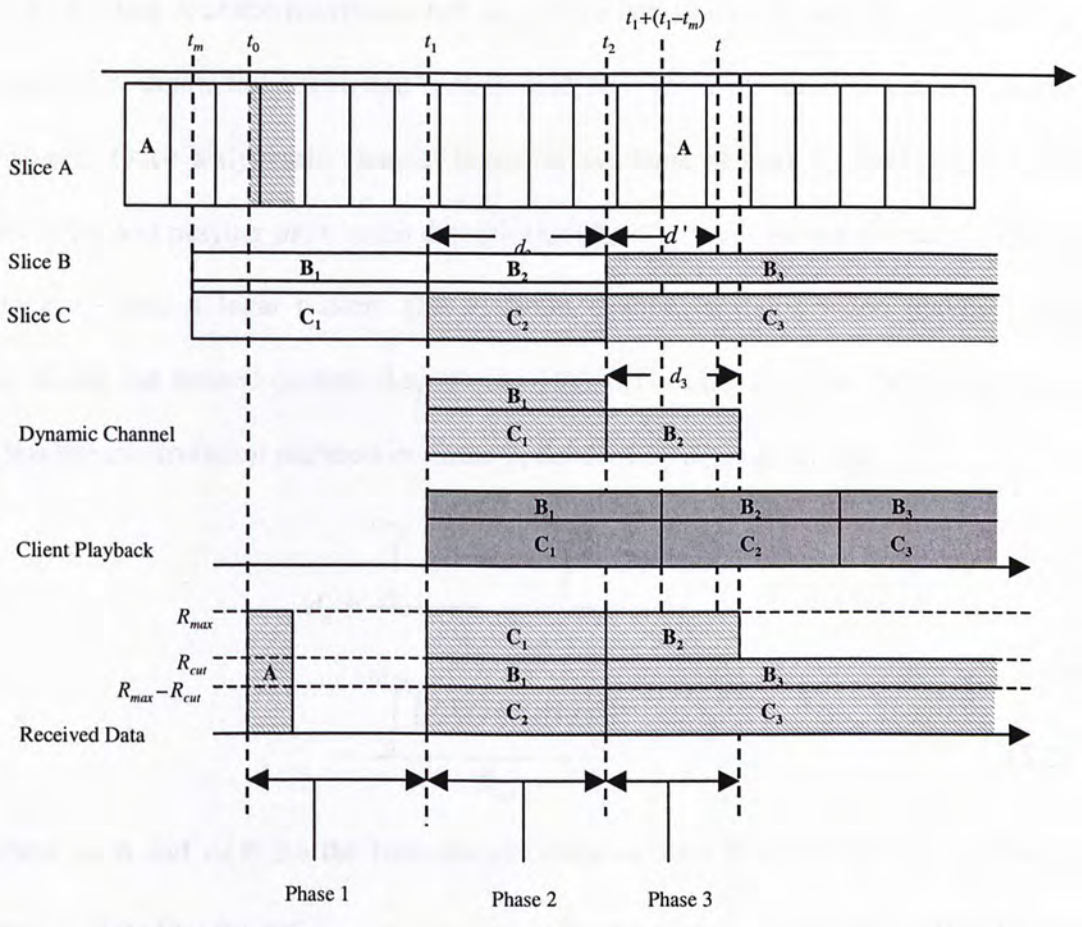

Figure 5.3 The three-phase patching process in the Slice-and-Patch algorithm.

bandwidth. Therefore offline numerical procedures are used to compute the maximum number of static multicast channels that can fit within the bandwidth limit $(B_{max}-R_{max})/2$. The remaining bandwidth will be used by the dynamic channels to patch newly admitted users and then merge them to one of the static multicast channels. Once the merging is completed, the user will not incur any additional load to the server for the rest of the video streaming session.

5.3 Three-Phase Patching

A new client goes through a three-phase patching process to begin a new video streaming session. Let the client arrive at time t_0. It immediately enters Phase 1 by

caching Slice A at the maximum rate R_{max} for a duration of d_1 seconds as shown in Figure 5.3. Next, the client will request and wait for a dynamic channel to begin Phase 2. Once a dynamic channel becomes available at time t_1, the client begins receiving and playing back video data blocks $\{B_1, C_1\}$ while simultaneously caching block C_2 into a local buffer. The dynamic channel sustains video playback by streaming the missed content (i.e., blocks $\{B_1, C_1\}$) to the client at the bit-rate R_{cut}. Thus the transmission duration in Phase 2, denoted by d_2, is given by

$$d_2 = \frac{\int_0^{t_1-t_m} v_b(\tau)d\tau + \int_0^{t_1-t_m} v_c(\tau)d\tau}{R_{cut}} = \frac{\int_0^{t_1-t_m} [v_b(\tau)+v_c(\tau)]d\tau}{R_{cut}} \quad (5.2)$$

where $v_b(\tau)$ and $v_c(\tau)$ are the transmission rates of Slice B and Slice C at playback point τ. Note that for $\tau \le T_A$, $v_b(\tau)+v_c(\tau)$ is bounded by R_{cut} according to the slicing procedure described in Section 5.1. Due to the rate varying nature of the video, it is possible that the combined bit-rate $v_b(\tau)+v_c(\tau)$ is lower than R_{cut} for some playback points. Thus, the transmission duration, denoted by d_2, cannot be larger than the length of video to be patched (t_1-t_m):

$$d_2 \le (t_1 - t_m) \quad (5.3)$$

By the end of Phase 2, the client will have already cached block C_2 and completed playback of blocks $\{B_1, C_1\}$. However, due to the limited client access bandwidth, the client cannot cache block B_2 and thus in Phase 3 the server will use the dynamic channel to stream block B_2 at the bit-rate $(R_{max}-R_{cut})$ to sustain continuous video playback. Concurrently, the client continues to cache data (e.g. B_3, C_3) from the

static channel for the rest of the video session. The duration of Phase 3, denoted by d_3, is given by

$$d_3 = \frac{\int_{t_1-t_m}^{t_1-t_m+d_2} v_b(\tau)d\tau}{R_{max} - R_{cut}} \tag{5.4}$$

where the numerator is the size of block B_2 and the denominator is the transmission rate. To deduce the upper bound for the transmission duration d_3, we would first show that the bit-rate of slice B in the first T_A seconds is actually bounded by (R_{max}–R_{cut}) in the following lemma.

Lemma 1: If $R_{cut} \le \frac{2}{3}R_{max}$, then $v_b(\tau) \le (R_{max} - R_{cut})$ for all $\tau \le T_A$.

Proof: Firstly, $(R_{max} - R_{cut}) \ge (R_{max} - \frac{2R_{max}}{3}) = \frac{R_{max}}{3}$. Secondly, according to the slicing procedure, the bit-rate of slice B is bounded by R_{cut}–(R_{max}–R_{cut}) (Please refer to Figure 5.1 and Figure 5.2). Thus,

$$\begin{aligned} v_b(\tau) &\le R_{cut} - (R_{cut} - R_{max}) \\ &= 2R_{cut} - R_{max} \\ &\le 2(\frac{2R_{max}}{3}) - R_{max} \\ &= \frac{R_{max}}{3} \end{aligned} \tag{5.5}$$

So, Lemma 1 follows. From (5.4), we can now deduce the upper bound for the transmission duration d_3

$$\begin{aligned} d_3 &= \frac{\int_{t_1-t_m}^{t_1-t_m+d_2} v_b(\tau)d\tau}{R_{max} - R_{cut}} \\ &\le \frac{\int_{t_1-t_m}^{t_1-t_m+d_2} (R_{max} - R_{cut})d\tau}{R_{max} - R_{cut}} \\ &= d_2 \end{aligned} \tag{5.6}$$

With the three-phase patching process defined, we can proceed to derive T_A used earlier in Section 5.1.

Recall that (c.f. Section 5.1) pre-processing of the video depends on the duration of the three-phase patching process, which in turns depends on the client arrival time. We first define $D(t_1)$ to be the length of time from t_m, i.e. the start of the multicast cycle, to the end of Phase 3 given that the dynamic channel is available at time t_1. This time interval comprises three parts. The first part is the time from t_m to the time the dynamic channel becomes available, having a length of (t_1-t_m) seconds. The second part is the transmission duration of Phase 2 as given by (5.2). The last part is the duration of Phase 3 as given by (5.4). Thus we can express $D(t_1)$ as

$$D(t_1)=(t_1-t_m)+d_2+d_3 \tag{5.7}$$

To determine the maximum duration of this interval, i.e., T_A, we first note that (t_1-t_m) is upper-bounded by T_r because it is the maximum time to the next multicast cycle. Thus if the dynamic channel is not available before then, the client can simply receive video data from the new multicast cycle to begin video playback. For the length of Phase 2, we note that the combined bit-rate $v_b(t)+v_c(t)$ cannot exceed R_{cut} due to the slicing procedure. Thus, the maximum length of Phase 2 can be computed from

$$\begin{aligned} d_2 &= \frac{\int_0^{t_1-t_m} \left(v_b(t)+v_c(t)\right) d\tau}{R_{cut}} \\ &\le \frac{\int_0^{t_1-t_m} R_{cut}\, d\tau}{R_{cut}} \\ &= t_1 - t_m \\ &\le T_r \end{aligned} \tag{5.8}$$

Similarly, the maximum length of Phase 3 is equal to the maximum size of segment B_2 (see Figure 5.3) divided by the transmission rate (R_{max}–R_{cut}):

$$
\begin{aligned}
d_3 &= \frac{\int_{t_1-t_m}^{t_1-t_m+d_2} v_b(\tau)d\tau}{R_{max}-R_{cut}} \\
&\le \frac{\int_{t_1-t_m}^{t_1-t_m+d_2} (2R_{cut}-R_{max})\,d\tau}{R_{max}-R_{cut}} \qquad \text{since } v_b(\tau) \le (2R_{cut}-R_{\max}) \\
&= \frac{(2R_{cut}-R_{max})\cdot d_2}{R_{max}-R_{cut}} \\
&\le \frac{(2R_{cut}-R_{max})}{R_{max}-R_{cut}} T_r \qquad \text{due to (5.8)}
\end{aligned} \tag{5.9}
$$

Finally, substituting (5.8) and (5.9) into the R.H.S. of (5.7) gives the desired result:

$$
\begin{aligned}
D(t_1) &= (t_1-t_m)+d_2+d_3 \\
&\le T_r + T_r + \frac{2R_{cut}-R_{max}}{R_{max}-R_{cut}} T_r \\
&= \left(\frac{2(R_{max}-R_{cut})}{R_{max}-R_{cut}} + \frac{2R_{cut}-R_{max}}{R_{max}-R_{cut}} \right) T_r \\
&= \frac{R_{max}}{R_{max}-R_{cut}} T_r = T_A
\end{aligned} \tag{5.10}
$$

5.4 Client Buffer Requirement

In this section, we derive the maximum client buffer size needed under the TSP algorithm by considering the buffer required during the three phases of patching. First, in Phase 1 the client prefetches slice A and thus the amount of buffer required is simply equal to the size of slice A, i.e., $\int_0^{T_A} v_a(\tau)d\tau$.

Phase 2 begins at time t_1 when the dynamic channel is available and finishes at time $t_2=t_1+d_2$. The client in this phase receives two streams of video data, one from the dynamic channel at the rate R_{max} and the other from the static channel

multicasting slice C. Thus, the total amount of data received at any time during Phase 2 (i.e., $t_1 \le t \le t_2$) is given by

$$\int_0^{T_A} v_a(\tau)d\tau + \int_{t_1}^{t} R_{max}d\tau + \int_{t_1}^{t} v_c(\tau)d\tau \tag{5.11}$$

Phase 3 begins at time t_2 and finishes at time t_3=t_1+d_2+d_3. In this phase, the dynamic channel streams B_2 (see Figure 5.3) to the client at the rate (R_{max}–R_{cut}). Concurrently, the client also caches data from the static channels multicasting slice B and slice C. Thus, the total amount of data received at any time t during Phase 3 (i.e., $t_2 \le t \le t_3$) is the sum of the total amount of data received up to time t_2 given by (5.11) plus the amount received during this phase up to time t, giving a total of

$$\begin{aligned}&\int_0^{T_A} v_a(\tau)d\tau + \int_{t_1}^{t_2} R_{max}d\tau + \int_{t_1}^{t_2} v_c(\tau)d\tau + \int_{t_2}^{t} (R_{max} - R_{cut})d\tau + \int_{t_2}^{t} \big(v_b(\tau) + v_c(\tau)\big)d\tau \\ &= \int_0^{T_A} v_a(\tau)d\tau + \int_{t_1}^{t} R_{max}d\tau + \int_{t_1}^{t} v_c(\tau)d\tau + \int_{t_2}^{t} v_b(\tau)d\tau - \int_{t_2}^{t} R_{cut}d\tau \end{aligned} \tag{5.12}$$

After Phase 3 completes, i.e. at time t_3, the dynamic channel will be released and the client will continue caching both slice B and slice C from the static channel. So, the total amount of data received at any time t where $t_3 \le t \le t_m + L$ is simply the sum of the total amount of data received up to time t_3 given by (5.12) and that received after time t_3 giving a total amount of

$$\begin{aligned}&\int_0^{T_A} v_a(\tau)d\tau + \int_{t_1}^{t_3} R_{max}d\tau + \int_{t_1}^{t_3} v_c(\tau)d\tau + \int_{t_2}^{t_3} v_b(\tau)d\tau - \int_{t_2}^{t_3} R_{cut}d\tau + \int_{t_3}^{t} \big(v_b(\tau) + v_c(\tau)\big)d\tau \\ &= \int_0^{T_A} v_a(\tau)d\tau + \int_{t_1}^{t_3} R_{max}d\tau + \int_{t_1}^{t} v_c(\tau)d\tau + \int_{t_2}^{t} v_b(\tau)d\tau - \int_{t_2}^{t_3} R_{cut}d\tau \end{aligned} \tag{5.13}$$

To simplify notations, let

$$r(t) = \begin{cases} R_{max} & \text{if } t_1 \le t \le t_2 \\ R_{max} - R_{cut} & \text{if } t_2 \le t \le t_3 \\ 0 & \text{otherwise} \end{cases} \tag{5.14}$$

and

$$v_r(t) = \begin{cases} v_c(t - t_m) & \text{if } t_1 \le t \le t_2 \\ v_b(t - t_m) + v_c(t - t_m) & \text{if } t_2 \le t \le t_m + L \\ 0 & \text{otherwise} \end{cases} \tag{5.15}$$

Then the sum of (5.11) to (5.13), i.e., the amount of data received by the client at any time t where $t_1 \le t \le t_m + L$ can be expressed as

$$\int_0^{T_A} v_a(\tau)d\tau + \int_{t_1}^{t} r(t)d\tau + \int_{t_1}^{t} v_r(\tau)d\tau \tag{5.16}$$

Now as playback starts at t_1, the accumulated amount of data consumed by time t is given by $\int_0^{t-t_1} v(\tau)d\tau$. Thus the excess amount of video data received but not yet played back at time t where $t_1 \le t \le t_m + L$ is given by

$$U_{tsp}(t_1, t) = \int_0^{T_A} v_a(\tau)d\tau + \int_{t_1}^{t} r(\tau)d\tau + \int_{t_1}^{t} v_r(\tau)d\tau - \int_0^{t-t_1} v(\tau)d\tau \tag{5.17}$$

and the maximum of (5.17) thus determines the client buffer requirement:

$$U_{tsp}^{\max} = \max\left\{U_{tsp}(t_1, t),\ \forall t_1, t \mid t_m \le t_1 \le (t_m + T_r),\ t_1 \le t \le (t_1 + L)\right\} \tag{5.18}$$

Chapter 6

PLAYBACK CONTINUITY

In the three-phase patching process, the client does not always receive video data according to the playback sequence. Consider the example in Figure 5.3, the client receives in Phase 2 video segments B_1, C_1, and C_2 simultaneously but C_2 is not played back until Phase 3. Consequently, to guarantee continuous video playback, it is not sufficient to just ensure the reception data rate is not lower than the video playback bit-rate. In the following, we investigate this playback continuity issue and present a proof that TSP can indeed guarantee playback continuity for the entire video duration.

Let $v(\tau)$ be the playback bit-rate of the video at playback point τ where $0 \le \tau \le L$. Let $v(\tau) = v_a(\tau) + v_b(\tau) + v_c(\tau)$ for $0 \le \tau \le L$ where $v_a(\tau)$, $v_b(\tau)$ and $v_c(\tau)$ are the playback bit-rate of slice A, slice B and slice C at playback point τ respectively. Let $c(t)$ be the total amount of *continuous* video data received by the client at time t. Assume the previous multicast of the static channel begins at time t_m and the client starts playback at t_1 as shown in Figure 5.3.Then to guarantee playback continuity we need to ensure that the amount of continuous video data received must always be larger than the amount required for continuous playback, or mathematically we need to establish that

$$c(t) \geq \int_{t_1}^{t} v(\tau - t_1)d\tau \text{ for } t_1 \leq t \leq (L + t_1). \quad (6.1)$$

Note that as playback does not begin until Phase 1 is completed, playback continuity is not affected by Phase 1. To derive $c(t)$ for the rest of the video session, we consider Phase 2 and Phase 3 in turn.

To begin with, we first denote the amount of *continuous* data received up to time t to be $c_a(t)$, $c_b(t)$ and $c_c(t)$ for slice A, slice B and slice C, respectively. Since the client has already received the whole slice A after Phase 1 is completed, $c_a(t) = \int_0^{t-t_1} v_a(\tau)d\tau$ at any time t for $t_1 \leq t \leq L + t_1$.

Phase 2 begins at time t_1 when a dynamic channel becomes available and ends at time $t_2 = t_1 + d_2$ when the dynamic channel has streamed all the missed data to the client. The following theorem proves the playback continuity during Phase 2.

Theorem 1: Video playback is continuous in Phase 2, i.e., $c(t) \geq \int_{t_1}^{t} v(\tau - t_1)d\tau$, for $t_1 \leq t \leq t_2$.

Proof: Consider $c_b(t)$ and $c_c(t)$. Since the dynamic channel streams blocks $\{B_1, C_1\}$ to the client continuously at the rate $R_{cut} \geq v_b(\tau) + v_c(\tau)$, the amount of video data transmitted by the dynamic channel at time t, $t \geq t_1$, can be computed from

$$\begin{aligned}(t - t_1)R_{cut} &= \int_{t_1}^{t} R_{cut}d\tau \\ &= \int_0^{t-t_1} R_{cut}d\tau \\ &\geq \int_0^{t-t_1} [v_b(\tau) + v_c(\tau)]d\tau \end{aligned} \quad (6.2)$$

Next, we note that the dynamic channel streams slice B and slice C in a continuous playback sequence. Thus (6.2) also gives the amount of continuous video data received by the client:

$$\begin{aligned} c_b(t)+c_c(t) &= (t-t_1)R_{cut} \\ &\geq \int_0^{t-t_1}[v_b(\tau)+v_c(\tau)]d\tau \\ &= \int_{t_1}^{t}[v_b(\tau-t_1)+v_c(\tau-t_1)]d\tau \end{aligned} \tag{6.3}$$

Together with slice A already received, we can compute the total amount of continuous data received at any time t during phase 2 from

$$\begin{aligned} c(t) &= c_a(t)+c_b(t)+c_c(t) \\ &= \int_0^{t-t_1} v_a(\tau)d\tau + c_b(t)+c_c(t) \\ &\geq \int_0^{t-t_1} v_a(\tau)d\tau + \int_{t_1}^{t}[v_b(\tau-t_1)+v_c(\tau-t_1)]d\tau \\ &= \int_{t_1}^{t} v_a(\tau-t_1)d\tau + \int_{t_1}^{t}[v_b(\tau-t_1)+v_c(\tau-t_1)]d\tau \\ &= \int_{t_1}^{t}[v_a(\tau-t_1)+v_b(\tau-t_1)+v_c(\tau-t_1)]d\tau \\ &= \int_{t_1}^{t} v(\tau-t_1)d\tau \end{aligned} \tag{6.4}$$

which shows that playback is always continuous during phase 2. ■

Before we proceed to the next theorem, we first derive the latest possible end time of Phase 2. From (5.3), $d_2 \leq (t_1 - t_m)$ because the dynamic channel may transmit slice B and slice C at a bit-rate higher than the playback rate. Therefore if we express t_2 in terms of t_1 and d_2, we can deduce the upper bound of t_2:

$$\begin{aligned} t_2 &= t_1 + d_2 \\ &\leq t_1 + (t_1 - t_m) \end{aligned} \tag{6.5}$$

In the next theorem, we prove that from t_2 up to $t_1+(t_1-t_m)$ within Phase 3, playback is also continuous.

Theorem 2: Playback is continuous, i.e. $c(t) \geq \int_{t_1}^{t} v(\tau - t_1)d\tau$, for $t_2 \leq t \leq t_1+(t_1-t_m)$.

Proof: For $t \geq t_2$, the client will have received the video blocks B_1, C_1, and C_2, and is receiving B_2. Given that the sizes of B_1 and C_1 equal $\int_0^{t_1-t_m} v_b(\tau)d\tau$ and $\int_0^{t_1-t_m} v_c(\tau)d\tau$ respectively, we can obtain the following inequality

$$c_b(t)+c_c(t) \geq \int_0^{t_1-t_m} [v_b(\tau)+v_c(\tau)]d\tau . \tag{6.6}$$

Therefore, we can compute the total amount of continuous video data received at time t from

$$\begin{aligned}
c(t) &= c_a(t)+c_b(t)+c_c(t) \\
&\geq \int_0^{t-t_1} v_a(\tau)d\tau + \int_0^{t_1-t_m} [v_b(\tau)+v_c(\tau)]d\tau \\
&= \int_{t_1}^{t} v_a(\tau-t_1)d\tau + \int_0^{t-t_1} [v_b(\tau)+v_c(\tau)]d\tau + \int_{t-t_1}^{t_1-t_m} [v_b(\tau)+v_c(\tau)]d\tau \quad \because t \leq t_1+t_1-t_m \\
&\geq \int_{t_1}^{t} v_a(\tau-t_1)d\tau + \int_0^{t-t_1} [v_b(\tau)+v_c(\tau)]d\tau \\
&= \int_{t_1}^{t} v_a(\tau-t_1)d\tau + \int_{t_1}^{t} [v_b(\tau-t_1)+v_c(\tau-t_1)]d\tau \\
&= \int_{t_1}^{t} [v_a(\tau-t_1)+v_b(\tau-t_1)+v_c(\tau-t_1)]d\tau \\
&= \int_{t_1}^{t} v(\tau-t_1)d\tau
\end{aligned} \tag{6.7}$$

which shows that playback is continuous for $t_2 \leq t \leq t_1+(t_1-t_m)$. ■

Intuitively, Theorem 2 considers the duration after Phase 2 when the client simply plays back the excess video data transmitted in Phase 2. The client accumulates excess video data because the dynamic channel in Phase 2 may transmit slice B_1 and C_1 at a rate higher than the playback rate.

After Phase 2, the client enters Phase 3 of a duration denoted by d_3. In the following, we prove that playback is also continuous during Phase 3.

Theorem 3: Video playback is continuous in Phase 3, i.e., $c(t) \geq \int_{t_1}^{t} v(\tau - t_1) d\tau$, for $t_2 \leq t \leq t_2 + d_3$.

Proof: We consider two possible cases depending on the total duration of Phase 2 and Phase 3.

Case I: $d_2 + d_3 \leq (t_1 - t_m)$

In this case, $t_2 + d_3 = [(t_1 + d_2) + d_3] \leq [t_1 + (t_1 - t_m)]$, which implies that Phase 3 ends before the time $t_1 + (t_1 - t_m)$. As Theorem 2 has already established playback continuity for $t_2 \leq t \leq t_1 + (t_1 - t_m)$, playback in Phase 3 must also be continuous as well.

Case II: $d_2 + d_3 \geq (t_1 - t_m)$

From Theorem 2, playback is continuous up to $t=t_1+(t_1-t_m)$. So, in the following, we only need to consider the duration $[t_1 + (t_1 - t_m)] \leq t \leq [t_2 + d_3]$ (see Figure 5.3). During this duration, the dynamic channel is streaming block B_2 at a rate of (R_{max}–R_{cut}). Now, according to Lemma 1, the playback rate of block B_2 must be lower than the dynamic channel's transmission rate, i.e., $v_b(\tau) \leq (R_{max} - R_{cut})$.

Thus after a duration of $d' = t - t_2$ from the start of Phase 3, the amount of video data streamed by the dynamic channel is equal to

$$\int_{t_2}^{t_2+d'} (R_{max} - R_{cut})d\tau \geq \int_{t_1-t_m}^{t_1-t_m+d'} v_b(\tau)d\tau \tag{6.8}$$

The continuous data of slice B at time t contains the whole video block B_1 and the part of block B_2 transmitted by the dynamic channel. Since the data is streamed in-order, the amount of continuous video data received d' seconds after Phase 2 is given by

$$\begin{aligned} c_b(t) &= \int_0^{t_1-t_m} v_b(\tau)d\tau + \int_{t_2}^{t_2+d'} (R_{max} - R_{cut})d\tau \\ &\geq \int_0^{t_1-t_m} v_b(\tau)d\tau + \int_{t_1-t_m}^{t_1-t_m+d'} v_b(\tau)d\tau \\ &= \int_0^{t_1-t_m+d'} v_b(\tau)d\tau \end{aligned} \tag{6.9}$$

For slice C, the continuous video data at time t contains the whole video block C_1 streamed by the dynamic channel during Phase 2, the whole video block C_2 cached from the static channel during Phase 2, and the portion of block C_3 transmitted by the dynamic channel. Since the video data is streamed in-order, the total amount of continuous data received d' seconds after Phase 2 is given by

$$\begin{aligned} c_c(t) &= \int_0^{t_1-t_m} v_c(\tau)d\tau + \int_{t_1-t_m}^{t_1-t_m+d_2} v_c(\tau)d\tau + \int_{t_2}^{t_2+d'} v_c(\tau)d\tau \\ &\geq \int_0^{t_1-t_m+d_2} v_c(\tau)d\tau \end{aligned} \tag{6.10}$$

Therefore, the total amount of continuous data received by the client at time t is given by

$$c(t) = c_a(t) + c_b(t) + c_c(t)$$
$$\geq \int_0^{t-t_1} v_a(\tau)d\tau + \int_0^{t_1-t_m+d'} v_b(\tau)d\tau + \int_0^{t_1-t_m+(t_1-t_m)} v_c(\tau)d\tau \qquad (6.11)$$

Now consider the upper limit of the second integral in (6.11)

$$\begin{aligned} & t_1 - t_m + d' \\ &= t_1 - t_m + t - t_2 \\ &= t_1 - t_m + t - (t_1 + d_2) \\ &= t - (t_m + d_2) \\ &\geq t - t_1 \qquad \text{since } d_2 \leq (t_1 - t_m) \text{ by (5.3)} \end{aligned} \qquad (6.12)$$

For the upper limit of the third integral in (6.11)

$$\begin{aligned} & t_1 - t_m + t_1 - t_m \\ &\geq d_2 + d_3 \qquad \text{since } d_2 \leq (t_1 - t_m) \text{ by (5.3) and } d_3 \leq d_2 \text{ by (5.5)} \\ &= t_1 + d_2 + d_3 - t_1 \\ &\geq t - t_1 \qquad \text{since } t \leq t_1 + d_2 + d_3 \end{aligned} \qquad (6.13)$$

We can then rewrite (6.11) as follows:

$$\begin{aligned} c(t) &= c_a(t) + c_b(t) + c_c(t) \\ &\geq \int_0^{t-t_1} v_a(\tau)d\tau + \int_0^{t_1-t_m+d'} v_b(\tau)d\tau + \int_0^{t_1-t_m+(t_1-t_m)} v_c(\tau)d\tau \\ &\geq \int_0^{t-t_1} v_a(\tau)d\tau + \int_0^{t-t_1} v_b(\tau)d\tau + \int_0^{t-t_1} v_c(\tau)d\tau \qquad \text{(from (6.12) and (6.13))} \\ &= \int_0^{t-t_1} v(\tau)d\tau \\ &= \int_{t_1}^{t} v(\tau - t_1)d\tau \end{aligned} \qquad (6.14)$$

which shows that playback is also continuous during Phase 3. ■

Up to now we have proved that playback is continuous from $t=t_1$ up to $t=t_2+d_3$, i.e., covering the period from Phase 1 to Phase 3. In the next theorem, we will show that playback is also continuous from the end of Phase 3 to the end of the video session.

Theorem 4: Video playback is continuous for the rest of the video session after Phase 3, i.e., $c(t) \geq \int_{t_1}^{t} v(\tau - t_1)d\tau$, for $t_2 + d_3 \leq t \leq L + t_1$.

Proof: At any time t after Phase 3, the client has received the whole video blocks B_1, B_2, C_1, and C_2. And at the start of Phase 3, i.e. when $t=t_2$, the client will start caching data from the static channel for blocks B_3 and C_3. Thus, the continuous data so far received for slice B includes blocks B_1, B_2 and B_3 with a total size given by

$$\begin{aligned} c_b(t) &= \int_0^{t_1-t_m} v_b(\tau)d\tau + \int_{t_1-t_m}^{t_1-t_m+d_2} v_b(\tau)d\tau + \int_{t_1-t_m+d_2}^{t-t_m} v_b(\tau)d\tau \\ &= \int_0^{t-t_m} v_b(\tau)d\tau \end{aligned} \tag{6.15}$$

Similarly, the continuous data so far received for slice C includes blocks C_1, C_2 and C_3 with a total size given by

$$\begin{aligned} c_b(t) &= \int_0^{t_1-t_m} v_c(\tau)d\tau + \int_{t_1-t_m}^{t_1-t_m+d_2} v_c(\tau)d\tau + \int_{t_1-t_m+d_2}^{t-t_m} v_c(\tau)d\tau \\ &= \int_0^{t-t_m} v_c(\tau)d\tau \end{aligned} \tag{6.16}$$

Therefore the total amount of continuous data received by the client is given by

$$\begin{aligned} c(t) &= c_a(t) + c_b(t) + c_c(t) \\ &= \int_0^{t-t_1} v_a(\tau)d\tau + \int_0^{t-t_m} v_b(\tau)d\tau + \int_0^{t-t_m} v_c(\tau)d\tau \\ &\geq \int_0^{t-t_1} v_a(\tau)d\tau + \int_0^{t-t_1} v_b(\tau)d\tau + \int_0^{t-t_1} v_c(\tau)d\tau \qquad \because t_1 \geq t_m \\ &= \int_0^{t-t_1} v(\tau)d\tau \\ &= \int_{t_1}^{t} v(\tau - t_1)d\tau \end{aligned} \tag{6.17}$$

which shows that playback is continuous after Phase 3 until the end of the video session. ■

Together, Theorem 1 to Theorem 4 establish the fact that playback continuity is guaranteed by the TSP algorithm for the entire duration of the video session.

Chapter 7

PERFORMANCE EVALUATION

In this section, we present simulation results to evaluate and compare the Static Channel Priority (SCP), Dynamic Channel Priority (DCP), the original Slice-and-Patch (S&P) [1], and the Turbo-Slice-and-Patch (TSP) algorithms investigated in this study. The simulator is developed in C++ using the CNCL simulation library [20]. The VBR video bit-rate traces are measured from 300 DVD videos comprising a wide variety of contents, ranging from full-length movies to documentaries. These video traces exhibit a wide spectrum of properties. For example, the video length ranges from 93 seconds to 14585 seconds, and the video bit-rate ranges from 1.02 Mbps to 9.85 Mbps. Note that the maximum bit-rate in fact exceeds the rate specified by the DVD standard. This is due to the way we measure the bit-rate traces. In particular, we do not measure the video bit-rate directly as most of the DVD bit streams are encrypted. Instead, we measure the I/O activities while playing back the DVD using a hardware MPEG2 decoder. Thus the bit-rate profiles not only capture the variations in the video encoding, but also capture the I/O behavior of the decoder as well.

The server is configured with a bandwidth of $50R_V$ bps and the client an access bandwidth of $2R_{avg}$ Mbps, where R_{avg} is the average bit-rate of the VBR-

encoded video. Each simulation runs for a simulated time of 30 days, with randomized initial condition. The arrival rate of client is 1 request per second. For a video of length 2 hours or 7,200 seconds, this arrival rate represents an average of 7,200 concurrent clients in the system.

In configuring the S&P and the TSP algorithms, which both has a system parameter R_{cut} that affects the system's performance, we simulate 20 values of R_{cut} linearly spaced from R_{avg} to $\frac{4}{3}R_{max}$ and select the one that achieves the lowest average latency. Our results in Section 7.3 show that while the choice of R_{cut} is dependent on the video bit-rate profile, the sensitivity is relatively modest and thus the simple procedure we employed is sufficient to obtain good results.

In the following sections, we first present results obtained from simulating 300 videos to evaluate and compare the algorithms' average latency (Section 7.1) and client buffer requirement (Section 7.2). Next, we show the effect of simulating using more values of R_{cut} (Section 7.3). Then, we investigate the performance variations between different videos by picking three videos from the first quartile, median, and third quartile respectively based on their latency performance, and compare their average latency versus arrival rate (Section 7.4), versus server bandwidth (Section 7.5), and versus the ratio of bandwidth allocated to static channel (Section 7.6).

7.1 Average Latency

Figure 7.1 compares the four algorithms' latency for 300 different videos. The figure is plotted as a cumulative plot. The horizontal axis is the latency increase compared to the CBR case. For example, a latency increase of 10% represents a latency 10%

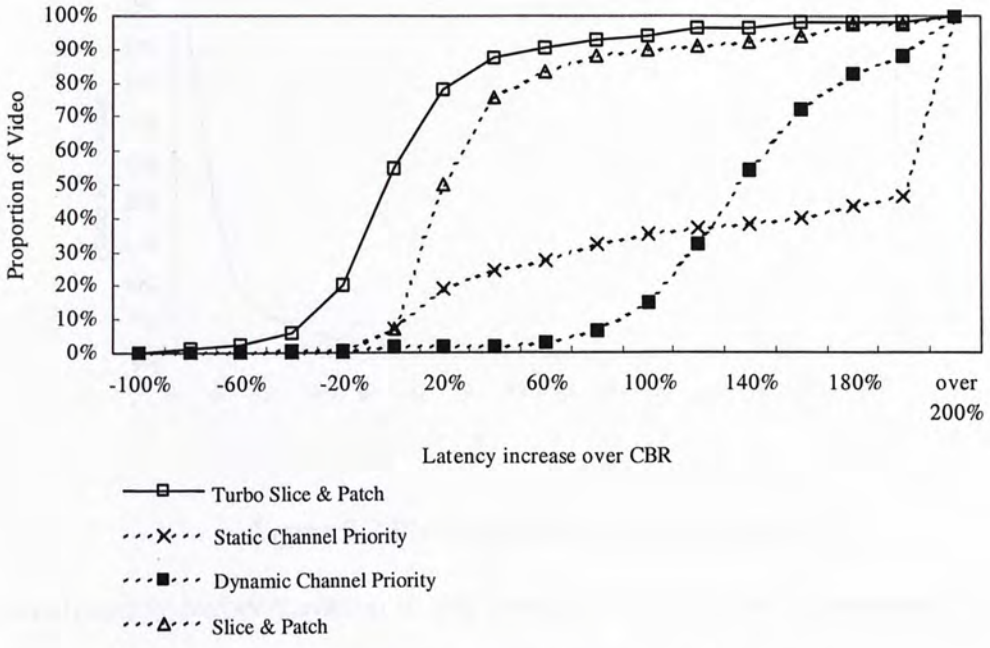

Figure 7.1 Comparison of mean latency for all videos.

longer than the latency achieved by the system streaming a CBR video of the same average bit-rate and duration. Note that a negative latency increase means that the VBR case achieves latency shorter than the CBR equivalent.

There are several observations. First, in terms of the median of latency increase over all 300 videos, SCP performs the worst at 190% (i.e., latency double of the CBR equivalent), DCP at 130%, S&P at 20%, and TSP the best with no increase at all (i.e., 0%). That is, the latency of half of the videos have lower mean latency than the CBR equivalent in TSP. Second, in terms of variations in latency increases, TSP is also the best with a standard deviation of only 37%. By contrast, SCP has the worst variation with a standard deviation of 4,867%, and a maximum latency increase over 2,000%. This shows that the performance of TSP is more robust and less affected by the variations in the video bit-rate profile.

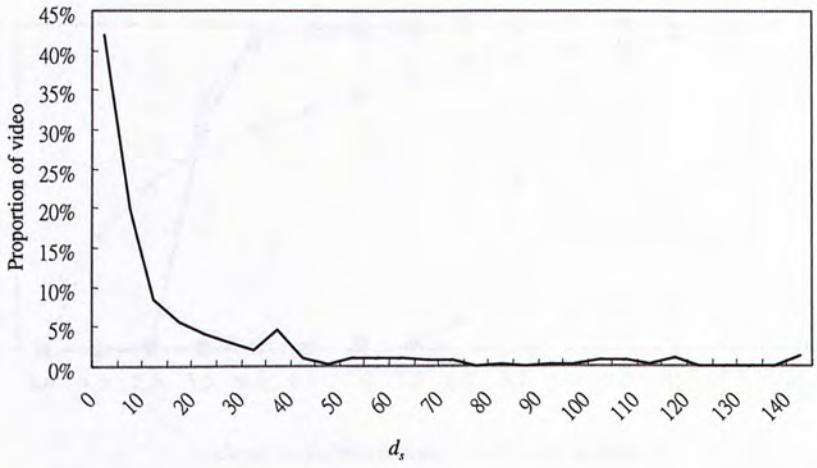

Figure 7.2 Playback delay, d_s, for all videos.

The significantly higher variation in the latency of SCP is due to variations in the bit-rate of the video's initial portion. In particular, the algorithm gives priority to cache from the static channel video data that cannot be used to begin video playback. Therefore if the initial portion of the video has a high bit-rate, then the dynamic channel will take a longer time to cache sufficient video data to begin playback and so lengthens the latency. By contrast, the DCP, S&P, and TSP algorithms are less sensitive to this effect because they allocate more bandwidth to cache video data that can be played back immediately. Figure 7.2. illustrates this problem by plotting the distribution of playback delay of SCP, i.e., the d_s in (4.4). The results show that more than 60% of videos require a playback delay of 5 seconds or more, with a mean as high as 13.9s. Note that this playback delay adds to the latency experienced by the client regardless of the system load, thus significantly degrading SCP's performance.

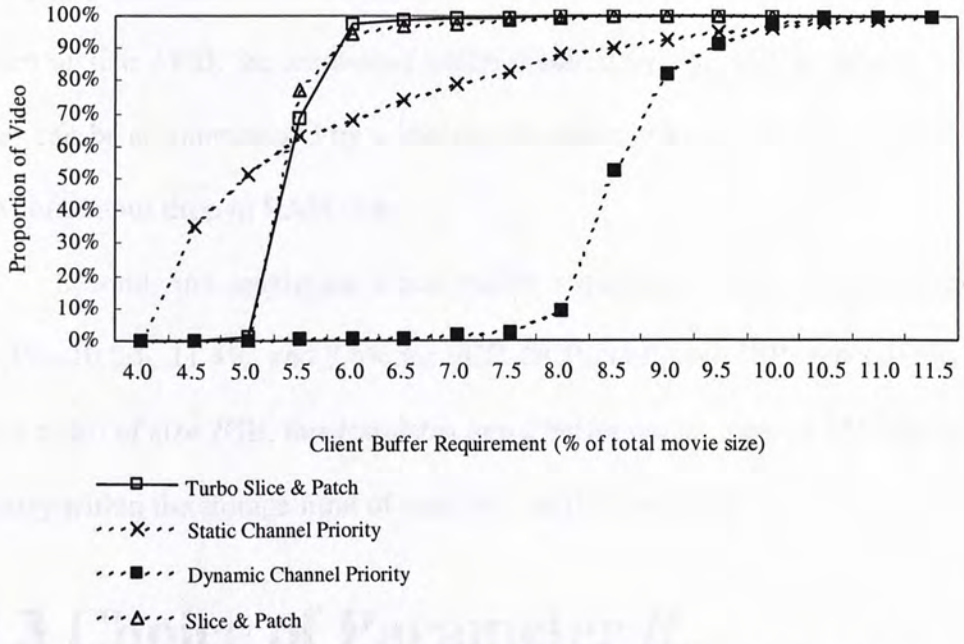

Figure 7.3 Comparison of client buffer requirement.

Third, comparing DCP with TSP, DCP manages to achieve lower latency than TSP in 6 out of the 300 videos. It is possible to devise a rule based on the ratio of the prefetch latency to the length of T_r as an indicator to select between DCP and TSP given a video's bit-rate profile. Our experiments show that such threshold-based selection can correctly pick the better performer 5 out of 6 times. Nevertheless, such a selection process still relies on the appropriate choice of the threshold and thus the accuracy is not guaranteed. Alternatively, since the selection is an offline process, one can always perform simulations of the two algorithms and pick the one with the best performance for use in the system.

7.2 Client Buffer Requirement

Figure 7.3. compares the four algorithms' client buffer requirement. These results are generated based on the assumption that there is a server bandwidth of $50R_{avg}$ bps. We first observe that the client buffer requirement of TSP is mostly within the range of 5% to 6.5% of the video size. The mean client buffer requirement for SCP, DCP,

S&P, and TSP are 5.7%, 8.6%, 5.6%, and 5.4% respectively. For instance, for a video of size 2 GB, the amount of buffer required by TSP will be around 130 MB. This can be accommodated by a low-cost harddisk or even stored in memory given the continuous drop in RAM cost.

Second, the maximum client buffer requirement over all 300 videos are 11.4%, 10.6%, 11.4%, and 7.8% for SCP, DCP, S&P, and TSP respectively. Again for a video of size 2GB, this translates into a buffer requirement of 156 MB for TSP, clearly within the storage limit of even the smallest harddisk.

7.3 Choice of Parameter R_{cut}

So far we have obtained TSP's performance results by picking the value of R_{cut} among 20 samples across the valid range that produces the lowest latency. To investigate the performance impact of this procedure, we repeat the simulations for 20, 40, and 80 samples, and summarize the results in Table 1. We observe that while evaluating more samples will produce lower latency, the difference quickly diminishes. For example, increasing the number of samples from 20 to 80 results in only 2% decrease in latency. Nevertheless, as the process is performed offline, one could afford to trade off simulation time to obtain better performance.

Table 1. Mean of lowest latency for different number of R_{cut} values.

Number of Data Points	**20**	**40**	**80**
Mean Latency (s)	4.59	4.52	4.49

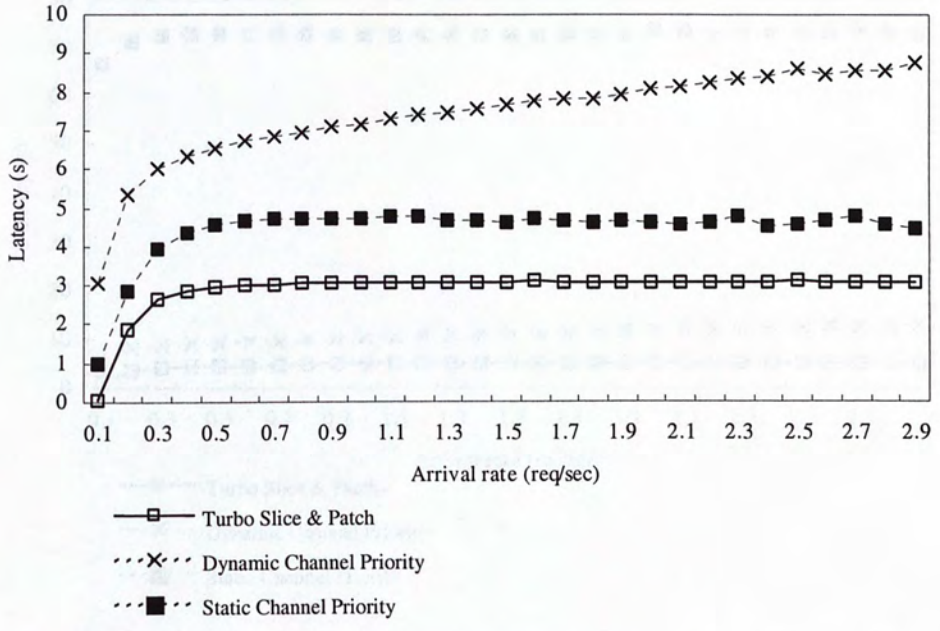

Figure 7.4 Comparison of latency for different arrival rates (Video 1).

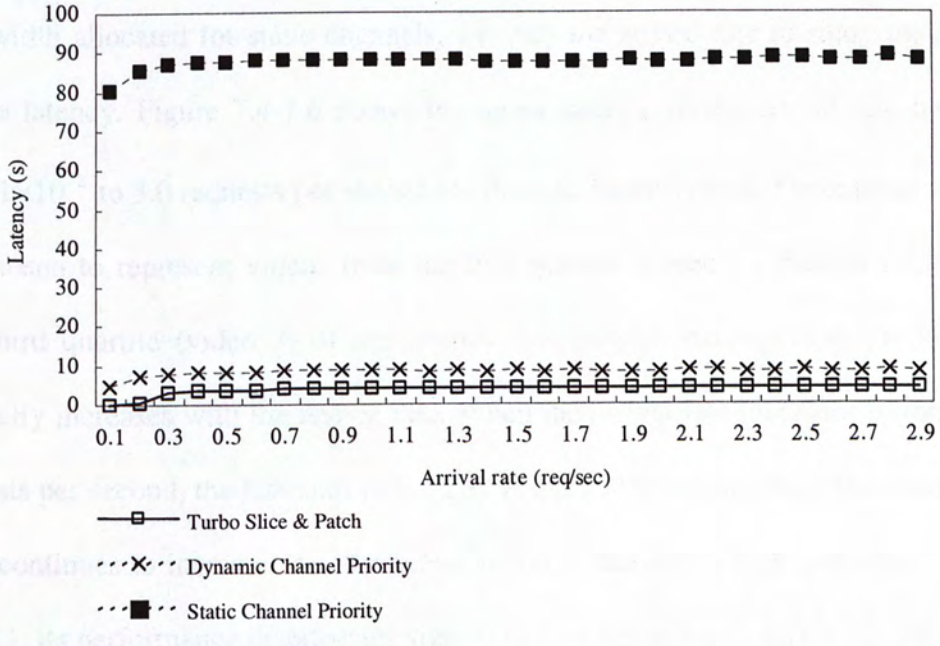

Figure 7.5 Comparison of latency for different arrival rates (Video 2).

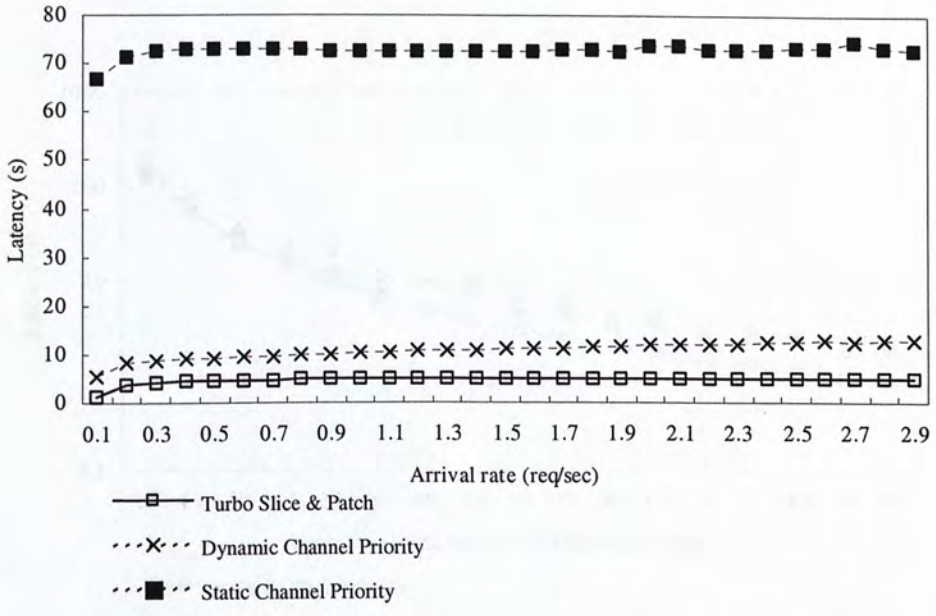

Figure 7.6 Comparison of latency for different arrival rates (Video 3).

7.4 Latency versus Arrival Rate

By fixing the server bandwidth at $50R_{avg}$ bps and a maximum of 50% server bandwidth allocated for static channels, we vary the arrival rate to study the effect on the latency. Figure 7.4-7.6 shows the mean latency versus arrival rate ranging from 1×10^{-2} to 3.0 requests per second for three different videos. These three videos are chosen to represent videos from the first quartile (video 1), median (video 2), and third quartile (video 3) of the latency distribution. As expected, the latency generally increases with the arrival rate. When the arrival rate increases beyond 0.5 requests per second, the latencies of both SCP and TSP level off while the latency of DCP continues to increase. Another observation is that while SCP performs well in video 1, its performance deteriorates significantly when streaming video 2 and video 3. This shows the sensitivity of SCP's performance to the particular video bit-rate profile. The performance of DCP and TSP are more robust in comparison, with TSP achieving the lowest latency in all three cases.

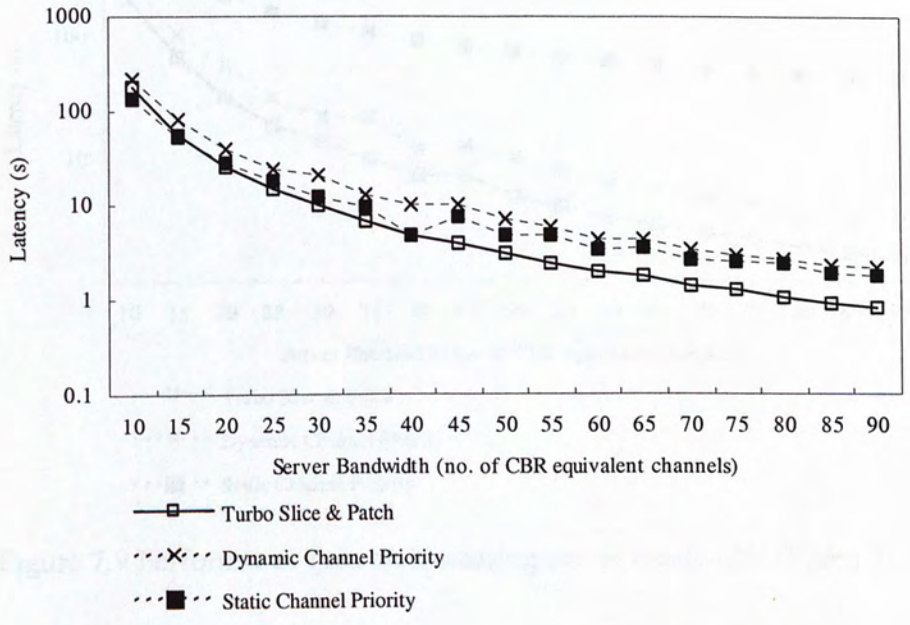

Figure 7.7. Performance gain on increasing server bandwidth (Video 1).

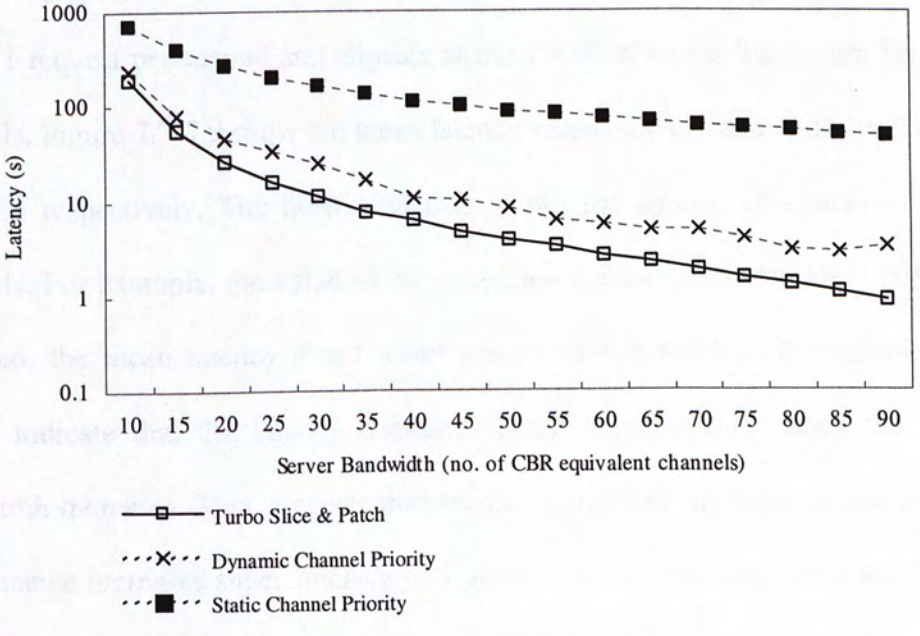

Figure 7.8 Performance gain on increasing server bandwidth (Video 2).

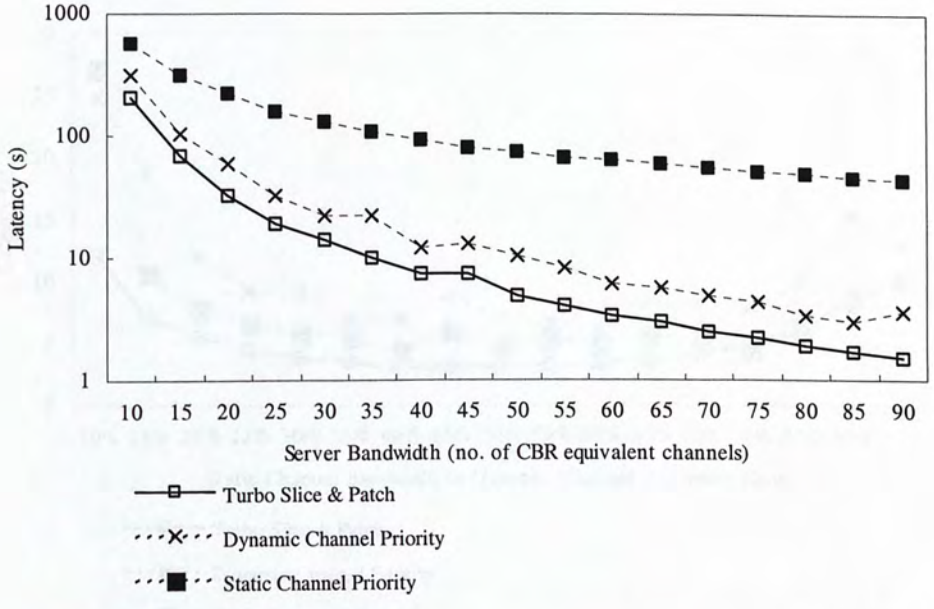

Figure 7.9 Performance gain on increasing server bandwidth (Video 3).

7.5 Server Bandwidth Comparison

To investigate the effect of server bandwidth to the mean latency, we fix the arrival rate at 1 request per second and allocate at most 50% of server bandwidth for static channels. Figure 7.7-7.9 show the mean latency versus server bandwidth for video 1, 2, and 3 respectively. The horizontal axis shows the number of equivalent CBR channels. For example, the value of 50 represents the server bandwidth is $50R_V$. As expected, the mean latency drops when more server bandwidth is available. The results indicate that the latency decreases nearly exponentially when the server bandwidth increases. This suggests that all four algorithms are super-scalar, i.e., the performance increases super-linearly with respect to the resources provided. This is also consistent with the super-scalar property of the original SS-VoD architecture streaming CBR videos [13].

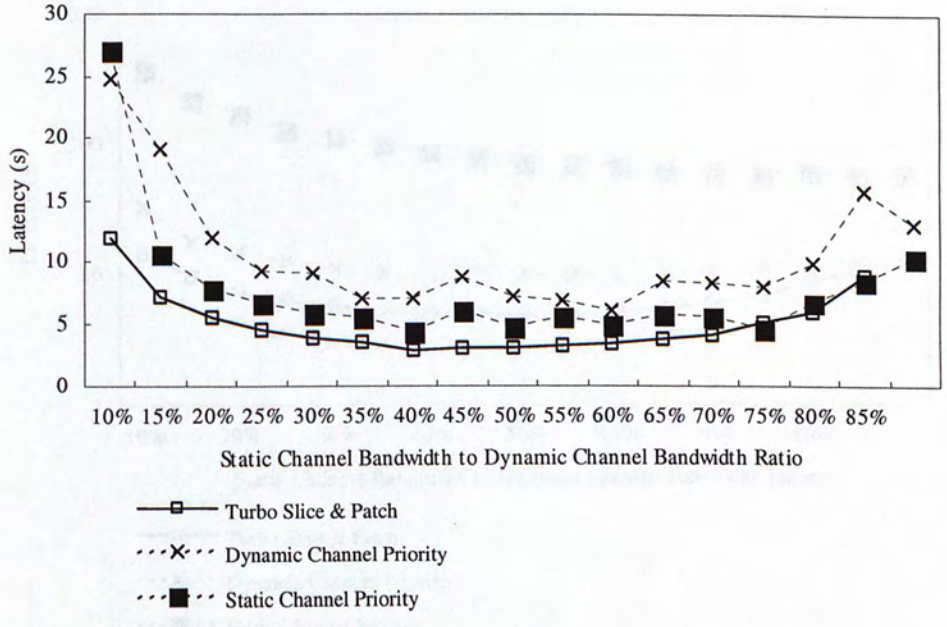

Figure 7.10. Effect of server bandwidth partitioning (Video 1).

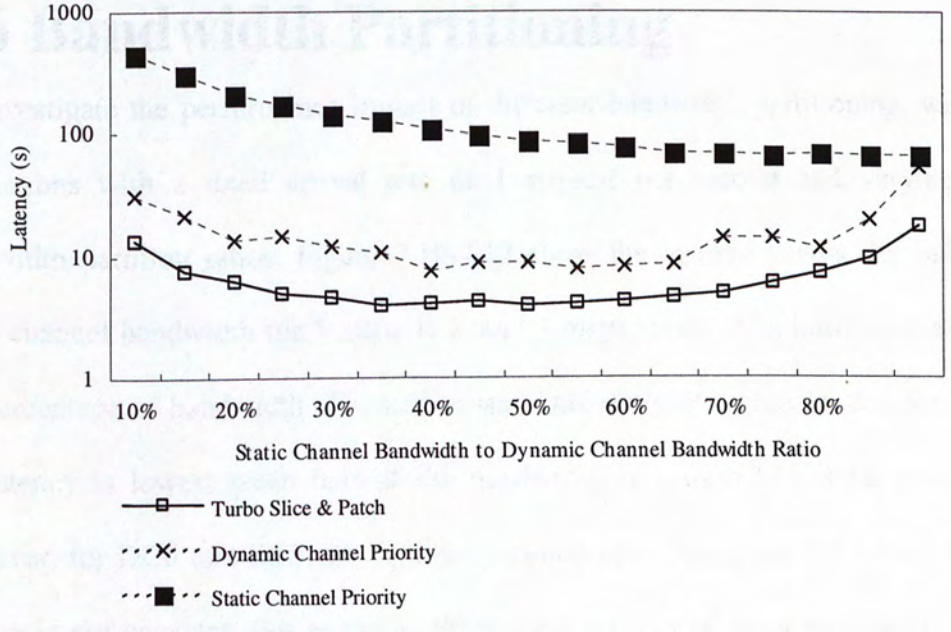

Figure 7.11. Effect of server bandwidth partitioning (Video 2).

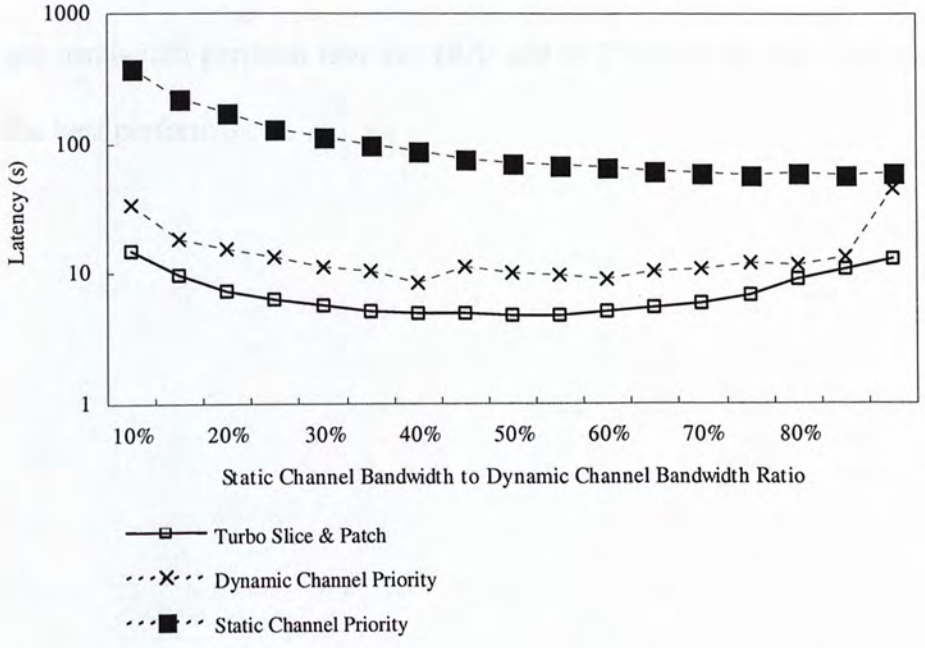

Figure 7.12. Effect of server bandwidth partitioning (Video 3).

7.6 Bandwidth Partitioning

To investigate the performance impact of different bandwidth partitioning, we run simulations with a fixed arrival rate of 1 request per second and varying the bandwidth partition ratios. Figure 7.10-7.12 show the latency versus the ratio of static channel bandwidth for Videos 1, 2, and 3 respectively. The horizontal axis is the percentage of bandwidth allocated to static channels. We observe that for TSP, the latency is lowest when half of the bandwidth is assigned to static channels. However, for DCP and SCP, the optimal partition ratio that gives the lowest mean latency is not constant. For example, there are a number of local minima for both algorithms for video 1. In the other two videos, the latency of SCP decreases with more static channel bandwidth while the optimal partition ratios for DCP is 40% for both Videos 2 and 3. These results suggest that TSP is significantly simpler to deploy in practice as a bandwidth partition ratio of 50% can already provide

consistent performance. By contrast, the service provider will need to determine and adjust the bandwidth partition ratio for DCP and SCP on a video-by-video basis to obtain the best performance.

Chapter 8

CONCLUSIONS

The Turbo-Slice-and-Patch algorithm investigated in this study addresses two challenges in video streaming. First, TSP employs network multicast to achieve a super-linear scalability that is essential to achieving economy-of-scale in provisioning metropolitan-scale video streaming services. Second, TSP employs a novel three-phase slice-and-patch algorithm to support the streaming of VBR-encoded videos with on average only 9% increase in latency. Given that a previous study has shown that one can achieve the visual quality of CBR-encoded videos using only half the bit-rate with VBR encoding, streaming VBR video using TSP in fact requires less resources than streaming CBR videos. With the continuous deployment of multicast in the infrastructure, TSP will serve as a candidate for implementing the future metropolitan video streaming services.

BIBLIOGRAPHY

[1] Kong Chun Wai and Jack Y. B. Lee, "Slice-and-Patch - An Algorithm to Support VBR Video Streaming in a Multicast-based Video-on-Demand System," *Proc. 2002 International Conference on Parallel and Distributed Systems*, Taiwan, Dec 17-20, 2002, pp. 517-529.

[2] A. Dan, D. Sitaram, and P. Shahabuddin, "Scheduling Policies for an On-Demand Video Server with Batching," *Proc. 2nd ACM 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] V.O.K. Li, W. Liao, X. Qui, and E.W.M. Wong, "Performance Model of Interactive Video-on-Demand Systems," *IEEE JSAC*, vol.14(6), Aug 1996, pp.1099-1109.

[5] 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.

[6] K.A. Hua, Y. Cai, and S. Sheu, "Patching: A Multicast Technique For True Video-on-Demand Services," *Proc. 6th International Conf on Multimedia*, Sept 1998, pp.191-200.

[7] Y. Cai, K. Hua, and K. Vu, "Optimizing Patching Performance," *Proc. SPIE/ACM Conference on Multimedia Computing and Networking*, CA, Jan. 1999, pp.204-215.

[8] 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*, Las Vegas, Sep 1997, pp.200-207.

[9] D. Saparilla, K.W. Ross, and M. Reisslein, "Periodic Broadcasting with VBR-Encoded Video" *Proc. IEEE Infocom 1999*, New York City, US, March 1999.

[10] S. Sen, Gao Lixin, and D. Towsley, "Frame-based Periodic Broadcast and Fundamental Resource Tradeoffs," *IEEE International Conference on Performance, Computing, and Communications*, 2001, pp.77-83.

[11] T.C. Chiueh and C.H. Lu, "A Periodic Broadcasting Approach to Video-on-demand Service," *Proc. of SPIE*, Philadelphia, 1996, pp.2615:162-9.

[12] A. Hu, I. Nikolaidis, and P. van Beek, "On the Design of Efficient Video-on-Demand Broadcast Schedules," *Proc. 7th International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems*, Maryland, 1999, pp.262-269.

[13] J.Y.B. Lee and C.H. Lee, "Design, Performance Analysis, and Implementation of a Super-Scalar Video-on-Demand System," *IEEE Transactions on Circuits and Systems for Video Technology*, vol.12(11), November 2002, pp.983-997.

[14] W.S. Tan, N. Duong, and J. Princen, "A Comparison Study of Variable-bit-rate versus Fixed-bit-rate Video Transmission," *Proc. Australian Broadband Switching and Services Symposium*, Australia, 1991, pp.134-141.

[15] W. Feng and S. Sechrest, "Smoothing and Buffering for the Delivery of Pre-recorded Video," *Proc. ISET/SPIE Multimedia Computing and Networking*, San Jose, Feb 1995, pp.234-244.

[16] J.D. Salehi, Z.L. Zhang, J.F. Kurose and D. Towsley, "Supporting Stored Video: Reducing Rate Variability and End-to-End Resource Requirements through Optimal Smoothing," *Proc. of ACM SIGMETERICS*, Philadelphia, May 1996, pp.222-231.

[17] W. Feng, F. Jahanian, and S. Sechrest, "Optimal Buffering for the Delivery of Compressed Pre-recorded Video," *Proc. of the IASTED/ISMM Int'l Conf. on Networks*, Jan. 1995.

[18] W. Feng, Mishra, and Ramakishnan, "A Comparison of Bandwidth Smoothing Techniques for the Transmission of Pre-recorded Compressed Video," *Proc. INFOCOM '97*, vol. 1, Japan, 1997, pp.58-66.

[19] D.Y. Lee and H.Y. Yeom, "Tip Prefetching: Dealing with the Bit-rate Variability of Video Streams" *Proc. of the IEEE ICMCS 1999*, vol. II, Italy, 1999, pp.352-356.

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