JOURNAL OF INFORMATION SCIENCE AND ENGINEERING 19, 517-530 (2003)

Slice-and-Patch – An Algorithm to Support VBR Video Streaming in a Multicast-based Video-on-Demand System*

C. W. KONG AND JACK Y. B. LEE
Department of Information Engineering
The Chinese University of Hong Kong
Shatin, N.T., Hong Kong
E-mail: {cwkong1, yblee}@ie.cuhk.edu.hk

In recent years, a number of sophisticated architectures have been proposed to provide video-on-demand (VoD) service using multicast transmissions. Compared with 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 problem, this paper presents a novel Slice-and-Patch (S&P) algorithm to support VBR video streaming in a multicast VoD system. Extensive trace-driven simulations were conducted to compare the performance of the S&P algorithm with two other algorithms based on priority scheduling. Results show that the S&P algorithm outperforms the other two priority scheduling algorithms for most videos. Compared with the CBR counterpart serving videos of the same average bitrate, the S&P algorithm is able to support VBR video streaming with an increase in latency of only 50%. Given that VBR-encoded video can achieve visual quality comparable to that of CBR-encoded video at half the bitrate, this S&P algorithm can potentially achieve performance comparable to that of CBR-based systems when combined with VBR encoding techniques.

Keywords: multicast, VBR, video-on-demand, streaming, slice-and-patch

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 duration of the 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 of serving 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 problem is to employ intelligent use of network multicast. Network multicast enables a server to send several 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 [1-3], patching [4-7], and periodic broadcasting [8-11].

Received May 15, 2002; accepted July 25, 2002.
Communicated by Biing-Feng Wang, Stephan Olariu and Gen-Huey Chen.
* A preliminary version of the paper was presented at the 2002 International Conference on Parallel and Distributed Systems, Chungli, Taiwan.

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 motion will typically result in lower visual quality than simple video scenes with little movement.

In contrast, videos encoded with constant-quality encoding algorithms have consistent visual quality, at the expense of bitrate variation. A study by Tan *et al*. [12] showed that VBR-encoded video can achieve visual quality similar to that of CBR-encoded video at only half the bitrate. This result suggests that VBR encoding has potential for providing high-quality VoD services. The challenges are the complex resource allocation and the scheduling problems resulting from the video bitrate variation.

This study addresses these challenges and presents a slice-and-patch (S&P) algorithm for allocating resources and scheduling video data transmissions in a multicast VoD system proposed by Lee and Lee [13]. The original multicast VoD system is designed for CBR videos, and combines techniques from batching, patching, and periodic broadcasting. This multicast VoD system can be scaled up to an unlimited number of concurrent users and thus is most suitable for serving popular movies in a metropolitan-scale VoD service. We give a brief overview of this VoD architecture in section 3 and refer interested readers to Lee and Lee [13] for details.

The S&P algorithm is designed based on two principles. First, video data corresponding to video bitrate peaks are prefetched at startup. This step reduces the worst-case peak rate of the video stream and thus allows more efficient resource allocation. Second, the video stream, minus the previously-mentioned peaks, is sliced into two sub-streams of lower bitrates and multicast periodically in two static multicast channels. A client, after prefetching the peaks, initiates patching to begin playback using a dynamically allocated video channel while at the same time caching video data from the static multicast channels. Eventually, video playback reaches the point where video data are already cached and the client can then release the dynamic channel and continue video playback using data received from the multicast channels.

We conducted simulations to study and compare the S&P algorithm with two other algorithms based on priority scheduling. Our simulation results show that the S&P algorithm outperforms the priority scheduling algorithms for most of the 50 tested videos. Compared with the CBR version of the system, the S&P algorithm can serve VBR videos of the same average bitrate with an average latency increase of only 50%. As VBR-encoded video needs only half the bitrate to achieve the same quality as CBR-encoded video, this S&P algorithm can potentially support VBR video streaming with resources comparable to those of CBR-based VoD systems.

The rest of the paper is organized as follows. Section 2 reviews some related works and compares them with this study; section 3 reviews the multicast VoD architecture; section 4 presents the two priority scheduling algorithms; section 5 presents the Slice-and-Patch algorithm; section 6 evaluates and compares the three algorithms using simulation results; and section 7 concludes the paper.

2. BACKGROUND

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

2.1 Previous Work

One of the most well-known solutions for VBR video delivery is temporal smoothing [14-17]. 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 [15] or to minimize the number of rate changes [16]. The schedule can be computed offline and with proper resource reservation, deterministic performance can be guaranteed. Interested readers are referred to Feng *et al.* [17] for a thorough comparison of various smoothing algorithms.

In another study by Lee and Yeom [18], a data prefetch technique was proposed to improve video server performance in serving VBR videos. Unlike smoothing, where all the video data are retrieved from the disk in sequence, data prefetching preloads video data corresponding to a video's bitrate peaks into the server's memory during system initialization. During operation, the server only needs to retrieve the remaining video data from the disk and combine it with the prefetched data for transmission to clients. As the remaining video stream has a lower peak bitrate, 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.* [8] schedules video data transmission using a priority scheduler (Join-the-Shortest Queue). In particular, the server schedules video data transmission according to the demand for data of each channel. A channel with the greatest demand for 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 that a client can receive all the 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, when the client is allowed 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 S&P algorithm investigated in this study, both temporal smoothing and the data prefetch techniques discussed above are orthogonal and complementary. With 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. With the data prefetch technique, the focus is on improving disk retrieval efficiency by intelligently preloading some video data into the server memory. Obviously, this tech-

nique does not affect the transmission schedule at all and thus can be integrated with any transmission scheduling algorithms including S&P.

S&P differs from the work by Saparilla *et al.* [8] in two major ways. First, the S&P algorithm guarantees that no video data will be skipped, thus ensuring visual quality. Second, S&P is targeted at clients with limited access bandwidth (twice the average bit rate of the video). In contrast, the algorithm proposed by Saparilla *et al.* assumes that the client has sufficient bandwidth to receive data from many channels simultaneously, which currently may not be practical.

This study is a first step in exploring algorithms for supporting VBR video delivery in multicast VoD systems. Designing the S&P algorithm has revealed many difficulties and challenges that are not present in conventional unicast VoD systems. Nevertheless, this will be an important area as VBR encoding is necessary to provide good visual quality. In addition, multicast VoD systems may be the only way to deploy cost-effective metropolitan-scale VoD services in the near future.

3. SYSTEM ARCHITECTURE

In this section, we give a brief overview of the multicast VoD architecture, super-scalar VoD (SS-VoD), investigated in this paper. The system consists of a number of service nodes delivering video data over multicast channels to clients. SS-VoD 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 when 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, SS-VoD employs patching to enable a client to start 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.

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 that 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. The video title is multicast repeatedly over all N_S static multicast channels in a time-staggered manner as shown in Fig. 1. Specifically, adjacent channels are offset by

$$T_R = L / N_S \tag{1}$$

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 is

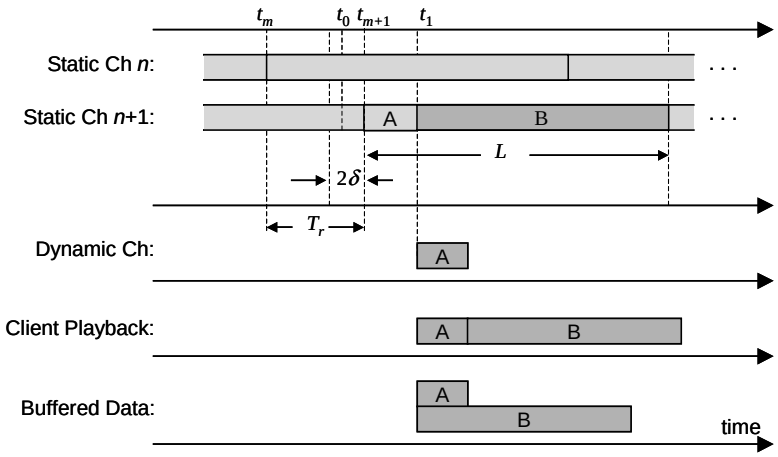


Fig. 1. The patching process in the super-scalar video-on-demand system supporting CBR video.

completed, regardless of the load on the server or how many users are active. These static multicast channels are used as the main channels for delivering video data to clients. A client may start out with a dynamic multicast channel, but it will shortly be merged back into 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, SS-VoD 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 Fig. 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., $w_i = t_{m+1} - t_0 \le 2\delta$. We say that these requests are *statically admitted*. This admission threshold is introduced to reduce the load on 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 caching 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 waiting period, they will be batched together 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 will 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 under heavy loads.

Compared with TVoD systems, a SS-VoD client must have the ability to receive two multicast channels concurrently and have a local buffer that can hold up to T_R seconds of video data. Given a video bitrate of 3Mbps (e.g., high-quality MPEG-4 video), a total downstream bandwidth of 6Mbps 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 today using a small harddisk in the client, and in the near future by simply using memory as technology improves.

3.3 Challenges in Supporting VBR-encoded Video

The SS-VoD architecture was originally designed for CBR videos. A problem arises if we want to support VBR videos. Specifically, even if a client has the ability to receive twice the video bitrate, this may not be sufficient to support two channels of VBR video of the same average bitrate due to bitrate variation. Temporal smoothing can be used to alleviate this problem but cannot solve it completely without adding excessive start-up delay time. We investigate in the next section two possible solutions to this problem based on priority scheduling.

4. PRIORITY SCHEDULING

The primary problem with supporting VBR video in SS-VoD 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 bitrate; then, the client has an access bandwidth of $2R_V$. However, a VBR video of average bitrate R_V will likely have bitrate peaks (valleys) higher (lower) than R_V even after smoothing is applied. It is easy to see that the access channel will become congested whenever peaks from both the dynamic channel and static channel overlap.

Assuming that the client access bandwidth is limited, we need to prioritize the transmission and reception of video data to stay within the given access bandwidth. The following sections present two such priority-scheduling algorithms.

4.1 Static Channel Priority

In the static channel priority algorithm, we let the static channels transmit at the original video bitrate and adjust the transmission rate of the dynamic channel to keep it within the access bandwidth limit. Let $v(t)$ be the video data consumption rate function that defines the rate at which video data are being consumed t seconds after playback has begun. Assume that the client arrives at time t_0, and that the immediate previous multicast cycle begins at time t_m; then, the client will cache video data starting from a playback point of $t_c = t_0 - t_m$, and the amount of access bandwidth left for the dynamic channel at time t will be equal to $u(t) = 2R_V - v(t - t_m)$ for $t \geq t_0$.

As the client will not yet have any video data from a playback point earlier than t_c, a dynamic channel will be allocated to begin streaming data from the beginning of the video to the playback point t_c. If the bandwidth available to the dynamic channel is sufficient to stream the video, i.e., $u(t) \geq v(t - t_0 - w)$ for $(t_0 + w) \leq t \leq (t_0 + w + t_c)$,

$$\int_{t_0+w}^{\tau} u(t)dt \geq \int_{t_0+w}^{\tau} v(t-t_0-w)\,dt, \text{ for } (t_0+w) \leq \tau(t_0+w+t_c), \quad (2)$$

where w is the waiting time for the dynamic channel, then no further action needs to be taken. Otherwise, the client will not be able to begin playback immediately when data are received because playback continuity cannot be sustained when the condition in (2) fails.

To tackle this problem, the client will have to delay the playback by t_s seconds so that the continuity condition will be satisfied:

$$\int_{t_0+w}^{\tau} u(t)dt \geq \int_{t_0+w+t_s}^{\tau} v(t-t_0-w-t_s)\,dt, \text{ for } (t_0+w+t_s) \leq \tau(t_0+w+t_c). \quad (3)$$

This is also the tradeoff for this algorithm.

4.2 Dynamic Channel Priority

To avoid the startup delay in the previous static channel priority algorithm, we can give priority to the dynamic channel during admission. Unlike the previous algorithm, we cannot simply transmit video data of the static channel using the left-over 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, the server will transmit video data from the beginning of the video at the maximum rate $2R_V$ until it catches up with the playback point, say s, currently being multicast by the static channel. At that instant, the client will then be able to release the dynamic channel and continue receiving data from the static channel for the rest of the session.

Similarly, we again invoke the playback continuity condition to find the value of s that satisfies the following condition:

$$2R_V(s-t_0-w) = \int_0^s v(t)\,dt, \text{ where } s \geq (t_0+w). \quad (4)$$

This algorithm does not incur a start-up delay, but a dynamic channel will consume more resources than that in the static priority algorithm for two reasons. First, the dynamic channel itself will be streamed at the maximum access bitrate. Second, the client will not be able to cache video data from the static channel while the dynamic channel is streaming. This will increase the time the dynamic channel takes to catch up with the static channel. Both factors will increase the dynamic channel's bandwidth consumption and result in a longer amount of time spent waiting for an available dynamic channel.

5. SLICE-AND-PATCH

The two algorithms presented in the previous section both have tradeoffs. In this section, we present a slice-and-patch (S&P) algorithm that combines the virtues of the static channel priority and the dynamic channel priority algorithms. In S&P, 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.

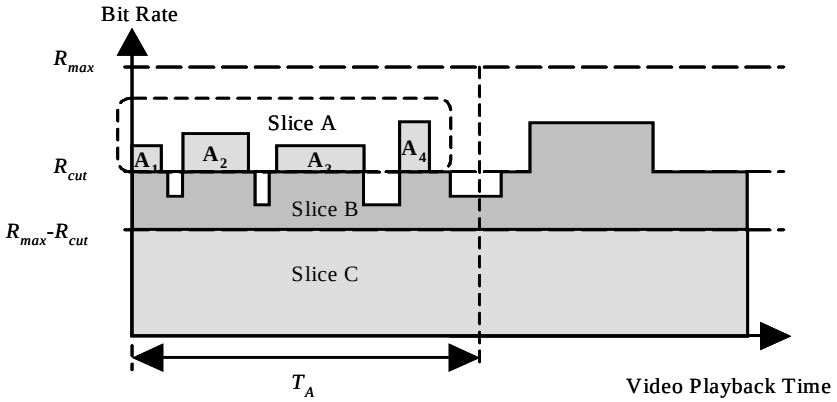


Fig. 2. Video slicing in the Slice-and-Patch algorithm.

5.1 Video Slicing

Video slicing is an offline process that divides a video data stream into three parts (i.e., slices) for transmission in three separate multicast channels. As shown in Fig. 2, the video data stream is sliced at two bitrates: R_{cut} and $R_{max} - R_{cut}$. The parameter R_{cut} is configurable from R_V to (2/3)R_{max} and can be optimized for a particular video.

To generate the first part, Slice A, we collect all the video data exceeding the bitrate R_{cut} (e.g., A_1, A_2, etc., shown in Fig. 2), starting from the beginning of the video and continuing up to the playback point T_A given by

$$T_A = \left(\frac{R_{\max}}{R_{\max} - R_{cut}} \right) T_R , \tag{5}$$

where R_{max} is the maximum access bandwidth of the client, and T_R is the repeating interval for the static multicast channels. We will derive T_A in section 5.3 when we explain the three-phase patching process. The purpose of this slicing step is to reduce the peak rate of the video stream to prevent congesting the client's access channel during patching. The resultant slice will be repeatedly multicast over a dedicated channel at a constant bitrate R_{max} as shown in Fig. 3. Assume that the size of the block is A Mb; then, the slice will be multicast repeatedly once every $t_{da} = A/R_{max}$ seconds.

The second part, Slice B shown in Fig. 2, has two parts. The first part, covering the first T_A seconds of the video, contains the *remaining* video data that exceed the bitrate ($R_{max} - R_{cut}$). The second part, extending from playback point T_A to the end of the video, contains *all* the video data that exceed the bitrate ($R_{max} - R_{cut}$). This slice will be multicast repeatedly over a separate multicast channel following the actual video data rate (as opposed to the constant transmission rate for Slice A).

Lastly, the third part, Slice C shown in Fig. 2, contains the rest of the video data. This slice will be multicast repeatedly over a separate multicast channel following the actual video data rate, which will not exceed ($R_{max} - R_{cut}$).

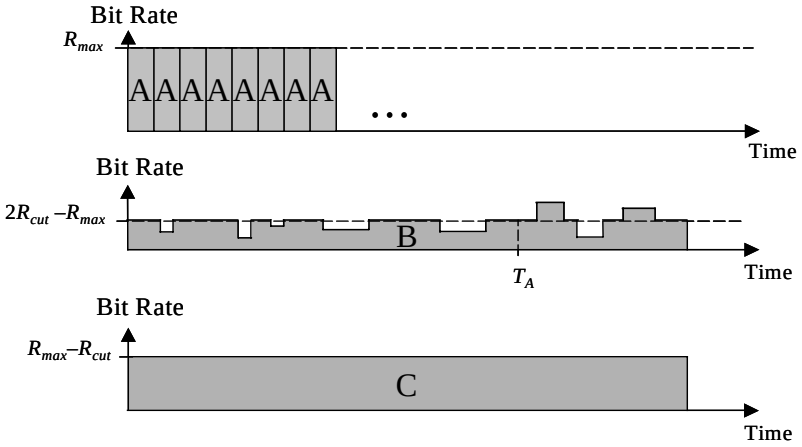


Fig. 3. The three types of multicast channels in the Slice-and-Patch algorithm.

5.2 Bandwidth Allocation

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

There are two types of static multicast channels, one type for transmitting Slice B and the other for 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 S&P does not occupy a fixed bandwidth. Therefore, offline numerical procedures are needed to compute the maximum number of static multicast channels that can fit within the bandwidth $(B_{max} - R_{max})/2$. The remaining bandwidth will be used by the dynamic channels.

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 t_{da} seconds (see Fig. 4). Next, the client requests 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. Note that the length of the blocks $\{B_1, C_1\}$ is equal to $t_1 - t_m$ seconds, and that this is also the duration of Phase 2.

At the beginning of Phase 3, the client has cached block C_2 and completed playback of blocks $\{B_1, C_1\}$. As Fig. 4 shows, to continue playback, the client needs block B_2. This is supplied by the dynamic channel at a bitrate of $R_{max} - R_{cut}$, as the static channel transmitting blocks $\{B_3, C_3\}$ occupies the remaining bandwidth of R_{cut}.

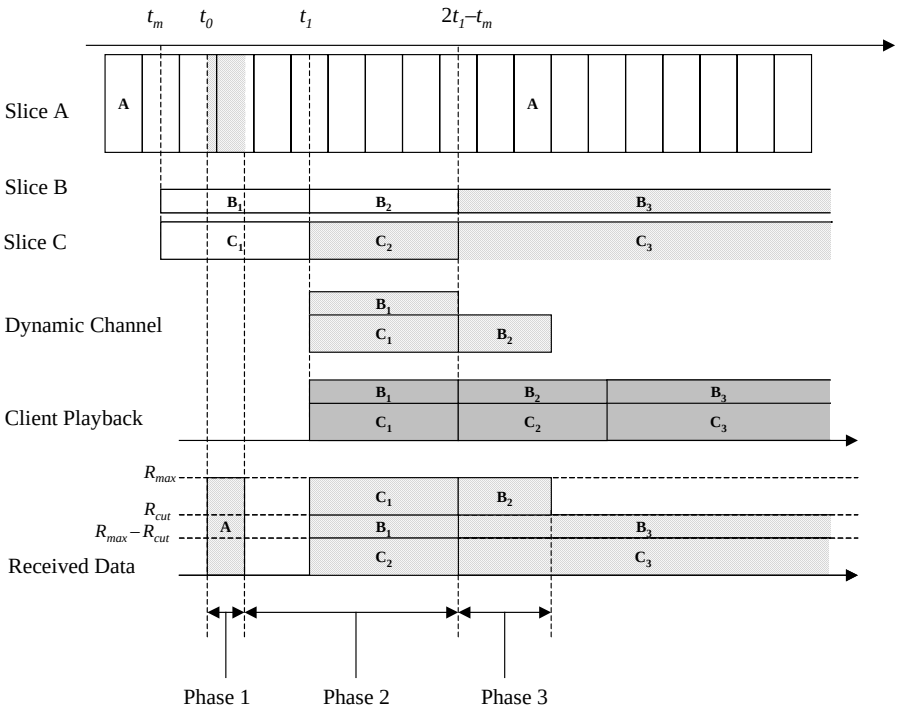


Fig. 4. The three-phase patching process in the Slice-and-Patch algorithm.

Since the size of block B_2 is equal to $(2R_{cut} - R_{max})(t_1 - t_m)$, the time it takes to transmit this block at a rate of $R_{max} - R_{cut}$ is equal to $(2R_{cut} - R_{max})(t_1 - t_m)/(R_{max} - R_{cut})$. To derive the duration of Phase 3, we note that $(t_1 - t_m)$ must be smaller than T_R and thus the duration will be bounded by $(2R_{cut} - R_{max})T_R/(R_{max} - R_{cut})$. So, the maximum duration of all the three phases will be bounded by $(2R_{cut} - R_{max})T_R/(R_{max} - R_{cut}) + T_R$. Since the first phase starts at $(T_r - t_m)$ the latest, Phase 3 will end no later than $(2R_{cut} - R_{max})T_R/(R_{max} - R_{cut}) + T_R + T_R$ which equals to T_A in (5) after simplification. After Phase 3 is completed, the client releases the dynamic channel and continues playback using data received from the static channel for the rest of the video session.

6. PERFORMANCE EVALUATION

In this section, we evaluate the three algorithms presented in section 4 and section 5 based on the results of simulations. The simulator was developed using the CNCL simulation library [19], and the simulations were conducted using 50 VBR video bitrate traces from DVD videos. To facilitate comparison, we scaled the video bitrate traces so that all videos had the same average bitrate of 3Mbps. The server was configured with a total bandwidth of 150Mbps, and the client had an access bandwidth of 6Mbps. Each simulation run simulated a duration of 30 days, with the first day of data skipped to reduce the initial condition effects. The client arrival rate was 1 request per second.

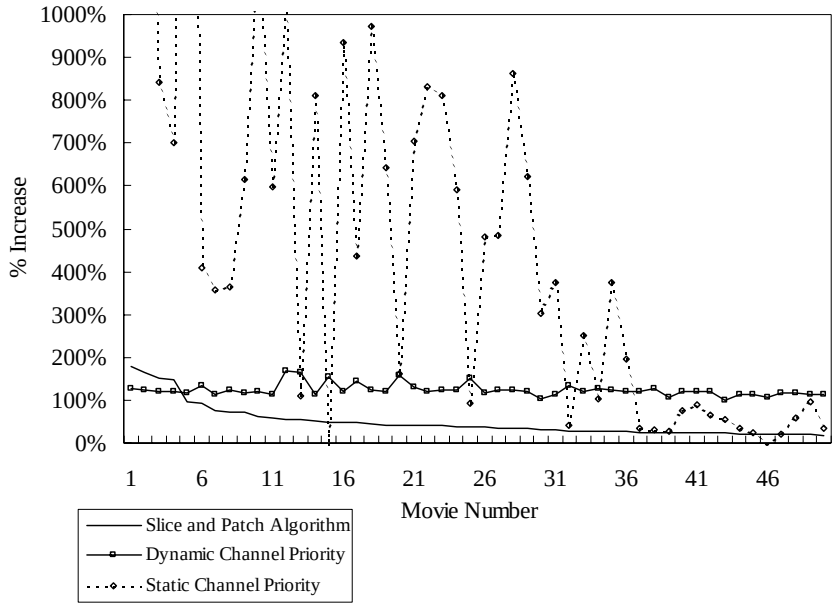


Fig. 5. Percentage of latency increases over the CBR-based system for 50 different videos.

For the S&P algorithm, we set the parameter R_{cut} to be equal to $1.1R_t$, where R_t is the average bitrate of the first $2T_R$ seconds of the video. Clearly this may not be optimal, and we are investigating an efficient way to find the optimal R_{cut} value without conducting a huge number of simulation runs.

Fig. 5 plots the simulation results for the three algorithms for 50 different videos. The vertical axis is the percentage increase in latency compared with the same system serving CBR video of the same length and average bitrate (i.e., 3Mbps). Thus, this shows the cost of supporting VBR-encoded video instead of CBR-encoded video although VBR-encoded video will have better visual quality [12].

Several observations can be made based on the simulation results. First, in terms of the average latency increase computed based on all 50 videos, S&P performs best at 50%, Dynamic Channel Priority is second best at 120%, and Static Channel Priority is worst at 550%. Second, in terms of variation in the latency increase, Dynamic Channel Priority is best with a consistent latency increase across all 50 videos (the standard deviation is only 14%). S&P has more variation at a standard deviation of 40%. Static Channel Priority is the worst one with a huge standard deviation of 865% and a maximum latency increase of over 2,000%. The higher variation for Static Channel Priority is due to variation in the bitrate of the video's initial portion. In particular, the algorithm allocates more bandwidth to cache static channel video data that cannot be immediately played back. Therefore, if the initial video portion has a high bitrate, then the dynamic channel will take more time to cache sufficient video data to begin playback. In contrast, the Dynamic Channel Priority and S&P algorithms are less sensitive to this effect because both algorithms allocate more bandwidth to caching video data that can be played back immediately.

Comparing the three algorithms, the S&P algorithm clearly performed best except for a few videos. Although on average, latency still increased by 50%, this did not account for the improved visual quality due to the use of VBR encoding techniques. Given that VBR-encoded video can achieve visual quality comparable to that of CBR-encoded video at twice the video bitrate, this S&P algorithm can potentially achieve a level of performance comparable to that of CBR-based systems with the proper choices of VBR encoding parameters.

The simulation results were obtained by simulating each video individually. In a real system with multiple videos, one can further improve system efficiency by allocating channels according to the video's popularity. Further investigations are needed to quantify the potential performance gains and also tradeoffs in applying non-uniform channel allocation policies.

7. CONCLUSIONS

This paper has investigated a multicast VoD system supporting streaming of VBR-encoded video. Unlike unicast-based VoD systems, existing algorithms, such as temporal smoothing, do not cater for the characteristics of multicast VoD systems, such as the use of periodic multicast and the limitation of the client access bandwidth. Therefore, new streaming algorithms are needed, and three such algorithms, namely, Static Channel Priority, Dynamic Channel Priority, and Slice-and-Patch, have been studied in this paper. In simulations, we found that the Slice-and-Patch algorithm generally outperformed the other two except for a few videos. We suspect that the exceptions were due to non-optimal configuration of the R_{cut} parameter in S&P. We are currently running additional simulations to further investigate this issue. Nevertheless, with a modest 50% latency increase over the case with CBR-encoded video, this S&P algorithm has very good potential to achieve a level of performance comparable to that of CBR systems with similar resources.

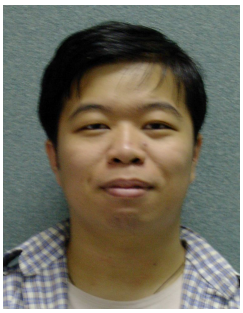
ACKNOWLEDGEMENTS

This research was funded by a Direct Grant, an Earmarked Grant (CUHK 4328/ 02E) from the HKSAR Research Grant Council, and AoE-IT, a research grant from the HKSAR University Grants Council.

REFERENCES

1. A. Dan, D. Sitaram, and P. Shahabuddin, "Scheduling policies for an on-demand video server with batching," in *Proceedings of 2nd ACM Multimedia*, 1994, pp. 15-23.
2. H. Shachnai and P. S. Yu, "Exploring waiting tolerance in effective batching for video-on-demand scheduling," in *Proceedings of 8th Israeli Conference on Computer Systems and Software Engineering*, 1997, pp. 67-76.
3. 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, 1996, pp. 1099-1109.
4. W. Liao and V. O. K. Li, "The split and merge protocol for interactive video-on-demand," *IEEE Multimedia*, Vol. 4, 1997, pp. 51-62.
5. K. A. Hua, Y. Cai, and S. Sheu, "Patching: A multicast technique for true video-on-demand services," in *Proceedings of 6th International Conference on Multimedia*, 1998, pp. 191-200.
6. Y. Cai, K. Hua, and K. Vu, "Optimizing patching performance," in *Proceedings of SPIE/ACM Conference on Multimedia Computing and Networking*, 1999, pp. 204-215.
7. 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," in *Proceedings of 6th International Conference on Computer Communications and Networks*, 1997, pp. 200-207.
8. D. Saparilla, K. W. Ross, and M. Reisslein, "Periodic broadcasting with VBR-encoded video" in *Proceedings of IEEE Infocom 1999*, 1999, pp. 464-471.
9. 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.
10. T. C. Chiueh and C. H. Lu, "A periodic broadcasting approach to video-on-demand service," in *Proceedings of SPIE*, 1996, pp. 162-169.
11. A. Hu, I. Nikolaidis, and P. van Beek, "On the design of efficient video-on-demand broadcast schedules," in *Proceedings of 7th International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems*, 1999, pp. 262-269.
12. W. S. Tan, N. Duong, and J. Princen, "A comparison study of variable-bit-rate versus fixed-bit-rate video transmission," in *Proceedings of Australian Broadband Switching and Services Symposium*, 1991, pp. 134-141.
13. J. Y. B. Lee and C. H. Lee, "Design, performance analysis, and implementation of a super-scalar video-on-demand system," to appear in *IEEE Transactions on Circuits and Systems for Video Technology*.
14. W. Feng and S. Sechrest, "Smoothing and buffering for the delivery of pre-recorded video," in *Proceedings of ISET/SPIE Multimedia Computing and Networking*, 1995, pp. 234-244.
15. J. D. Salehi, Z. L. Zhang, J. F. Kuros, and D. Towsley, "Supporting stored video: reducing rate variability and end-to-end resource requirements through optimal smoothing," in *Proceedings of ACM SIGMETERICS*, 1996, pp. 222-231.
16. W. Feng, F. Jahanian, and S. Sechrest, "Optimal buffering for the delivery of compressed pre-recorded video," in *Proceedings of the IASTED/ISMM International Conference on Networks*, 1995.
17. W. Feng, Mishra, and Ramakishnan, "A comparison of bandwidth smoothing techniques for the transmission of pre-recorded compressed video," in *Proceedings of INFOCOM '97*, Vol. 1, 1997, pp. 58-66.
18. D. Y. Lee and H. Y. Yeom, "Tip prefetching: Dealing with the bit rate variability of video streams," in *Proceedings of the IEEE International Conference on Multimedia Computing and Systems 1999*, Vol. II, 1999, pp. 352-356.
19. ComNets Class Library and Tools: http://www.comnets.rwth-aachen.de/doc/cncl.html



Chun-Wai Kong received the B.Eng. degree in Information Engineering from the Chinese University of Hong Kong in 2000. He is currently a M.Phil candidate in the Department of Information Engineering at the Chinese University of Hong Kong. His research interest is multicast video streaming.

Jack Y. B. Lee is with the Department of Information Engineering at the Chinese University of Hong Kong. He directs the Multimedia Communications Laboratory (http://www.mcl.ie.cuhk.edu.hk) to conduct research in distributed multimedia systems, fault-tolerant systems, multicast communications, and Internet computing. He can be reached at jacklee@computer.org.