RECURSIVE PATCHING
An Efficient Technique for Multicast Video Streaming

Y. W. Wong, Jack Y. B. Lee
Department of Information Engineering
The Chinese University of Hong Kong, Shatin, N.T., Hong Kong
Email: ywwong1@ie.cuhk.edu.hk, yblee@ie.cuhk.edu.hk

Keywords: Video-on-demand, multicast, patching, transition, recursive.

Abstract: Patching and transition patching are two techniques proposed to build efficient video-on-demand (VoD) systems. Patching works by allowing a client to playback video data from a patching stream while caching video data from another multicast video stream for later playback. The patching stream can be released once video playback reaches the point where the cached data begins, and playback continues via the cache and the shared multicast channel for the rest of the session. Transition patching takes this patching technique one step further by allowing a new client to cache video data not only from a full-length multicast channel, but also from a nearby in-progress patching channel as well to further reduce resource consumption. This study further generalizes these patching techniques into a recursive patching scheme where a new client can cache video data recursively from multiple patching streams to further reduce resource consumption. This recursive patching scheme unifies the existing patching schemes as special cases. Simulation results show that it can achieve significant reductions (e.g. 60%~80%) in startup latency at the same load and with the same system resources.

1 INTRODUCTION

Although extensive researches on video-on-demand (VoD) have been conducted in the last decade, commercial deployment of VoD services in the market is still far from commonplace. Apart from content copyright issues, the main reason is the immense network and server resources needed to serve a large user population. As traditional VoD systems make use of unicast for video streaming, the required server and network bandwidth grows linearly with the number of subscribers, thereby rendering metropolitan-scale deployment economically difficult, if not impossible.

To tackle this challenge, researchers have recently investigated a number of innovative video streaming architectures based on network multicast to dramatically reduce the server and network resources needed in large-scale VoD systems.

One technique, called *batching*, groups users waiting for the same video data and then serves them using a single multicast channel (Dan *et al.*, 1994a; Dan *et al*., 1994b; Dan *et al*., 1996; Aggarwal *et al*., 1996; Shachnai and Yu, 1997). This batching process can occur passively while the users are waiting, or actively by delaying the service of earlier users to wait for later users to join the batch. Various batching policies have been proposed in recent years, including first-come-first-serve (FCFS) and maximum queue length (MQL) proposed by Dan *et al*. (1994a), maximum factored queue (MFQ) proposed by Aggarwal *et al*. (1996), Max_Batch and Min_Idle proposed by Shachnai and Yu (1997), etc.

Another technique, called *patching*, exploits client-side bandwidth and buffer space to merge users on separate transmission channels onto an existing multicast channel (Liao and Li, 1997; Hua *et al*., 1998; Cai *et al*., 1999; Carter *et al*., 1997; Eager *et al*., 2000). The idea is to let a client cache data from a nearby (in playback time) multicast transmission channel while sustaining playback with data from another transmission channel – called a patching channel in (Hua *et al.*, 1998). This patching channel can be released once video playback reaches the point where the cached data begins, and playback continues via the cache and the shared multicast channel for the rest of the session.

Cai and Hua (1999) took this patching technique one step further by allowing a new client to cache video data not only from a full-length multicast channel, but also from a nearby in-progress patching channel as well. This technique, called *transition patching*, can further reduce resource consumption when compared to simple patching.

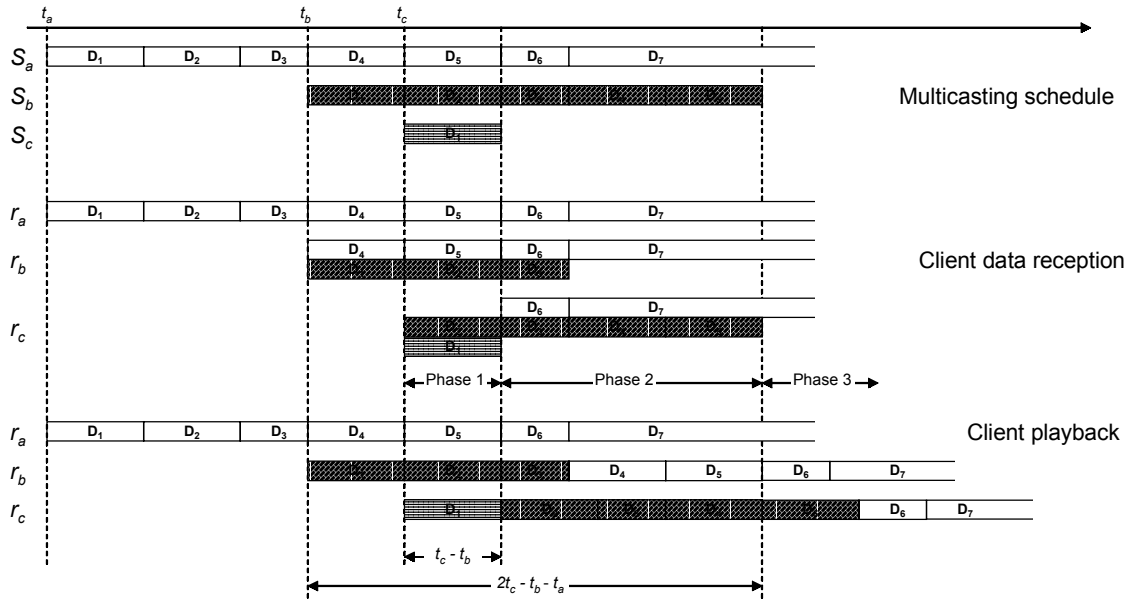


Figure 1: Streaming, data reception and playback schedules in transition patching.

In this study, our first contribution is the generalization of these patching techniques into a recursive patching scheme where a new client can cache video data recursively from multiple patching streams to further reduce resource consumption. Second, this recursive patching scheme also unifies the existing patching schemes as special cases. Third, we also consider recursive patching in bandwidth-limited systems and incorporate batching to further reduce resource consumption, especially in heavily-loaded systems. Our simulation results show that recursive patching can achieve startup latency reduction of 60%~80% compared to transition patching.

The rest of this paper is organized as follows: Section 2 reviews transition patching; Section 3 introduces our recursive patching scheme; Section 4 addresses the stream assignment problem; Section 5 compares the performance of different patching schemes using simulation; and Section 6 summarizes the study.

2 TRANSITION PATCHING

Fig. 1 illustrates the patching and transition patching techniques. There are three clients, denoted by r_a, r_b and r_c, which arrive at the system at time instants t_a, t_b, and t_c respectively requesting the same video. We assume that the length of the video being served is L and is encoded at a constant bit-rate of R bps. To facilitate discussion, we divide the video into 7 logical segments (D_1 to D_7) and denote the group of video segments from the r^{th} segment to the s^{th} segment by $[D_r, D_s]$.

Assuming the system is idle when r_a arrives, then the system will assign a *regular stream* (R-stream), denoted by S_a, to stream the whole video from the beginning to the end (i.e. $[D_1, D_7]$) to client r_a. The cost of serving client r_a is thus equal to the bandwidth-duration product LR. For client r_b arriving at time t_b, it clearly cannot begin playback by receiving video data from stream S_a as it has missed the first (t_b-t_a) seconds of the video, i.e., $[D_1, D_3]$.

Instead of starting another R-stream for client r_b, the system assigns a *patching stream* (P-stream) S_b to transmit only the first (t_b-t_a) seconds (i.e., $[D_1, D_3]$) of missed video data to enable client r_b to begin playback. At the same time, client r_b also caches video segment $[D_4, D_6]$ from multicast stream S_a for later playback. After (t_b-t_a) seconds, the video playback of client r_b will reach video time (t_b-t_a) and thus the client can continue playback using the cached data received from S_a for the rest of the video. The P-stream S_b can then be released for reuse by other clients. Given that S_b is occupied for duration much shorter than the length of the video ((t_b-t_a) versus L), the cost of serving client r_b is thus significantly reduced.

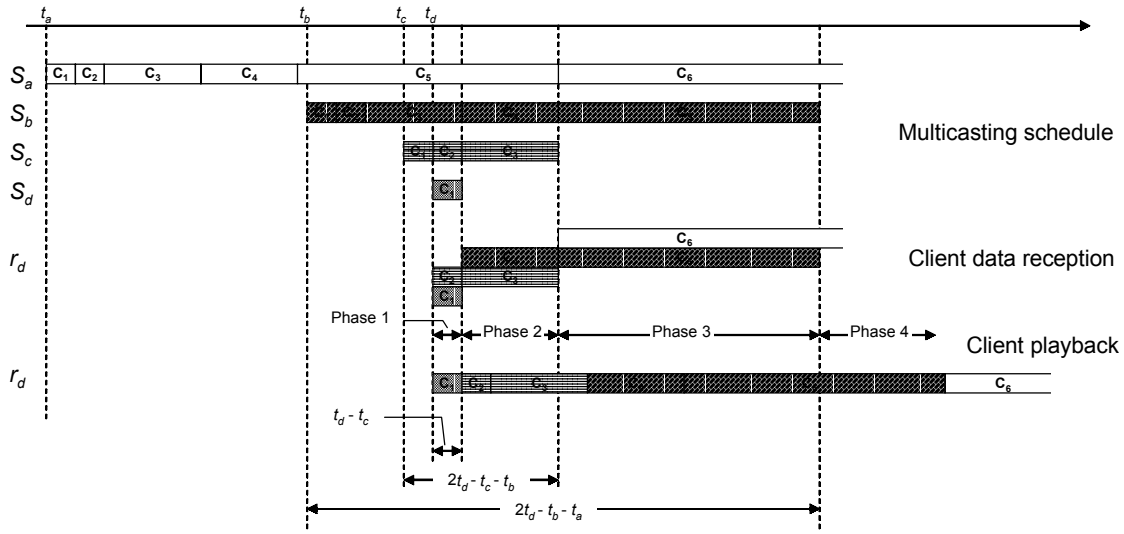


Figure 2: Illustration of recursive patching.

This technique is called *patching* in this study and it has been studied by a number of researchers (Liao and Li, 1997; Hua *et al*., 1998; Cai *et al*., 1999; Carter *et al*., 1997; Eager *et al*., 2000) under different names. The tradeoffs of patching are the need for network multicast, the need for the client to receive two video streams simultaneously, and the additional client buffer required (i.e. $(t_b-t_a)R$ bytes) to cache video data received from the R-stream. Nevertheless, subject to the client's buffer availability, a client admitted using patching always consumes less server resource than a client admitted using an R-stream as is the case in a conventional VoD system.

In patching, a patched client receives data from a P-stream while caching video data from another R-stream. Cai and Hua (1999) proposed a *transition patching* technique which extends patching to allow a client to cache video data not only from an R-stream, but also from another P-stream as well. In other words, a client admitted using transition patching will first receive video data from two P-streams, then releases one P-stream to be replaced by an R-stream, and finally releases the remaining P-stream to continue playback for the rest of the video using data received from the R-stream.

For example, consider client r_c in Fig. 1 which arrives at time t_c and is admitted using transition patching. We note that for client r_c, it has already missed video segment [D_1, D_4] multicast from the R-stream S_a. To patch this missed video segment, transition patching employs a three-phase admission process as shown in Fig. 1. In Phase 1, a P-stream S_c is allocated to stream the initial video segment D_1 to client r_c to begin playback. At the same time, the client caches video data segment D_2 being multicasted by the P-stream S_b. In Phase 2, the P-stream S_c is released and the client begins caching video data segment [D_6, D_7] from the R-stream S_a. Note that the client also continues to receive video segment [D_3, D_5] from the P-stream S_b. Finally, in Phase 3 the remaining P-stream S_b is released and the client simply continues playback using cached data and data received from the R-stream S_a.

This transition patching technique differs from simple patching in two aspects. First, the P-stream S_c allocated for client r_c is occupied for a duration of (t_c-t_b) seconds, which is shorter than the duration when simple patching is used, i.e., (t_c-t_a) seconds. Second, the P-stream S_b is extended from (t_b-t_a) seconds to $(2t_c-t_a-t_b)$ seconds to support client r_c. This stream is called a *transition stream* (T-stream) in (Cai and Hua, 1999). Thus the net gain in resource reduction is equal to $\{[(t_c-t_a)-(t_c-t_b)] - [(2t_c-t_a-t_b)-(t_b-t_a)]\} = 3t_b-2t_c-t_a$.

For example, suppose L, t_a, t_b and t_c equal to 7200, 0, 200 and 250 seconds respectively. Then the costs of supporting r_a, r_b and r_c are 7200R, 200R and 150R respectively, representing resource savings of 97.22% and 97.92% for clients r_b and r_c.

3 RECURSIVE PATCHING

The fundamental principle of patching is to cache video data from the nearest stream so as to minimize the amount of video data missed. We observe that this principle not only can be applied to a single patching stream, but also to a series of patching streams as well. This motivates us to devise a new

recursive patching technique where video data from multiple levels of patching streams are cached to further reduce the resources required.

Fig. 2 illustrates the recursive patching technique by considering a fourth client r_d which arrives at the system at time t_d, in addition to the three clients r_a, r_b, and r_c considered in Fig. 1. To ease discussion, we divide the whole video into 6 different segments C_1 to C_6.

For the client r_d, it has missed the initial $(t_d\text{-}t_a)$ seconds of the video. Thus, if we use simple patching it will incur a cost of $(t_d\text{-}t_a)R$ bytes. Using transition patching with S_b as the transition stream will incur a cost of $(3t_d\text{-}2t_c\text{-}t_b)R$ bytes.

Now consider the use of recursive patching, which in this case is divided into 4 phases as shown in Fig. 2. In Phase 1, the client caches video segment C_2 from S_c while playing back video segment C_1 using data received from S_d. In Phase 2, client r_d continues to receive video segment C_3 from S_c but releases S_d and replaces it with S_b to receive video segment C_4. In Phase 3, client r_d continues to receive video segment C_5 from S_b but releases S_c and replaces it with S_a to receive video segment C_6. Finally, in Phase 4 the client releases the remaining P-stream S_b and contines playback till the end of the video by receiving video data from S_a.

Subtracting the lengths of different streams, the total patching cost to serve r_d is $[(5t_d\text{-}2t_c\text{-}2t_b\text{-}t_a) - (3t_c\text{-}2t_b\text{-}t_a)]R = 5(t_d\text{-}t_c)R$. Compared to the cost of $(3t_d\text{-}2t_c\text{-}t_b)R$ bytes in transition patching, there is a gain of $(3t_c\text{-}2t_d\text{-}t_b)R$ bytes. If t_d equals to 260 seconds, the cost of serving r_d is $50R$, or 99.31% resource saving over serving with a new regular stream.

Note that for the example in Fig. 2, the client caches from at most two streams at any time and so the client bandwidth requirement is the same as simple patching and transition patching. In the whole patching process, the client caches video data through a total of three P-streams and one R-stream. In general, a client can cache video data through even more P-streams as long as there are eligible P-streams.

Let k be the total number of streams (i.e., one R-stream plus k-1 P-streams) which a client caches data from. Then we call this process *k-phase recursive patching* (kP-RP). It is worth noting that simple patching and transition patching are equivalent to 2P-RP and 3P-RP respectively under this unified framework.

4 STREAM ASSIGNMENT

In admitting a new client through k-phase recursive patching, there could be more than (k-1) P-streams eligible for patching. In this case, the system needs to determine the subset of eligible streams to use to increase resource reduction. We call this the *stream assignment problem*. In the next section, we first review the stream assignment scheme employed in the transition patching study (Cai and Hua, 1999) and then extend it to recursive patching in Section 4.2.

4.1 The Equal-Split Stream Assignment Scheme

Fig. 3a depicts the equal-split stream assignment scheme proposed in the transition patching study (Cai and Hua, 1999). There are two parameters in this scheme, namely the regular window length denoted by ω_r and the patching window length denoted by ω_t.

The first stream allocated in a regular window is always an R-stream, streaming the video from the beginning to the end. The first stream in a patching window (except the first one), however, will be a T-stream, which is a P-stream extended to support other clients' transition patching. The rest are simple patching streams (P-streams) streaming the missed initial video segment for the clients.

For example, when the first client r_1 arrives at the system, an R-stream S_1 is assigned, and all the subsequent requests (r_2 through r_{k-1}) are assigned P-streams to perform simple patching. When a client r_k arriving more than ω_t time units after the R-stream S_1, the server will assign a T-stream S_k to it. Within the next ω_t window (r_{k+1} through r_{q-1}), all requests are then assigned P-streams to perform transition patching via the T-stream S_k. The next request r_q is again assigned a new T-stream which also starts another ω_t window and so on. This process repeats until a request r_p whose arrival time exceeds ω_r seconds from the R-stream S_1. In this case a new R-stream will be assigned to begin a new regular window.

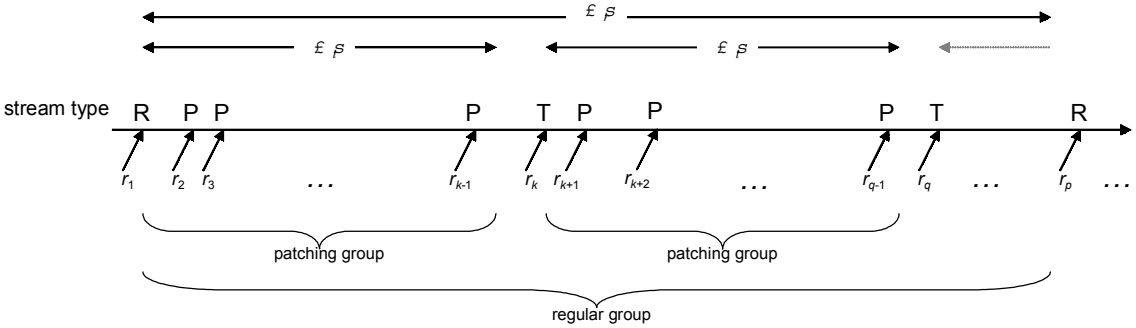
The study by Cai and Hua did not address how to configure ω_r and ω_t to optimize performance. In our experiments, we simply optimize ω_r and ω_t using exhaustive search in unit of seconds. As this computation can be done offline, it does not affect the system's runtime performance.

4.2 A Hierarchical Equal-Split Stream Assignment Scheme

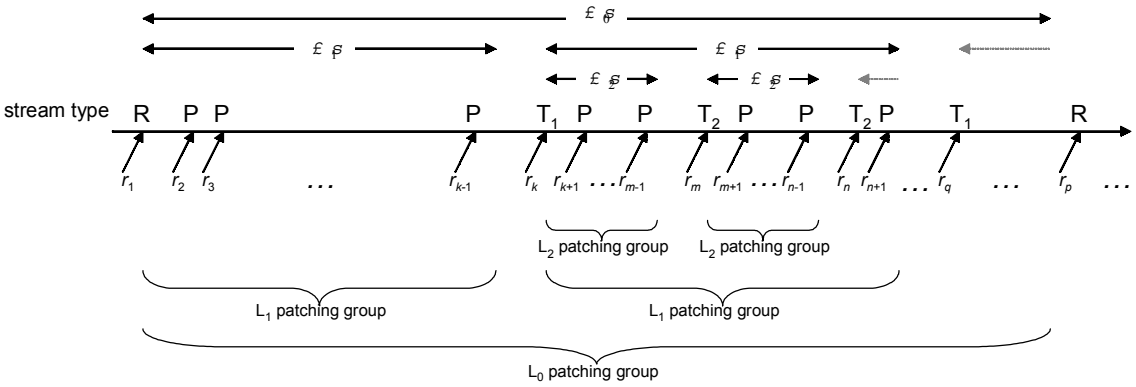
Unlike transition patching, a client admitted with k-phase recursive patching may cache data through a new P-stream plus up to k-2 extended transition streams, say streams s_1, s_2, …, s_{k-2}, before merging back to an R-stream. To be consistent with Cai and Hua's study, we denote these transition streams by T_1-stream, T_2-stream, …, T_{k-2}-stream.

T_{k-2}-stream. In Phase 2, it will release the P-stream and begin caching data from the first stream, in the L_{k-3} patching group, which is a T_{k-3}-stream. In general, the client caches video data from the T_{k-i}-stream and the T_{k-i-1}-stream during Phase i for $i<(k-1)$. In Phase (k-1), the client caches video data from a T_1-stream and the R-stream, and finally releases the T_1-stream to continue playback using the R-stream in the last Phase k.

Again, we can optimize the set of window lengths $\{\omega_0, \omega_1, \ldots, \omega_{k-2}\}$ offline using exhaustive



(a) The equal-split stream assignment scheme.



(b) The hierarchical equal-split stream assignment scheme.

Figure 3: Illustration of stream assignment schemes.

Similarly we can generalize the two-level equal-split stream assignment scheme to a hierarchical equal-split stream assignment scheme with (k-1) levels as shown in Fig. 3b for k=4. Let ω_i be the window length for level i. Then ω_0 and ω_1 are equivalent to the regular window length ω_r and patching window length ω_t in the original equal-split stream assignment scheme.

Streams within the same level i window belong to an L_i patching group. For a client admitted via k-phase recursive patching, it will begin with a new P-stream and at the same time caches video data from the first stream in the L_{k-2} patching group, which is a

search. Note that the computational complexity increases exponentially with k. Further researches are therefore needed to devise computationally-efficient algorithms for optimizing the window lengths.

5 PERFORMANCE EVALUATION

In this section, we evaluate the performance of recursive patching using simulations. We assume Poisson client arrivals and all clients playback the video from the beginning to the end without

interactive operations. For simplicity, we ignore network delays and processing delays. Table 1 lists the system parameters used in our simulation study. We use startup latency - defined as the time from client arrival to the time playback can begin, as the performance metric for comparison.

Table 1: Parameters used in simulations.

Parameter	Range of values
Request arrival rate (/s)	0.1 - 1.0
Movie length (L)	7200
Number of server channels (C)	20

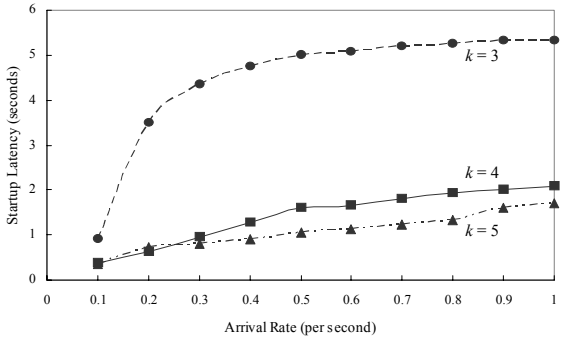


Figure 4: Performance of kP-RP.

Fig. 4 plots the startup latency versus arrival rate ranging from 0.1 to 1 client/second. There are three curves plotting the startup delay for 3-phase, 4-phase, and 5-phase recursive patching respectively. Note that 3-phase recursive patching is equivalent to transition patching.

Compared to transition patching (i.e. with k=3), the 4-phase recursive patching can achieve significantly lower startup latency under the same load. For example, the latency is reduced by 78%, 67% and 62% at arrival rates of 0.3/s, 0.6/s and 0.9/s respectively. The improvement is particularly significant at higher arrival rates. This can be explained by the observation that at higher arrival rates, the streams are more closely spaced and thus, enables more data sharing by patching recursively.

The latency is further reduced when 5-phase recursive patching is employed although the reduction is less significant. Compared to transition patching, 5P-RP can achieve latency reductions of 81%, 78% and 70% at arrival rates of 0.3/s, 0.6/s and 0.9/s respectively. We were not able to generate results for larger values of k due to the extensive computation time needed for optimizing the window lengths $\{\omega_0, \omega_1, \ldots, \omega_{k-2}\}$ (c.f. Section 4.2). Nevertheless the current results do suggest that the improvements will be less and less when k is increased further.

6 CONCLUSION

We investigated in this study a generalized recursive patching scheme (kP-RP) for building efficient, large-scale video-on-demand systems. This new scheme unified the existing patching and transition patching schemes as special cases of 2P-RP and 3P-RP respectively. By using larger values of k (i.e., k>3), we showed that the recursive patching scheme can achieve significantly lower startup latency compared to even the already efficient transition patching scheme, at the same load and with the same system resources. Optimization of the patching window lengths in stream assignment, however, turned out to be very computationally expensive. Therefore further research is needed to reduce the computation time needed and also to extend this algorithm to accommodate different system parameter settings like client access bandwidth, client buffer storage limit, etc.

ACKNOWLEDGEMENT

This work was supported in part by the Hong Kong Special Administrative Region Research Grant Council under Grant CUHK4328/02E and by the Area-of-Excellence in Information Technology.

REFERENCES

Aggarwal, C. C., Wolf, J. L. and Yu, P. S., 1996. On optimal batching policies for video-on-demand storage servers. In *Proc. International Conference on Multimedia Systems*, June 1996.

Cai, Y., Hua, K. and Vu, K., 1999. Optimizing patching performance. In *Proc. SPIE/ACM Conference on Multimedia Computing and Networking*, San Jose, CA, January 1999, pp.204-215.

Cai, Y. and Hua, K. A., 1999. An efficient bandwidth-sharing technique for true video on demand systems. In Proc. *7th ACM International Conference on Multimedia, Orlando*, Florida, United States, 1999.

Carter, S.W., Long, D. D. E., Makki, K., Ni, L.M., Singhal, M. and Pissinou, N., 1997. Improving video-on-demand server efficiency through stream tapping. In *Proc. 6th International Conference on Computer*

Communications and Networks, September 1997, pp.200-207.

Dan, A., Sitaram, D. and Shahabuddin, P., 1994. Scheduling policies for an on-demand video server with batching. In *Proc. 2nd ACM International Conference on Multimedia*, pp. 15-23.

Dan, A., Shahabuddin, P., Sitaram, D. and Towsley, D., 1994. Channel allocation under batching and VCR control in movie-on-demand servers. *IBM Research Report RC19588*, Yorktown Heights, NY.

Dan, A., Sitaram, D. and Shahabuddin, P., 1996. Dynamic batching policies for an on-demand video server. *ACM Multimedia Systems*, vol. 4, no. 3, June 1996, pp. 112-121.

Eager, D. L., Vernon, M. K. and Zahorjan, J., 2000. Bandwidth skimming: A technique for cost-effective video-on-demand. In *Proc. IS&T/SPIE Conf. On Multimedia Computing and Networking 2000 (MMCN 2000)*, San Jose, CA, January 2000, pp.206-215.

Hua, K. A., Cai, Y. and Sheu, S., 1998. Patching: A multicast technique for true video-on-demand services. In *Proc. 6th International Conference on Multimedia*, September 1998, pp.191-200.

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

Shachnai, H. and Yu, P. S., 1997. Exploring wait tolerance in effective batching for video-on-demand scheduling. In *Proc. 8th Israeli Conference on Computer Systems and Software Engineering*, June 1997, pp. 67-76.