Hybrid Broadcast for the Video-on-Demand Service

MA Huadong (马华东)[1] and Kang G. Shin[2]

[1] *College of Computer Science and Technology, Beijing University of Posts and Telecommunications Beijing 100876, P.R. China*

[2] *Real-Time Computing Lab, EECS Department, The University of Michigan Ann Arbor, MI 48109-2122, USA*

E-mail: mhd@bupt.edu.cn; kgshin@eecs.umich.edu

Received February 13, 2001; revised November 14, 2001.

Abstract Multicast offers an efficient means of distributing video contents/programs to multiple clients by batching their requests and then having them share a server's video stream. Batching customers' requests is either client-initiated or server-initiated. Most advanced client-initiated video multicasts are implemented by patching. Periodic broadcast, a typical server-initiated approach, can be entirety-based or segment-based. This paper focuses on the performance of the VoD service for popular videos. First, we analyze the limitation of conventional patching when the customer request rate is high. Then, by combining the advantages of each of the two broadcast schemes, we propose a hybrid broadcast scheme for popular videos, which not only lowers the service latency but also improves clients' interactivity by using an active buffering technique. This is shown to be a good compromise for both lowering service latency and improving the VCR-like interactivity.

Keywords quality-of-service, interactivity, scheduling, video-on-demand, broadcast, multicast

1 Introduction

Video-on-demand (VoD) has a wide spectrum of applications, such as home entertainment, digital video libraries, movie-on-demand, distance learning, tele-shopping, news-on-demand, and medical information service. Basically, a VoD service allows remote clients to select and play back any video from a large collection of videos stored at one or more servers in any mode and at any time. The VoD service is usually long-lived ① and real-time, and requires high storage-I/O & network bandwidths and VCR-like interactivity. A VoD system is usually designed with focus on system cost and consumer-perceived quality-of-service (QoS). The key elements of the system cost are video server capacity, storage-I/O and network bandwidth and throughput, and customer premise equipment (CPE). The VoD clients' QoS is related to service latency, interactivity, and playback effects. Usually, there is a trade-off between clients' QoS and system cost.

Interactivity is a fundamental feature of the VoD service. Customers may generally have the following types of interactions: Play/Resume, Stop/Pause/Abort, Fast Forward/Rewind. There are two types of VoD services to support client interactivity: true VoD (TVoD) that provides all control functions, and near VoD (NVoD) that provides only restricted control functions. A conventional TVoD system uses one dedicated channel for each service request, which offers the client the best QoS and TVoD service. However, this service incurs high system cost, especially in terms of storage-I/O and network bandwidths. Moreover, such a VoD service has poor scalability and low performance/cost efficiency.

This paper was partly done when Ma Huadong visited the University of Michigan. The work reported in this paper is supported by the NSF of USA under Grant EIA-9806280, the National Natural Science Foundation of China under Grant No.69873006, and a grant for Excellent Young Teachers by the MOE of China.

① A typical movie-on-demand service lasts 90–120 minutes.

Because different videos are requested at different rates and at different times, researchers assume that the popularity of videos follows the Zipf distribution[1] with a skew factor of 0.271. Videos are usually divided into hot (popular) and cold, and requests for the top 10–20 videos are known to constitute over 60% of the total demand. So, it is crucial to improve the service efficiency of hot videos. This paper considers the problem of multicasting a few popular videos during prime time on the network, while focusing on service latency and user interactivity as clients' QoS.

Multicast allows users to share a server stream by batching their requests in two different ways. One is *client-initiated multicast*; when a server channel becomes available, it selects a batch of requests and multicasts the requested video according to some scheduling policy, such as MQLF (Maximum Queue Length First), FCFS (First Come First Served), and MFQLF (Maximum Factored Queue Length First). The other is *periodic broadcast* (*server-initiated*) which uses a fixed number of multicast channels to periodically broadcast video objects to a group of subscribers. It is efficient in transmitting popular videos from one server to many clients.

In this paper we will focus on the performance of video multicast when the request rate is high but there are only a limited number of channels available. We propose a new hybrid broadcast scheme which not only lowers service latency but also improves the VCR interactivity. We use both entirety-based and segment-based broadcasting to provide the VoD service for popular videos. The leading part of a video is periodically broadcast to clients by using several channels and the skyscraper broadcasting (SB). The rest of the video is multicast to clients at a fixed interval via additional regular channels. Thus, the maximum service latency is limited by the smallest broadcasting segment of the leading part. At the same time, the VCR-like interactivity is improved with the CPE buffer and active buffering. We show that this is a good compromise of two common broadcast schemes for better service latency and VCR interactivity.

The remainder of this paper is organized as follows. Section 2 introduces the conventional multicast and broadcast schemes for the VoD service, and analyzes their support for the VCR interactivity. Section 3 analyzes the performance of patching when the request rate is high. The hybrid broadcast scheme for the VoD service is presented in Section 4, while its performance is evaluated and compared with other related techniques in Section 5. Finally, we conclude the paper with Section 6.

2 Related Work

Existing schemes for allocating VoD service channels can be broadly divided into *user-centered* and *data-centered* schemes[2]. A user-centered scheme dedicates channels to individual users, whereas a data-centered scheme dedicates channels to video objects. The latter scheme relies on multicast and has potential for dramatically reducing network and server bandwidth requirements. The data-centered multicast VoD service can be either *client-initiated* or *server-initiated*[3]. In the client-initiated service, the service is initiated by clients. In the server-initiated service, the server channels are dedicated to individual videos. Popular videos are broadcast periodically under this scheme, and a new request dynamically joins the stream that is being broadcast with a small delay.

2.1 Client-Initiated Service

For the client-initiated multicast service, the same video is batched at equally-spaced intervals[4]. The requests by multiple clients for the same video arriving within a short time are served with a single stream. The batching mechanism has a fixed maximum service latency and supports the NVoD interactivity. However, some clients may renege due to a long wait.

In order to reduce service latency, dynamic multicast techniques, such as Adaptive Piggybacking[5], Patching[6], and Controlled CIWP (Client-Initiated-with-Prefetching)[3], have been proposed. For example, Adaptive Piggybacking[5] allows clients arriving at different times to share a data stream by altering the playback rates of requests in progress (for the same object), for the purpose of merging their respective video streams into a single stream that can serve the entire group of merged requests.

This approach can lower the service latency as compared to simple batching. But it is restrictive in that the variation of the playback rate must be within, say 5%, of the normal playback rate, or it will result in a perceivable deterioration of QoS. This fact limits the number of streams that can be merged.

To eliminate the service latency, patching was introduced in [6]. In the patching scheme, most of the time channels are used to patch the missing portion of a service, rather than having to multicast the video in its entirety. Given that there is an existing multicast of a video, when to schedule another multicast for that same video is a critical factor. The time period after a multicast, during which patching must be used, is referred to as the *patching window*[7]. Two simple approaches are discussed in [6]. The first one uses the length of the video as the patching window. This approach is called *Greedy Patching* because it tries to exploit an existing multicast as much as possible. Over-greed can actually result in less data sharing. The other scheme, called *Grace Patching*, uses a patching stream for the new client only if it has enough buffer space to absorb the skew. Hence, under Grace Patching, the patching window is determined by the client buffer size. Considering such factors as video length, client buffer size, and request rate, the authors of [7] generalized the patching technique by deriving the optimal patching window for each video.

An improved patching technique is called *Transition Patching*[8]. The server's scheduling policy is illustrated in Fig.1. The first request A is serviced by a regular stream. For all requests arriving in the next time W_r (the minimal scheduling interval of two consecutive regular streams), either patching stream or transition stream is scheduled. Patching streams are scheduled for the requests arriving in the next W_t time unit, where W_t is called the *transition window* (no larger than W_r). These consecutive patching streams form a patching group. Then, a transition stream is scheduled for the next request B. For the requests arriving in the next transition window, for instance, request C, will share the transition stream, and only patching streams are scheduled. This pattern is repeated until the time skew of a new request to the regular stream is greater than W_r, in which case a new regular stream will be started.

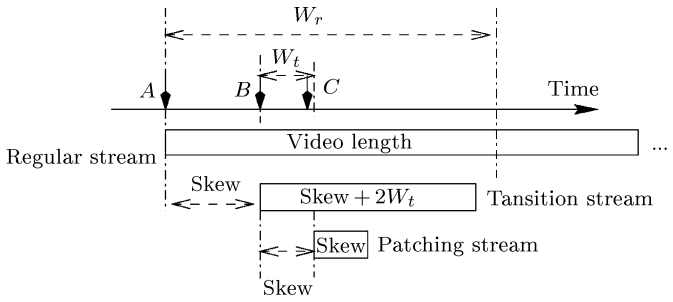

Fig.1. Illustration of transition patching.

At the client site, two data loaders are used to receive data from three streams and one video player is used to fetch data from the local storage and play them back. Transition Patching outperforms Grace Patching and makes more improvement when the request rate is high.

2.2 Server-Initiated Broadcasting

For a highly popular video, periodic broadcast can be used to improve the system performance. The simplest server-initiated scheme, called *entirety-based broadcasting*, broadcasts an entire video at a fixed time interval using dedicated channels[4]. This is the same as the client-initiated multicast scheduled at a fixed interval. The other periodic broadcast techniques called *segment-based broadcast*, include Pyramid Broadcasting[9,10], Permutation-Based Pyramid Broadcasting[11], Skyscraper Broadcasting[12], Greedy Disk-conserving Broadcasting[13], Dynamic Skyscraper Broadcasting[14], and Catching[15]. These schemes divide the server bandwidth into a large number, say K, of logical chan-

nels. To broadcast a video over its K dedicated channels, the video stream is divided into K fragments of increasing size, each of which is repeatedly broadcast on its own channel. To play back a video, a client tunes into the appropriate channel to download the data fragment of the video at its first occurrence. Clients must be able to receive from two channels simultaneously and buffer the fragment received earlier than needed for playback.

Periodic broadcast of the initial smallest fragment is most frequent, allowing new requests to begin quickly. For lower service latency, the size of the first fragments can be made very small to allow them to be broadcast more frequently. As a result, the worst-case service latency experienced by any subscriber is guaranteed to be independent of the current number of pending requests, and such a guarantee can generally reduce the reneging behavior of clients, and therefore improve the server throughput. Based on this idea, the Skyscraper Broadcasting (SB) offers a simple and efficient implementation.

In the SB scheme, K channels are assigned to each of the N most popular objects. Each of the K channels carries a specific segment of the video at the playback rate. The progression of relative segment sizes on the channel, $\{1, 2, 2, 5, 5, 12, 12, 25, 25, 52, 52, \ldots\}$, is bounded by the parameter W, in order to limit the storage capacity required at the client end. To illustrate the SB, let us assume that $K = 8$ and $W = 12$ for a video object. Note that repeated broadcasting of the first unit segment occurs on channel 1, repeated broadcasting of the next two-unit segment occurs on channel 2, and so forth. For given progression and alignment of relative segment sizes, a client starting in any unit-segment broadcast can receive the necessary sequence of segments without jitter, requiring simultaneous reception on at most two channels. Let L be the total size of the video and S be the sum of the relative segment sizes that are broadcast on K channels, then the storage required in the CPE is equal to $(W-1) * L/S$, where L/S is the size of a unit-segment. If T is the total video playback time, the duration of each unit-segment broadcast is T/S. For the above example, if $T = 90$ minutes, a new broadcast begins on channel 1 every 1.76 minutes.

The disadvantage of segment-based broadcasting is its poor interactivity. In general, it can offer only partial VCR-like interactions at the expense of the CPE buffer. If the client has enough storage space to buffer the entire video (e.g., a 90-minute MPEG-1 movie needs 900MB disk space), backward interactions, such as Rewind, Reverse Search, Slow Motion, and Stop/Pause, can be supported. However, buffering an entire video will be difficult due to copyright protection. Even if there is no problem in buffering the whole video at the client end, providing forward interactions, such as Fast Forward (FF) and Fast Search (FS), is quite difficult. The client can try to "catch" the next data segment and begin consuming the movie from the point onwards, but there is no guarantee that the next segment will be available at the moment the FF/FS command is given.

3 Segmented Patching

3.1 Patching for Popular Videos

Under the assumption that the regular multicast channels are scheduled at an equally-spaced interval for the VCR interactivity, we now analyze the performance of patching techniques when the request rate is high but there are only a limited number of patching channels. In such a case, conventional patching will degenerate to segmented patching, meaning that a patching channel is scheduled for a batch of several delayed requests. If the segmented patching is viewed as a different multicast scheduling policy, it works as follows. First, we use the equally-spaced batching to multicast videos so that the VCR interactivity may be supported, and the patching window is not greater than the CPE buffer size. No-delay admission cannot be guaranteed for the prime time and the popular movies with high request rate (λ), by patching only due to the limited number of available channels. Thus, a request often waits until a channel becomes available. During this wait, other requests may join, thus patching a batch of several requests. So, the segmented patching is a deformation of patching when the request rate exceeds the capacity with which patching channels can support no-delay service.

In order to evaluate the mean and maximum waiting times of clients' requests, we divide the patching window into k segments according to the available patching channels, generating $k-1$ patching points $t_1, t_2, \ldots, t_{k-1}$ (see Fig.2). We will show that uniformly-distributed fixed points provide the best patching performance.

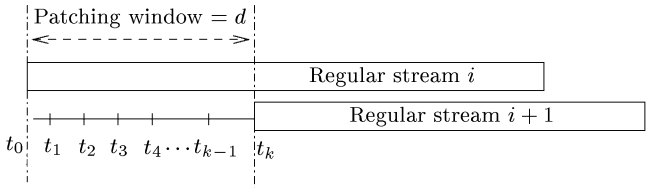

Fig.2. Illustration of segmented patching.

3.2 Performance Analysis

Lemma 1. *For an interval $[0, d]$, if there is any partition, say $t_0 = 0 < t_1 < t_2 < \cdots < t_k = d$, and the length of interval $[t_{i-1}, t_i]$ is d_i, then $\max_{i\in\{1,\ldots,k\}}\{d_i\}$ is minimized if and only if $d_1 = d_2 = \cdots = d_k$.*

The proof is straightforward, and thus omitted. Lemma 1 yields the following result.

Theorem 1. *When the patching window is partitioned into equal intervals, the segmented patching minimizes the worst-case service latency for clients' requests.*

Clearly, the segmented patching for a video needs L/d regular multicast channels, where L is the length of video, d is the size of the patching window. For Grace Patching, we obtain the following results.

Corollary 1. *If the worst-case service latency for clients' requests remains at d/k, the segmented Grace Patching for a video needs $\frac{k-1}{2}$ or more patching channels.*

Proof. Given the patching window is partitioned into k intervals, the patching channel capacity required by the segmented Grace Patching for a video is $\frac{d}{k} + \frac{2d}{k} + \cdots + \frac{(k-1)d}{k} = \frac{(k-1)d}{2}$. □

Assume that requests for the video arrive according to a Poisson process with rate λ. If all of the requests can wait at most d/k time units, we also have the optimal performance on the mean waiting time as stated in the next theorem.

Theorem 2. *When the patching window is partitioned into equally-spaced intervals, the segmented patching minimizes the mean waiting time for clients' requests without reneging.*

Proof. For any division of the patching window with interval sizes $d_1, d_2, \ldots, d_k$, the mean waiting time is

$$\frac{\Sigma_{i=1}^{k}\lambda d_i \times d_i/2}{\lambda d} = \frac{1}{2d}\Sigma_{i=1}^{k} d_i^2$$

So, finding the minimum mean waiting time reduces to

$$\min_{\{d_1,d_2,\ldots,d_k\}} \Sigma_{i=1}^{k} d_i^2$$

subject to $0 \le d_i \le d$ $(i = 1, 2, \ldots, k)$, and $\Sigma_{i=1}^{k} d_i = d$.

The solution to this problem is $d_1 = d_2 = \cdots = d_k = d/k$. □

One can consider possible reneging using the patience model in [16], i.e., a customer agrees to wait τ or more time units with probability $p(\tau, \bar{\tau}) = e^{-\frac{\tau}{\bar{\tau}}}$, where $\bar{\tau}$ is the average time customers agree to wait. In general, the patience rate $\alpha = \frac{1}{\bar{\tau}}$ can be assumed independently of the requested video. We also have the optimal performance on the maximum throughput as follows.

Theorem 3. *When the patching window is partitioned into equal intervals, the segmented patching maximizes the service throughput.*

Proof. According to the analysis in [16], the problem of calculating the number of customers waiting between two consecutive services can be regarded as that of making a transient analysis of an

$M/M/\infty$ "self-service" queuing system with arrival rate λ and self-service with a negative exponential distribution with rate α. So, the mean number of customers in the system at time t is

$$L_{M/M/\infty}(t,i) = \frac{\lambda}{\alpha}(1 - e^{-\alpha t}) + ie^{-\alpha t}.$$

For the segmented patching, the service throughput in the interval $[t_{i-1}, t_i]$ of size d_i is

$$T_i = \frac{\lambda}{\alpha}(1 - e^{-\alpha d_i}).$$

In the patching window, the total throughput is

$$T = \frac{\lambda}{\alpha}(k - \Sigma_{i=1}^{k} e^{-\alpha d_i}).$$

Thus, finding the maximum throughput reduces to

$$\min_{\{d_1,d_2,\ldots,d_k\}} \Sigma_{i=1}^{k} e^{-\alpha d_i}$$

subject to $0 \le d_i \le d$ $(i = 1, 2, \ldots, k)$, and $\Sigma_{i=1}^{k} d_i = d$.

The solution to this problem is $d_1 = d_2 = \cdots = d_k = d/k$. □

From Theorems 1–3, we draw the final conclusion as follows.

Theorem 4. *When the patching window is partitioned into equal intervals, the segmented patching can achieve the optimal service latency and throughput.*

Now, we consider the optimal partition of patching segments when the number of patching channels is fixed. We want to achieve the minimum mean worst-case service latency. This problem for Grace Patching can be solved as in the following theorem.

Theorem 5. *Assume that there are N videos requested with probabilities $p_1, p_2, \ldots, p_N$, respectively. If there are C patching channels available, then the mean worst-case service latency for Grace Patching is minimized when the segment number for video i*

$$s_i = \frac{\sqrt{p_i}(2C + N)}{\Sigma_{i=1}^{N} \sqrt{p_i}}, \quad i = 1, \ldots, N.$$

Proof. The mean worst-case service latency m can be expressed as

$$m = \Sigma_{i=1}^{N} \frac{p_i d}{s_i}.$$

By Corollary 1, $s_i = 2C_i + 1$, where C_i is the number of patching channels required by video i. Thus, finding the minimum worst-case service latency reduces to

$$\min_{\{C_1,C_2,\ldots,C_N\}} \Sigma_{i=1}^{N} \frac{p_i d}{(2C_i + 1)}$$

subject to $0 \le C_i \le C$ $(i = 1, 2, \ldots, N)$, and $\Sigma_{i=1}^{N} C_i = C$.

The solution to this problem is $s_i = 2C_i + 1 = \frac{\sqrt{p_i}(2C+N)}{\Sigma_{i=1}^{N}\sqrt{p_i}}$, $i = 1, \ldots, N$. □

3.3 Discussion

Two segmented patching strategies using the Optimal Segmented Grace Patching (OSGP) are considered as references. According to the analysis in [8], Transition Patching (TP) outperforms Grace Patching (GP) for all scenarios, and makes significant performance gains when the request rate is high. At the same time, the Segmented Transition Patching (STP) outperforms OSGP. For

example, suppose the regular patching window $W_r = d$, the transition window $W_t = d/k$ ($k \geq 4$). Then, the patching channel capacity C_{stp} required by the STP for a video is

$$C_{\text{stp}} = \Big(1+2+3+4+3+\cdots+2\Big(\frac{k}{2}-1\Big)+3\Big)\Big/k = \frac{k^2+4k-8}{4k} \quad (k \text{ is an even number})$$

$$C_{\text{stp}} = \Big(1+2+3+4+3+\cdots+2\Big(\frac{k-1}{2}-1\Big)\Big)\Big/k = \frac{k^2+6k-15}{4k} \quad (k \text{ is an odd number})$$

It is easy to prove that $C_{\text{stp}} \leq \frac{k-1}{2}$ when $k \geq 4$, especially $C_{\text{stp}} < \frac{k-1}{2}$ when $k \geq 6$. STP is more efficient when k is an even number. So, we assume STP with an even number of segments.

Because the segmented patching with a fixed interval is a optimal, it can be the choice of the patching strategy. The advantages of segmented patching include: (1) preservation of scalability, i.e., performance will not degrade when the request rate increases; (2) the "predictable" service latency. The channel assignment can be decided according to clients' patience. A user must wait for some time bounded by d/k under segmented patching, irrespective of whether the request rate is high or low. On the other hand, clients can be served without delay by the conventional GP or TP when the request rate is low. Then, in this case, we need to select a patching strategy according to the relationship between the request rate and available channels. Intuitively, for given C patching channels, if the request rate $\lambda \leq \frac{k}{d}$ or $\frac{2C+1}{d}$, GP can make the service latency zero (whereas OSGP keeps the latency constant) and hence outperforms OSGP. When λ increases, say $\lambda \geq 4\frac{k}{d}$ or $\frac{4(2C+1)}{d}$, our simulation has shown OSGP to outperform GP with respect to service latency. Theoretically, we can think of OSGP as the optimal approximation of GP when the request rate gets high. We can draw a similar conclusion on TP. The detailed simulation results will be presented in Section 5.

4 A Hybrid Broadcast Scheme

4.1 The Basic Model

There are two main ideas behind the Hybrid Broadcast (HB). First, a video is divided into the leading part and the video body. Like entirety-based broadcasting, the video body is broadcast via regular multicast channels at a fixed interval d. Meanwhile, the leading part is broadcast by using SB and additional C channels. Note that the size of the leading part should equal d, or the phase offset of broadcasting the video body. Therefore, clients' requests are served with low latency, and moreover, VCR-like interactions are supported by using the CPE buffer and the regular multicast channels. This is a good compromise of the two broadcast schemes for low service latency and better VCR interactivity.

For example, the leading part (e.g., 5 minutes long) is periodically broadcast to clients by using 3 channels and SB. The video body part (85 minutes of a 90-minute video) is broadcast to clients at a specific interval (5 minutes) by using the additional multicast channels. The progression of relative segment sizes on the periodically broadcasting channels is $\{1,2,2,5,5,12,12,25,25,52,52,\ldots\}$, where the size of the i-th segment is computed by the following recursive formula[12].

$$f(i) = \begin{cases} 1, & \text{if } i = 1 \\ 2, & \text{if } i = 2,3 \\ \Big(2+2\Big\lfloor\frac{i}{2}\Big\rfloor - i\Big) f(i-1) + \Big(1+2\Big\lfloor\frac{i}{2}\Big\rfloor - i\Big)\Big(1+\Big\lfloor\frac{i-4\Big\lfloor\frac{i}{4}\Big\rfloor}{2}\Big\rfloor\Big), & \text{otherwise} \end{cases}$$

This scheme is illustrated in Fig.3. The worst-case service latency is bounded by the phase offset for every client. In this example, the worst-case service latency is 1 min.

For a 90-minute video with (1) $C = 4$, $d = 10$ min., the number of regular channels M =8, the worst-case service latency is 1 min.; (2) $C = 5$, $d = 9$ min., $M = 9$, the worst-case service latency is 0.6 min.; (3) $C = 5$, $d = 15$ min., $M = 5$, the worst-case service latency is 1 min.; and so on.

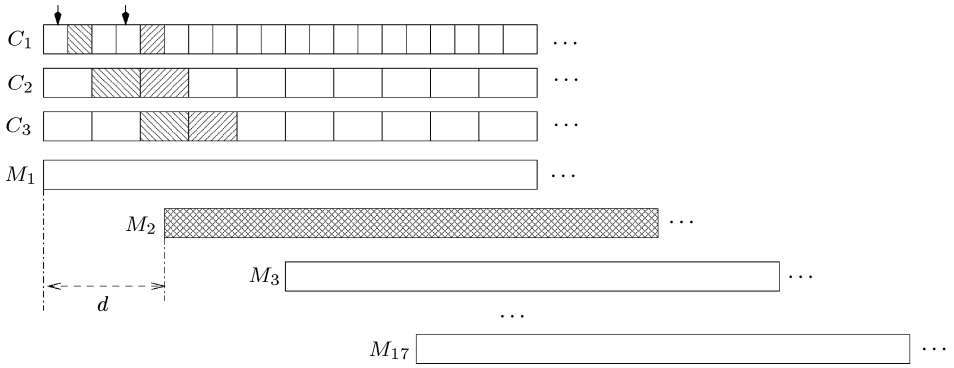

Fig.3. Illustration of the hybrid broadcast scheme ($L = 90$ min., $C = 3$, $M = 17$, $d = 5$ min.).

4.2 Support for the VoD Service

The HB scheme capitalizes on the conventional sever-initiated multicast approaches. Its excellent scalability allows the VoD system to accept as many user requests as possible. We now discuss the HB's support for both request admission and VCR interactivity on the client side.

4.2.1 Limited Service Latency

In the admission phase, the client uses the same method as that of SB[12] to download and play back the video data. HB allows for two simultaneous downloading streams. When the number of available channels is fixed, HB can provide bounded worst-case service latency. Assume that L is the length of each video measured in minutes, M, the number of channels used for regularly broadcasting the video body, then $d = \frac{L}{M+1}$ is the interval of equally-spaced multicast groups. Suppose W is the width of SB. The broadcast video is partitioned into C segments corresponding to the C channels dedicated to it. Then, the Worst-case Service Latency (WSL) for a user request can be calculated as

$$WSL = \frac{d}{\Sigma_{i=1}^{C} \min(f(i), W)} = \frac{L}{(M+1)\Sigma_{i=1}^{C} \min(f(i), W)}.$$

In general, the Mean Service Latency (MSL) is calculated as $MSL = \frac{WSL}{2}$.

4.2.2 VCR Interactivity

A multicast VoD system can offer two kinds of VCR interactivities, depending on whether I-Channels are available or not: *continuous* or *discontinuous* interactions. Continuous interactions allow a customer to fully control the duration of all actions to support the TVoD service, whereas discontinuous interactions execute an action specified only for durations that are integer multiples of a pre-determined time increment to support the NVoD service. The traditional entirety-based broadcast supports the NVoD service, and can offer restricted TVoD with the help of the CPE buffer. HB supports restricted VCR interactivity without incurring any additional bandwidth cost, and achieves better interaction QoS than both entirety-based and periodic broadcasts.

In order to improve interactivity, HB uses the *backup buffer* besides a general buffer (called *play buffer*). There are two backup buffers: *forward buffer* and *backward buffer*. At most d minutes after admission, HB uses only one downloading thread (called *main thread*) to obtain the video data from the nearest regular multicast channel, say M_i. Thus, we can use the other downloading thread (called *backup thread*) to prefetch the video data from next channels and store them in backup buffers for possible interactions. Note that the main thread may only consume the prefetched data without downloading data.

At first, when a client is normally playing back the video, the main thread downloads the video data from its regular multicast channel M_i and stores them in the play buffer, and the backup thread downloads the video data from channel M_{i-1} ② and stores them in the forward buffer. Then, considering the client's interactive behaviors, we illustrate this active buffering scheme in the following two cases.

Case 1. When the client is executing forward interactions, if the resume point is located in the play buffer, both the main and backup threads do not change their downloading channels and buffer use. Otherwise, once the resume point is outside the play buffer and within the forward buffer (as shown in Fig.4), the play buffer is replaced by the forward buffer and the old play buffer acts as the backward buffer, and the old backward buffer acts as the forward buffer where the backup thread stores the video data downloaded from channel M_{i-2}. Note that HB cannot support continuous forward interactions once the resume point exceeds the play and forward buffers.

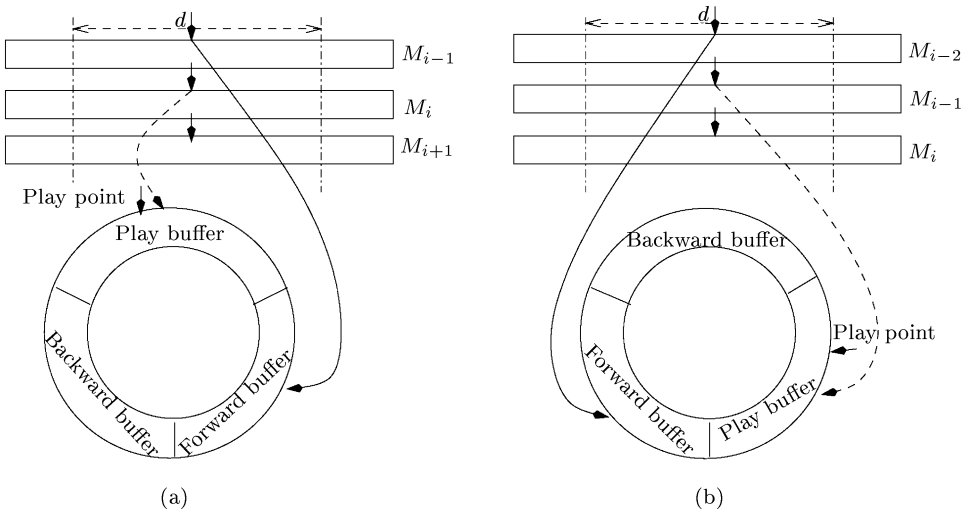

Fig.4. Forward interactions. (a) Before forward interactions. (b) After forward interactions.

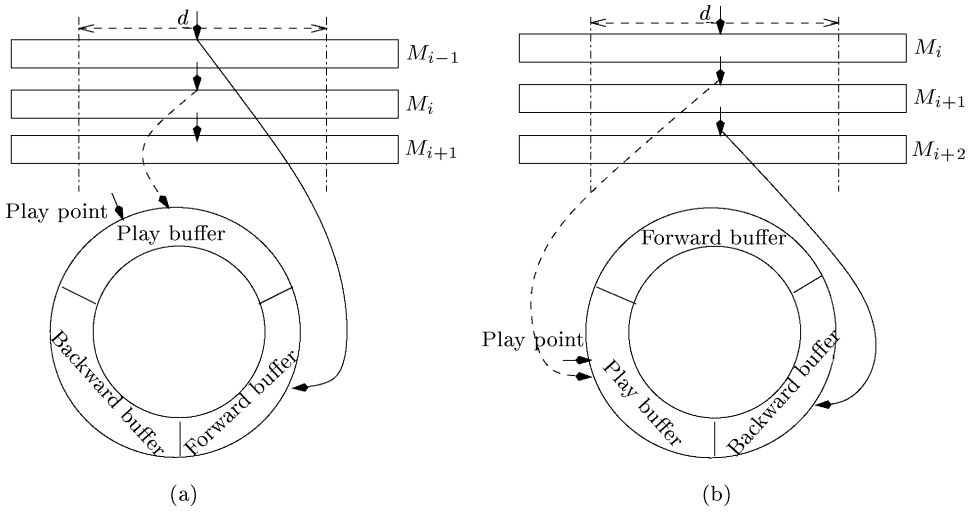

Fig.5. Backward interactions. (a) Before backward interactions. (b) After backward interactions.

②The group index of the multicast stream scheduled later is greater than that scheduled earlier.

Case 2. When the client is executing backward interactions, if the resume point is located in the play buffer, the main and backup threads do not change their downloading channels and buffer use either. If the resume point is outside the play buffer and within the backward buffer (as shown in Fig.5), the play buffer is replaced by the backward buffer and the old play buffer acts as the forward buffer, and the old forward buffer acts as the backward buffer where the backup thread stores the video data downloaded from channel M_{i+2}. Note that HB cannot support continuous backward interactions once the resume point exceeds the play and backward buffers.

Note that the I/O capacity required at the client site is the same as that of SB or Patching. HB works when the CPE buffer size $B \geq d$. The suggested B is $3d$. If $d = 10$ min., the buffer needs to cache 30-minute video data. Almost all broadcasting schemes require at least a buffer of the same size. If $B > d$ and $B \neq 3d$, we can take the play buffer size d and the forward/backward buffer size $\frac{B-d}{2}$.

5 Performance Evaluation

The performance of HB is comparatively evaluated along with the existing schemes. HB is shown to outperform the conventional approaches for popular videos.

5.1 Channel Assignment

Assume that clients' service requests arrive according to a Poisson process with rate λ. The channel allocation policy is related to service latency, discontinuity of the VCR, and phase offset d or CPE buffer size B. For the HB scheme, we must have $M \geq \lceil \frac{L}{d} \rceil - 1$, i.e., we need at least $\lceil \frac{L}{d} \rceil$ channels for this scheme.

However, if the interval between two regular channels is too large, the discontinuity of the VCR interactivity will increase. So, we need to have as many regular multicast channels as possible. Note that HB becomes SB when $d = L$, and degenerates to the basic entirety-based broadcast when $d = 0$.

A TVoD system should enable the clients to view any video, at any time, and with any VCR-like interactivity. Periodic broadcast is one of the most efficient VoD services. Generally, the segment-based broadcast VoD service with low service latency, such as SB, is difficult to support VCR-like functions, because it cannot support forward interactive operations. On the other hand, entirety-based broadcasting at fixed intervals can support restricted VCR-like functions by employing the CPE buffer. Compared with SB, HB can offer the same low service latency like SB. Moreover, HB can support the restricted VCR-like interactivity. So, HB is a broadcasting scheme better than SB. Thus, we just compare HB with the segmented patchings (they also offer both lower service latency and the VCR-like interactivity).

5.2 Service Latency Analysis and Comparison

For a video of length L, if the interval d between regular multicast channels is fixed, WSL depends on the number of segments for broadcasting or patching the leading part of size d. The number of regular multicast channels for patching is $\lceil \frac{L}{d} \rceil$, whereas that number for HB is $M = \lceil \frac{L}{d} \rceil - 1$, and therefore, if the total number of available channels is fixed, the number of broadcasting channels for HB is one more than that of patching channels for segmented patching. Moreover, the WSL of HB decreases exponentially with the increase of broadcasting channels, whereas the WSL of segmented patching linearly decreases with the increase of patching channels. So, HB is more efficient than the segmented patching in improving WSL.

For example, for a 90-minute video, let us consider HB, OSGP and TP as different multicasting strategies. Figs.6(a) and (b) plot WSLs of these schemes for $d = 10$ min. and 5 min., respectively.

We also want to find the relationship between the multicast strategy and the request rate. Intuitively, when the request rate is low, the conventional patching can offer zero-delay service. When the request rate gets higher, HB can offer a lower service latency. Thus, a practical multicast VoD system

should choose a multicast scheme based on the request rate. Our simulations support this intuition and more.

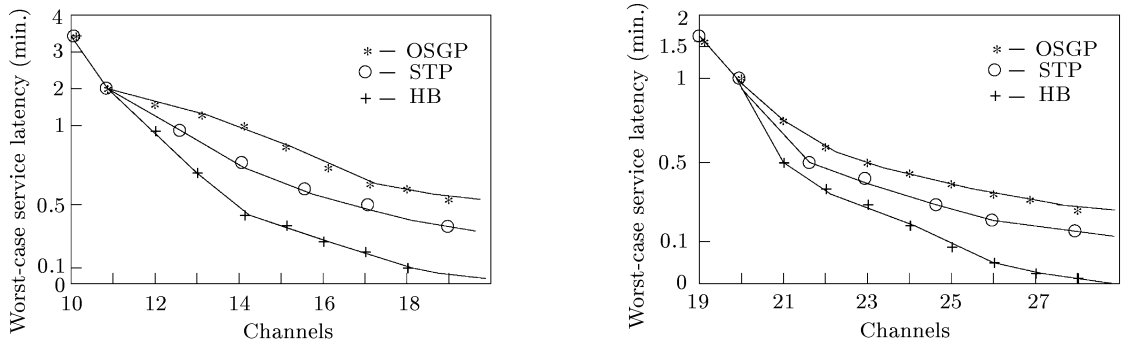

Fig.6. Comparison of HB and segmented patching. (a) $d = 10$ min. (b) $d = 5$ min.

5.3 Interactivity Analysis

Compared to both the entirety-based and segment-based broadcasting, by prefetching as much video data as possible for further interactions, the suggested active buffering technique improves the VCR interactivity, especially for forward interactions which are the main drawback of the segment-based broadcasting.

The discontinuity of interactions after exceeding the CPE buffer is determined by the average phase offset d. So, it will improve interactivity when d is small, i.e., M should be large enough. However, it is often impossible to make d small because there are only a limited number of channels. When some I-Channels are available, continuous VCR actions are supported by some TVoD interactivity protocols, such as the SAM protocol[17]. More efficient VCR interactivity are proposed in [18, 19]. (Supporting TVoD service with I-Channels is beyond the scope of this paper.)

In order to evaluate the improvement of interactivity, we use the model proposed in [18]. In this model, a set of states corresponding to different VCR actions are the designed durations and probabilities of transition to neighboring states. If the initial state is Play, then the interaction system randomly transits to other interactive states or remains in the Play state according to behavior distribution. As shown in Fig.7, transition probabilities P_i $(i = 0, \ldots, 9)$ are assigned to a set of states corresponding to different VCR actions. A client stays in each interaction state for an exponentially-distributed period of time, d_i $(i = 0, 1, 2, \ldots, 8)$ are the mean durations for the corresponding interactive states $(d_1 = 0)$.

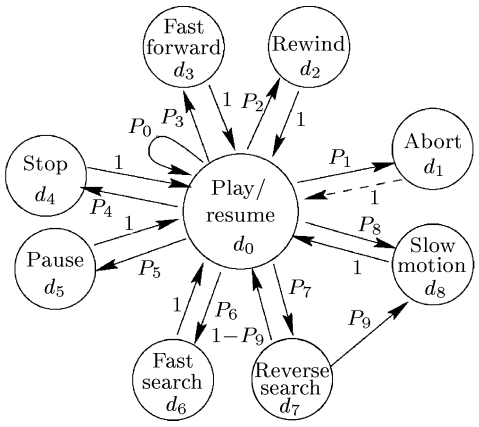

Fig.7. VCR interactivity model.

5.4 The Simulation Results

5.4.1 The Simulation of Admission of Clients

We have chosen 10 popular videos of $L = 90$ min. each in our simulation. The requests arrive according to a Poisson process with the rate ranging from 5 to 100 per minute. The video selection follows a Zpif-like distribution with a skew factor of 0.271. Two regular multicast intervals (patching

windows), $d = 5$ or 10 min., are used. The number of available channels is varied from 190 to 290 for the 5-minute interval, and from 100 to 200 for the 10-minute interval. The results are obtained from the 10-hour data simulations. We choose MFQLF as the batching scheme, which selects the pending batch with the largest size weighted by the best factor $p_i^{-1/2}$ to serve next (p_i is the associated access frequency). For segmented patching and HB, we assign patching channels or broadcasting channels according to Theorem 5. The simulation parameters are summarized in Table 1.

Table 1. Parameters Setting

Parameter	Default	Variation
Number of videos N	10	N/A
Skew factor	0.271	N/A
SB width W	52	N/A
Video length (minutes)	90	N/A
Patching window (minutes)	5	5–10
CPE buffer size B (minutes)	15	15–30
Request rate λ (per minute)	40	5–100
Available bandwidths (channels)	220	100–290

We focus on the effect of the following three parameters on the mean service latency: request rate, available channels, and the phase offset (patching window). We take $d = 5$ min., Fig.8(a) shows the effects of request rates: HB offers low MSL when the request rate is high, especially for $\lambda > 30$, when the mean request-arrival interval ($\frac{1}{\lambda}$) is roughly a half of the segment size. Fig.8(b) shows the effects of the available channels when $\lambda = 40$: HB offers a lower MSL when there are not enough channels available. The case of $d = 10$ min. yields results similar to the case of $d = 5$ min.

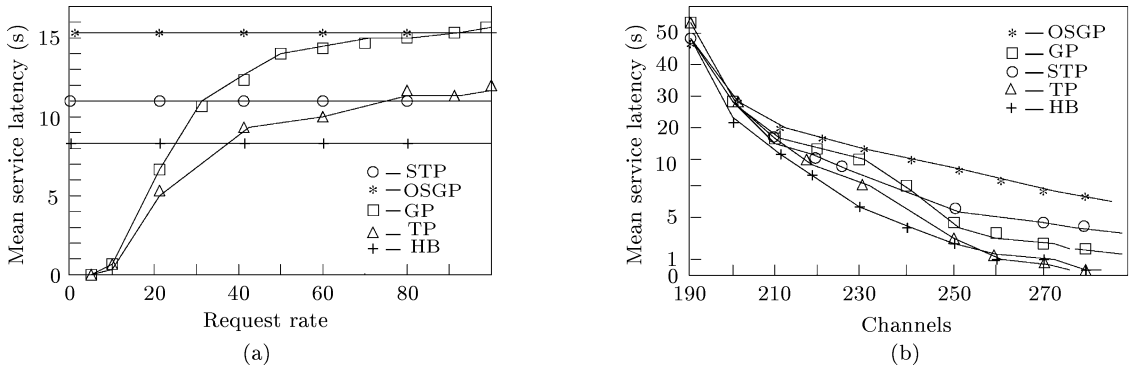

Fig.8. Mean service latency ($d = 5$ min.). (a) Effects of request rates (Channel = 220). (b) Effects of available channels ($\lambda = 40$).

5.4.2 The Simulation of Client Interactions

We compare the interactivity of HB with other similar schemes. Actually, we need to evaluate the effect of active buffering. For tractability, our simulation assumes that Slow Motion is requested only after Reverse Search, i.e., $P_8 = 0$, and $P_9 = 0.5$. The other transition possibilities are summarized in Table 2.

Table 2. Transition Probabilities from Play/Resume

Parameter	P_0	P_1	P_2, P_3	P_4	P_5	P_6, P_7
Default	0.50	0.04	0.08	0.06	0.08	0.08

The default values of the mean duration for various interactive states are given in Table 3. Meanwhile, the speedup factors of Fast Forward/Rewind and Fast Search/Reverse Search are defined as K_0, K_1, respectively, and the speeddown factor of Slow Motion is defined as K_2. We take $K_0 = 10$, $K_1 = 3$, and $K_2 = 2$.

Table 3. Mean Interactive Durations

Parameter	d_0	d_1	d_2, d_3	d_4, d_5	d_6, d_7	d_8
Default	10	0	0.5	5	2	2

We studied the QoS of interactions for a video of 90 minutes length in our simulation. Requests arrive according to a Poisson process with the rate ranging from 1 to 9 per minute. The phase offset is varied from 1 to 15 minutes, and the CPE buffer size is ranging from 3 to 45 minutes. The results are collected from the 10-hour simulations.

We find the QoS of interactions is not related to the request rate and the interaction frequency due to the scalability of HB. Thus, we focus on the effect of the CPE buffer size or the phase offset on interactivity. Fig.9(a) shows the relationship between the phase offset and the blocking rate: Active buffering offers a lower blocking rate than no active buffering. Note that the blocking rate is high when the phase offset is low, because the CPE buffer is assumed to be small. If the CPE buffer size increases, the blocking rate decreases. Fig.9(b) shows the relationship between the phase offset and the mean waiting time of interactions: Active buffering lowers the mean interaction delay.

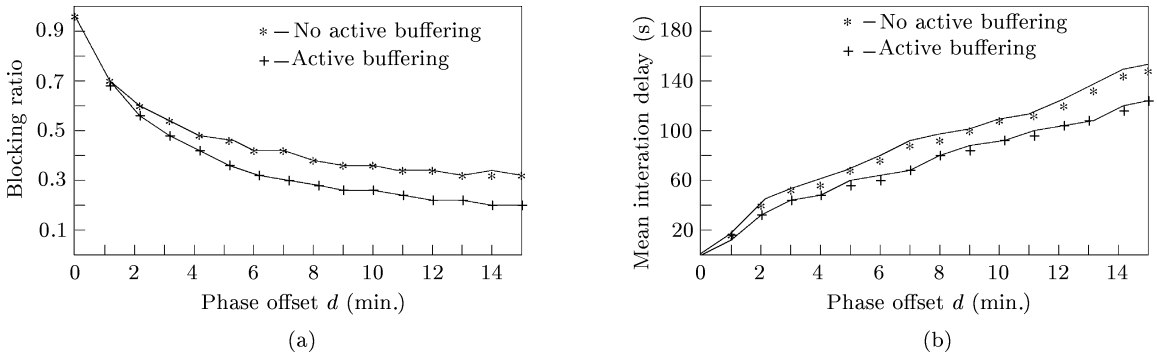

Fig.9. QoS of interactions ($\lambda = 5$, Buffer size = 3d). (a) Blocking rate as a function of phase offsets. (b) Mean waiting time as a function of phase offsets.

6 Conclusion

Server storage and network I/O throughput are known to be a serious bottleneck in VoD systems. Many researchers have shown that multicast is a good remedy for this problem. In this paper, we examine the existing multicast schemes, discuss their drawbacks, and propose a new approach called the *Hybrid Broadcast* (HB) that supports VCR interactions with low service latency. To study the performance of HB, we analyze the segmented patching, a deformation of patching when the request rate is high. We have also shown that different request rates can be handled by selecting different multicast strategies for optimal performance, and active buffering can be used to improve interactivity. Our results indicate that HB can achieve significantly better performance than the conventional approaches. HB provides low access latency and improves VCR interactivity for popular videos even when there are only a limited number of channels available, without incurring any additional bandwidth cost.

References

[1] Zipf G. Human Behaviour and the Principle of Least Effort. Addison-Wesley, 1949.
[2] Ma Huadong, Kang G Shin. A new scheduling scheme for multicast true VoD service. *Lecture Notes in Computer Science* (*Proc. PCM2001*), Springer, Oct., 2001, Vol.2195, pp.708–715.
[3] Gao L, Towsley D. Supplying instantaneous video-on-demand services using controlled multicast. In *Proc. IEEE Multimedia Computing and Systems,* Florence, Italy, June, 1999, pp.117–121.
[4] Dan A, Sitaram D, Shahabuddin P. Scheduling policies for an on-demand video server with batching. In *Proc. of ACM Multimedia'94,* San Francisco, October, 1994, pp.15–23.

[5] Golubchik L, Lui J, Muntz R. Adaptive piggybacking: A novel technique for data sharing in video-on-demand storage servers. *Multimedia Systems,* 1996, 4(3).
[6] Hua K A, Cai Y, Sheu S. Patching: A multicast technique for true video-on-demand services. In *Proc. ACM Multimedia'98,* Bristol, UK, Sept., 1998, pp.191–200.
[7] Cai Ying, Hua Kien A. Optimizing patching performance. In *Proc. SPIE's Conference on Multimedia Computing and Networking (MMCN'99),* San Jose, January, 1999, pp.204–216.
[8] Cai Ying, Hua Kien A. An efficient bandwidth-sharing technique for true video-on-demand systems. In *Proc. ACM Multimedia'99,* Orlando, Nov., 1999, pp.211–214.
[9] Viswanathan S, Imielinski T. Pyramid broadcasting for video-on-demand services. In *Proc. the SPIE Multimedia Computing and Networking Conference,* San Jose, California, 1995, Vol.2417, pp.66–77.
[10] Viswanathan S, Imielinski T. Metropolitan area video-on-demand service using pyramid broadcasting. *Multimedia Systems,* August 1996, 4(4): 197–208.
[11] Aggarwal C C, Wolf J L, Yu P S. A permutation-based pyramid broadcasting scheme for video-on-demand systems. In *Proc. IEEE Int. Conf. on Multimedia Computing and Systems,* Hiroshima, Japan, June, 1996, pp.118–126.
[12] Hua K A, Sheu S. Skyscraper broadcasting: A new broadcasting scheme for metropolitan video-on-demand systems. In *Proc. ACM SIGCOMM'97,* Sept., 1997, pp.89–100.
[13] Gao L, Kurose J, Towsley D. Efficient schemes for broadcasting popular videos. In *Proceedings of NOSSDAV,* Cambridge, UK, July, 1998.
[14] Eager D L, Vernon M K. Dynamic skyscraper broadcasts for video-on-demand. In *Proc. MIS'98,* Istanbul, Turkey, Sept., 1998.
[15] Gao L, Zhang Z-L, Towsley D. Catching and selective catching: Efficient latency reduction techniques for delivering continuous Multimedia streams. In *Proc. ACM Multimedia'99,* Orlando, Nov., 1999, pp.203–206.
[16] Emmanuel L Abram-Profeta, Kang G Shin. Scheduling video programs in near video-on-demand systems. In *Proc. ACM Multimedia'97,* Seattle, Nov., 1997, pp.359–369.
[17] Wanjiun Liao, Victor O K Li. The split and merge protocol for interactive video-on-demand. *IEEE Multimedia,* Oct.–Dec., 1997, 4(4): 51–62.
[18] Emmanuel L Abram-Profeta, Kang G Shin. Providing unrestricted VCR capability in multicast video-on-demand systems. In *Proc. IEEE Int. Conference on Multimedia Computing and Systems,* June–July, 1998, pp.359–369.
[19] Ma Huadong, Kang G Shin. Multicast video-on-demand services. *ACM Computer Communication Review,* ACM Press, 2002, 32(1): 31–43.
[20] Derek Eager, Mary Vernon, John Zahorjan. Optimal and efficient multicast schedules for video-on-demand servers. In *Proc. ACM Multimedia'99,* Orlando, Nov., 1999, pp.199–202.

MA Huadong is a professor of the College of Computer Science and Technology, Beijing University of Posts and Telecommunications, China. He received his Ph.D. degree in computer science from the Institute of Computing Technology, Chinese Academy of Sciences in 1995, M.S. degree in computer science from Shenyang Institute of Computing Technology, the Chinese Academy of Sciences in 1990 and B.S. degree in mathematics from Henan Normal University in 1984. His research interests are multimedia, computer animation and networking software, and he has published over 40 papers and 2 books in these fields.

Kang G. Shin is a professor and Director of the Real-Time Computing Laboratory, Department of Electrical Engineering and Computer Science, The University of Michigan, USA. He has supervised the completion of 40 Ph.D. theses, and authored/coauthored over 600 technical papers and numerous book chapters in the areas of distributed real-time computing and control, computer networking, fault-tolerant computing, and intelligent manufacturing. His current research focuses on QoS sensitive computing and networking with emphases on timeliness and dependability. He received the B.S. degree in electronics engineering from Seoul National University, Seoul, Korea in 1970, and both the M.S. and Ph.D. degrees in electrical engineering from Cornell University, Ithaca, New York in 1976 and 1978, respectively. He is an IEEE fellow, ACM fellow and member of the Korean Academy of Engineering.