25

Cost Effective and Scalable Video Streaming Techniques

Kien A. Hua **Mounir Tantaoui**
School of Electrical Engineering and Computer Science
University of Central Florida
Orlando, Florida, USA
kienhua@cs.ucf.edu, tantaoui@cs.ucf.edu

1. INTRODUCTION

Video on demand (VOD) is a key technology for many important applications such as home entertainment, digital libraries, electronic commerce, and distance learning. A VOD system allows geographically distributed users to play back any video from a large collection stored on one or more servers. Such a system may also support VCR-like interactions such as fast forward, fast rewind, jump forward, jump backward, and pause. To accept a client request, the VOD server must allocate enough resources to guarantee a jitter-free playback of the video. Such resources include storage and network I/O bandwidth. Sufficient storage bandwidth must be available for continuous transfer of data from storage to the network interface card (NIC), which in turn needs enough bandwidth to forward the stream to remote clients. Due to the high bandwidth requirement of video streams (e.g., 4 megabits/second for MPEG-2 videos), server bandwidth determines the number of clients the server is able to support simultaneously [18]. The simplest VOD system dedicates one video stream for each user (*Unicast*). Obviously, this approach is very expensive and not scalable.

To support a large number of users, requests made to the same video can be batched together and serviced with a single stream using *multicast*. This is referred to as *Batching*. This solution is quite effective since applications typically follow the 80-20 rule. That is, 20% of the data are requested 80% of the time. Since majority of the clients request popular videos, these clients can share the video streams and significantly reduce the demand on server bandwidth. A potential drawback of this approach is the long service delay due to the batching period. A long batching period makes the multicast

more efficient, but would result in a long wait for many clients. Some may decide to renege on their service request. On the other hand, a batching period too short would defeat the purpose of using multicast. This scheme also has the following limitation. A single video stream is not adaptable to clients with different receiving capability. Supporting VCR-like interactivity is also difficult in this environment.

In this chapter we present several cost-effective techniques to achieve scalable video streaming. We describe a typical architecture for video-on-demand streaming in Section 2. In Sections 3 and 4, we discuss periodic broadcast techniques and multicast techniques, respectively. A new communication paradigm called *Range Multicast* is introduced in Section 5. Techniques to handle VCR interactions are presented in Section 6. Section 7 describes other techniques that deal with user heterogeneity. Finally, we summarize this chapter in Section 8.

2. VOD SYSTEM ARCHITECTURE

A server channel is defined as a unit of server capacity (i.e., server bandwidth and computing resources) required to support a continuous delivery of video data. The number of channels a server can have typically depends on its bandwidth. These channels are shared by all clients. Their requests are queued at the server, and served according to some scheduling policy when a free channel becomes available. When a service is complete, the corresponding channel is returned to the pool to serve future requests. When multicast is used for video delivery, a channel is allocated to load the video from storage and deliver it to a group of clients simultaneously as shown in Figure 1. The communication tree illustrates the one-to-many delivery mechanism. At the receiving end, video data are either sent to the video player to be displayed or temporarily stored in a disk buffer for future display.

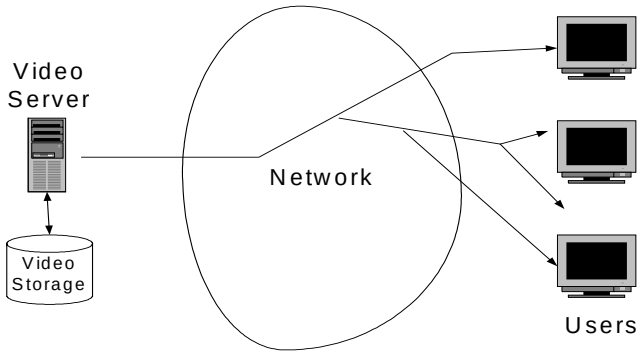

Figure 1. A multicast delivery

2.1 SERVER ARCHITECTURE

A typical architecture of a video server is illustrated in Figure 2. The *Coordinator* is responsible for accepting requests from users. To deliver a video, the Coordinator dispatches a *Data Retrieval Handler* to load the data

blocks[1] from disk, and a *Video Delivery Handler* to transmit these blocks to the clients. Data retrieved from storage are first staged in a streaming buffer. The *Video Delivery Handler* feeds on this buffer, and arranges each data block into a packet. The header of the packet contains the location of the block in the video file. This information serves as both the timestamp and the sequence number for the client to order the data blocks for correct display. The *Directory Manager* maintains a video directory that keeps information about the videos currently in delivery, such as video title, the multicast address of the channel currently in use, and other important characteristics of the video. The system administrator, through a *graphical user interface* (GUI), can perform various administrative works such as adding or removing a video from the database.

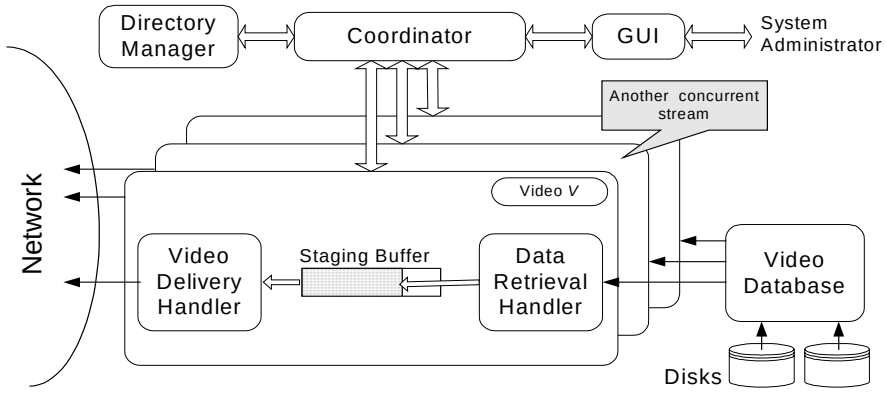

Figure 2. Server architecture

2.2 CLIENT ARCHITECTURE

The components of a typical client software are illustrated in Figure 3. The *Coordinator* is responsible for coordinating various server activities. The main role of the *Directory Explorer* is to maintain an up-to-date directory of the videos. To request service, the user can select a video from this catalogue. This action sends a command to the *Coordinator*, which in turn activates the *Loader* to receive data, and the *Video Player* to render the video onto the screen. The Loader and the Video Player communicates through a staging buffer. In some system, the incoming stream can also be saved to disk for future use, or to support VCR-like interactions.

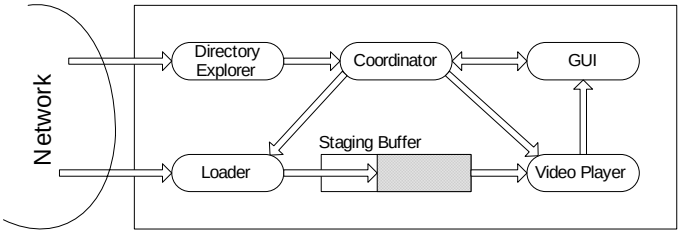

Figure 3. Client architecture

[1] A block is the smallest unit for disk access

3. PERIODIC BROADCAST TECHNIQUES

Periodic Broadcast is a highly scalable solution for streaming popular videos. In this environment, a video is segmented into several segments, each repeatedly broadcast on a dedicated channel. A client receives a video by tuning to one or more appropriate channels at a time to download the data. The communication protocol ensures that the broadcast of the next video segment is available to the client before the playback of the current segment runs out. Obviously, this scheme is highly scalable. The system can serve a very large community of users with minimal server bandwidth. In fact, the bandwidth requirement is independent of the number of users the system is designed to support. A limitation of this approach is its non-zero service delay. Since each client cannot begin the playback until the next occurrence of the first segment, its broadcast period determines the worst service latency. Many periodic broadcast techniques have been designed to keep this delay, or the size of the first segment, as small as possible to provide near-on-demand services. These techniques can be classified into two major categories. The first group, called *Server-Oriented Approach*, includes techniques that reduce service latency by increasing server bandwidth. Methods in the second category, called *Client-Oriented Approach*, improve latency by requiring more client bandwidth. We discuss these two approaches in this section.

To facilitate the following discussion, we assume a video v of L seconds long. A portion of the server bandwidth, B Mbits/sec, is allocated to v. This dedicated bandwidth is organized into K logical channels by time multiplexing. In other words, we repeatedly broadcast the data segments of v on K channels. The playback rate of v is b Mbit/sec.

3.1 SERVER-ORIENTED APPROACH: INCREASING SERVER BANDWIDTH

We first focus on techniques that allow the users to reduce service latency by increasing only server bandwidth. The rationale for this approach is that server bandwidth, shared by a large community of users, contributes little to the overall cost of the VOD environment. Researchers in this camp argue that this solution is much less expensive than the Client-Oriented Approach which demands each user to equip with substantial client bandwidth, e.g., using a T1 line instead of DSL or a cable modem.

3.1.1 Staggered Broadcasting

Staggered Broadcasting [8][9] is the earliest and simplest video broadcast technique. This scheme staggers the starting times for the video v evenly across K channels. In other words, if the first channel starts broadcasting video v at the playback rate b at time t_0, the second channel starts broadcasting the same video at time $t_0 + L/K$, the third channel at time $t_0 + 2*L/K$, and so on. The difference in the starting times, L/K, is called the *phase offset*. Since a new stream of video v is started every phase offset, it is the longest time any client needs to wait for this video.

Another way to implement the Staggered Broadcasting scheme is as follows. Video v can be fragmented into K segments ($S_1, S_2, S_3, .., S_K$) of equal size,

each of length L/K. Each channel C_i, $1 \le i \le K$, repeatedly broadcasts segment S_i at the playback rate b. A client requesting video v tunes to channel C_1 and waits for the beginning of segment S_1. After downloading segment S_1, the client switches to channel C_2 to download S_2 immediately. This process is repeated for the subsequent data segments until segment S_K is downloaded from channel C_k.

The advantage of Staggered Broadcasting is that clients download data at the playback rate. They do not need extra storage space to cache the incoming data. This simple scheme, however, scales only linearly with increases to the server bandwidth. Indeed, if one wants to cut the client waiting time by half, one has to double the number of channels allocated to the video. This solution is very demanding on server bandwidth. In the following, we present more efficient techniques that can reduce service latency exponentially with increases in server bandwidth.

3.1.2 Skyscraper Broadcasting

In *Skyscraper Broadcasting* [18], the server bandwidth of B Mbit/sec is divided into $\lfloor B/b \rfloor$ logical channels of bandwidth b. Each video is fragmented into K data segments. The size of segment S_n is determined using the following recursive function:

$$f(n) = \begin{cases} 1 & n = 1, \\ 2 & n = 2 \text{ or } 3, \\ 2 \cdot f(n-1) + 1 & n \bmod 4 = 0, \\ f(n-1) & n \bmod 4 = 1, \\ 2 \cdot f(n-1) + 2 & n \bmod 4 = 2, \\ f(n-1) & n \bmod 4 = 3. \end{cases} \qquad (1)$$

Formula 1 expresses the size of each data segment in term of the size of the first segment. Expanding this formula gives us the following series referred to as the *broadcast series*:

[1, 2, 2, 5, 5, 12, 12, 25, 25, ...]

That is, if the size of the first data segment is D_1, the size of the second and third segments are $2 \cdot D_1$, the fourth and fifth are $5 \cdot D_1$, sixth and seventh are $12 \cdot D_1$, and so forth. This scheme limits the size of the biggest segments to W units or $W \cdot D_1$. These segments stack up to a skyscraper, thus the name *Skyscraper Broadcasting*. W is called the *width* of the skyscraper.

Skyscraper Broadcasting repeatedly broadcasts each segment on its dedicated channel at the playback rate b. Adjacent segments having the same size form an *odd* or *even group* depending on whether their sizes are odd or even, respectively. Thus, the first segment is an odd group by itself; the second and third segments form an even group; the fourth and fifth form an odd group; the sixth and seventh form an even group; and so on. To download the video, each client employs two concurrent threads - an *Odd Loader* and *an Even Loader*. They download the odd groups and the even groups, respectively. When a loader reaches the first W-segment, the client

uses only this loader to download the remaining *W*-segments sequentially to minimize the requirement on client buffer space.

Figure 4 gives an example of the Skyscraper Broadcasting scheme, where three clients *x*, *y*, and *z* requested the video *v* just before time slots 5, 10, and 11, respectively. The segments downloaded by each of the three clients are filled with a distinct texture. Let us focus on Client *x* whose segments are black. Its Odd Loader and Even Loader start downloading the first and second segments, respectively, at the beginning of the fifth time slot. When the second segment is exhausted, the Even Loader switches to download the third segment on Channel 3. Similarly, when the first segment is exhausted, the Odd Loader turns to download first the fourth and then the fifth segments. After the Even Loader has finished downloading the third segment on Channel 3, this loader tunes into Channel 6 to wait for the next occurrence of segment 6 at the beginning of time slot 13. If *W* is set to 12 in this example, this client will continue to use the Even Loader to download the remaining *W*-segments. The playback timing of the downloaded segments is illustrated at the bottom of Figure 4. We note that each segment is available for download before it is required for the playback.

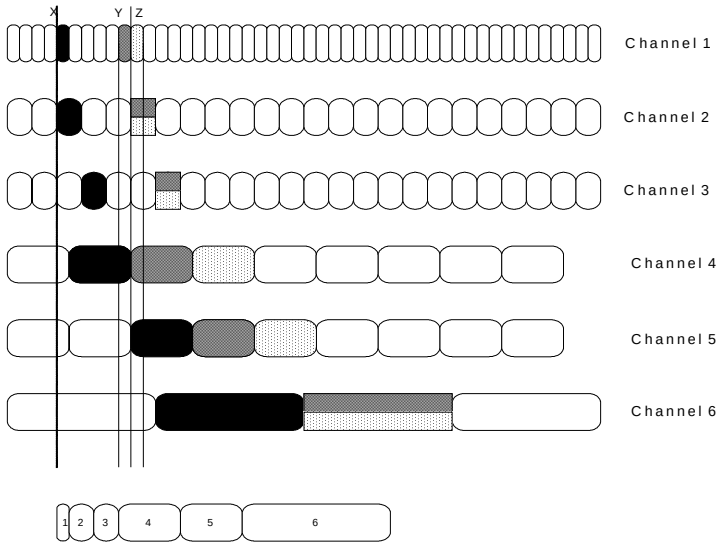

Figure 4. Skyscraper downloading scheme

The worst waiting time experienced by any client, which is also the size of the first segment D_1, is given by the following formula:

$$D_1 = \frac{L}{\sum_{i=1}^{K} \min(\ f\ (i), W\ )}$$

The major advantage of this approach is the fixed requirement on client bandwidth regardless of the desired service latency. To achieve better service latency, one needs only add server bandwidth. This additional cost is usually negligible because the access latency can be reduced at an exponential rate; and server resources are shared by a very large user

community. In practice, it is difficult to provide near-on-demand services using Staggered Broadcasting. Skyscraper technique addresses this problem efficiently. As an example, using 10 channels for a 120-minute video, Staggered Broadcasting has a maximum waiting time of 12 minutes while it is less than one minute for Skyscraper Broadcasting. Adding only a few more channels can further reduce this waiting time.

3.1.3 Client-Centric Approach

The *Client-Centric-Approach* (CCA) [16] is another periodic broadcast technique that allows one to improve service delay by adding only server resources once the client capability has been determined. As in the *Skyscraper* technique CCA divides server bandwidth into *K* logical channels, each repeatedly broadcasts a distinct video segment. The fragmentation function is given in Formula 2, where the parameter *c* denotes the maximum number of channels each client can tune into at one time to download simultaneously *c* segments:

$$f(n) = \begin{cases} 1 & n = 1, \\ 2 \cdot f(n-1) & n \bmod (c) \neq 1, \\ f \cdot (n-1) & n \bmod (c) = 1. \end{cases} \qquad (2)$$

CCA can be viewed as a generalization of the Skyscraper technique in that each transmission group can have more than two segments, and the number of data loaders is not limited to two as in Skyscraper Broadcasting. CCA enables applications to exploit the available client bandwidth. This approach, however, is not the same as the Client-Oriented Approach presented in Section 3.2, which, given a fixed client bandwidth, cannot improve the service delay by adding only server resources.

Similar to Skyscraper Broadcasting, CCA also limits the sizes of larger data fragments to *W*. At the client end, the reception of the video is done in terms of *transmission groups*. The video segments are grouped into *g* transmission groups where $g = \lceil K/c \rceil$. Therefore, each group has c segments except for the last group. To receive the video, a client uses *c* loaders; each can download data at the playback rate *b*. When a loader L_i finishes downloading segment S_i in $group_j$, this loader switches to download segment S_{i+c} in $group_{j+1}$. Since segments in the same group (of different sizes) are downloaded at the same time, continuity within the same group is guaranteed. Furthermore, since the size of the last segment in $group_j$ is always the same size as the first segment of the next $group_{j+1}$, continuity across group boundaries is also guaranteed. Comparing to the Skyscraper scheme, CCA uses extra client bandwidth to further reduce the access latency. For L=120 minutes (video length), K=10 channels, and c=3 loaders, CCA cut the access latency by half.

3.1.4 Striping Broadcasting

Striping Broadcasting [26] employs a 2D data fragmentation scheme as follows. Let *N* be the index of the first W-segment. The size of each segment *i*, denoted by L_i, is determined using the following equation:

$$L_i = \begin{cases} 2^{i-1} \cdot L_1 & i \in [2, N-1], \\ 2^{N-1} \cdot L_1 & i \in [N, K]. \end{cases}$$

That is, the size of the first *N-1* segments increase geometrically, while the sizes of the other segments (i.e., W-segments) are kept the same. Each of the W-segments is further divided into two equally-sized fragments called *stripes*. Compared to Skyscraper technique, Striping Broadcasting employs a faster segment-size progression. This results in a smaller first segment, and therefore better service latency. Since the size of all segments must be equal to the size of the entire video, we can compute the worst access delay as follows:

$$L_1 = \frac{L}{(K-N+2) \cdot 2^{N-1}}$$

To deliver a video, the server periodically broadcasts each of the first N-1 segments, at the playback rate, on its own channel. Each stripe of the W-segments is also periodically broadcast at half the playback rate. To allow the client to download the W-segments as late as possible to save buffer space, phase offsets are employed in the broadcast. Let D_i denote the phase offset for segment i where $1 \le i < N$; and D_{i1} and D_{i2} represent the phase offsets for the first and second stripes, respectively, of segment i where $N \le i \le K$. The phase offsets are calculated as follows:

$$\begin{pmatrix} D_1 \\ D_i \\ D_{i1} \\ D_{i2} \end{pmatrix} = \begin{cases} 0 & \\ 2^{i-2} \cdot L_1 & i \in [2, N-1] \\ 2^{N-2} \cdot L_1 & i \in [N, K] \\ 2^{N-1} \cdot L_1 & i \in [N, K] \end{cases}$$

A broadcast example using the non-uniform phase delay is shown in Figure 5. The server broadcasts an index at the beginning of each occurrence of the first segment to inform the client when to download the remaining segments.

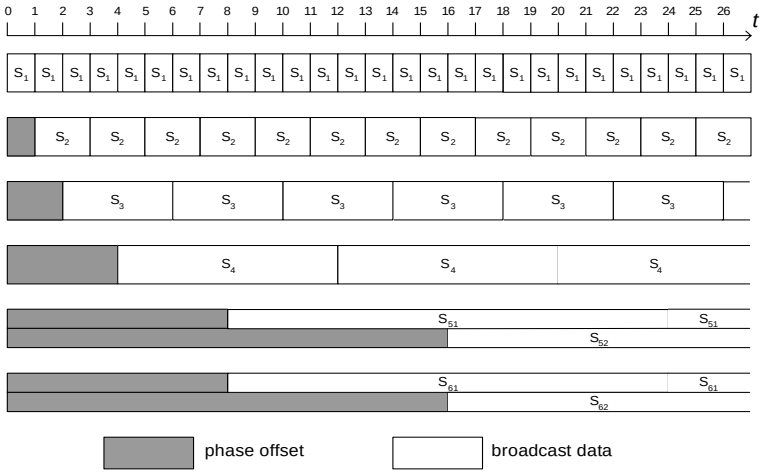

Figure 5. Broadcast schedule when N=5 and K=6

The client uses three concurrent threads to download the video from up to three channels simultaneously. The first loader first downloads the first segment as soon as possible. All three loaders then download the remaining segments in the order specified in the broadcast index.

Striping Broadcasting betters Skyscraper Broadcasting in terms of service latency. The former also requires less client buffer space. As an example, to limit the service delay under 24 seconds for a one-hour video, Striping Broadcasting requires the client to cache no more than 15% the size of the video while Skyscraper Broadcasting requires 33%. Skyscraper Broadcasting, however, is less demanding on client bandwidth since it uses only two concurrent download threads.

A near-video-on-demand system based on Striping Broadcasting was implemented at the University of Central Florida [26]. The prototype runs on Microsoft Windows operating system. The server software lets the content publisher select and specify suitable parameters for videos to be broadcast (see Figure 6). Once the selection is done, the publisher can multicast the video directory to a selected multicast group.

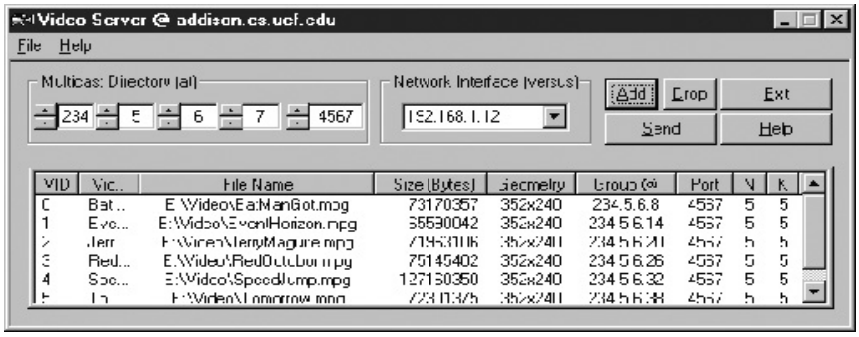

Figure 6. Server software: main window

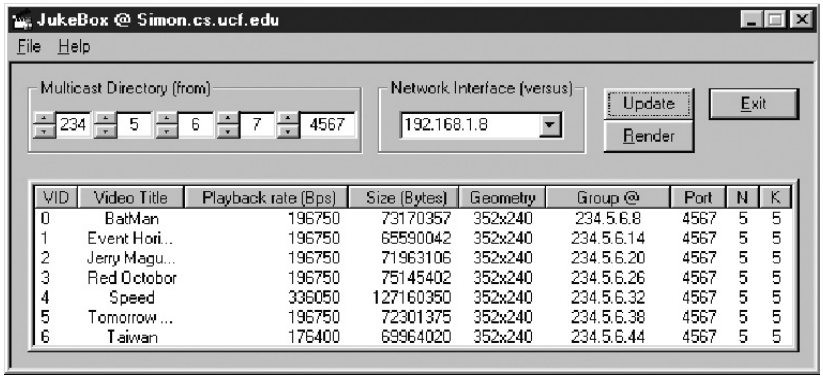

Figure 7. Client software: JukeBox

A user wishing to use the service selects the desired video through a client software called *JukeBox* (see Figure 7). In response, a video player and a control panel pop up (see Figure 8); and the selected video is subsequently

played out. The client software is built on Microsoft DirectShow and currently supports MPEG-1 system files containing both video and audio tracks. Other file formats supported by DirectShow can also be supported without much modification to the software.

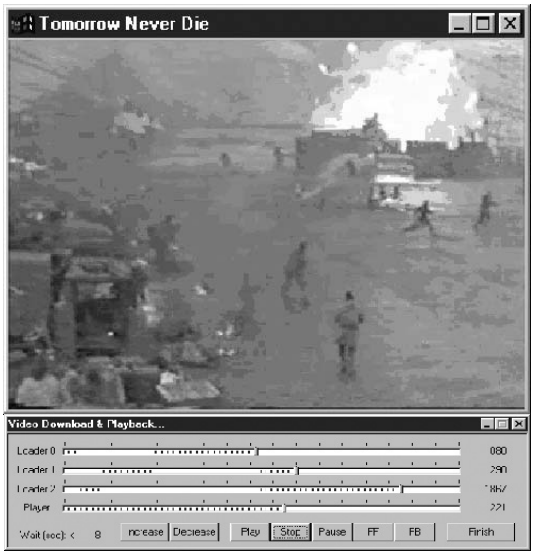

Figure 8. Playback of a video

Experiments with the prototype on a local area network were conducted. Four video clips, each over 5 minutes long, were repeatedly broadcast in these experiments. The number of jitters observed was very small (less than three per playback). This problem was due to packet loss, not a result of the broadcast technique. Each jitter was very brief. The playback quality was comparable to that of commercial streaming systems. The service delays were measured to be less than 13 seconds.

3.2 CLIENT-ORIENTED APPROACH: INCREASING CLIENT BANDWIDTH

All the broadcast techniques, discussed so far, aim at enabling the users to improve service latency by adding only server bandwidth. In this subsection, we discuss techniques that require increases to both server and client bandwidth in order to improve system performance.

3.2.1 Cautious Harmonic Broadcasting

Caution Harmonic Broadcasting [22] partitions each video into K equally-sized segments. The first channel repeatedly broadcasts the first segment S_1 at the playback rate. The second channel alternatively broadcasts S_2 and S_3 at half the playback rate. Each of the remaining segments S_i is repeatedly broadcast on its dedicated channel at $1/(i\text{-}1)$ the playback rate. Although this scheme uses many channels to deliver a video, the total bandwidth grow slowly following the harmonic series, typically adding up to only $5b$ or $6b$.

A client plays back a video by downloading all the segments simultaneously. This strategy has the following drawbacks:

- The client must match the server bandwidth allocated to the longest video. The requirement on client bandwidth is therefore very high making the overall system very expensive.
- Improving access delay requires adding bandwidth to both server and client bandwidth. This makes system enhancement very costly.
- Since the client must receive data from many channels simultaneously (e.g., 240 channels are required for a 2-hour video if the latency is kept under 30 seconds), a storage subsystem with the capability to move their read heads fast enough to multiplex among so many concurrent streams would be very expensive.

3.2.2 Pagoda Broadcasting

As in the Harmonic scheme, *Pagoda Broadcasting* [23] also divides each video into equally-sized segments. However, it addresses the problems of having too many channels by allowing segments to share channels. The number of segments allocated to each channel is determined according to the following series:

$$\{1, 3, 5, 15, 25, 75, 125, \ldots\}$$

Since the numbers grow very fast in the above series, this scheme requires much less channels than in the Harmonic Broadcasting. The segments assigned to a channel do not have to be consecutive. An example is given in Figure 9. It shows that 19 segments are repeatedly broadcast on four channels. The idea is to broadcast each segment at least once every period in term of the Harmonic Broadcast scheme.

1st Stream	S_1	S_1	S_1	S_1	S_1	S_1	S_1	S_1	S_1	S_1
2nd Stream	S_2	S_4	S_2	S_5	S_2	S_4	S_2	S_5	S_2	S_4
3rd Stream	S_3	S_6	S_8	S_3	S_7	S_9	S_3	S_6	S_8	S_3
4th Stream	S_{10}	S_{11}	S_{12}	S_{13}	S_{14}	S_{15}	S_{16}	S_{17}	S_{18}	S_{19}

Figure 9. A broadcast example in Pagoda Broadcast

Each channel broadcasts data at the playback rate. A client requesting a video downloads data from all the channels simultaneously. The benefit of Pagoda Broadcasting is to achieve a low server bandwidth requirement as in Harmonic Broadcast without the drawback of using many channels. Pagoda Broadcasting, however, has not addressed the high demand on client bandwidth. For instance, keeping service delay less than 138 seconds for a 2-hour video requires each client to have a bandwidth five times the playback rate. Furthermore, performance enhancement or adding a longer video to the database may require the clients to acquire additional bandwidth. In comparison with the Server-Oriented Approach, presented in Section 3.1, the savings in server bandwidth under Pagoda Broadcasting is not worth the significantly more expensive client hardware. Nevertheless, this scheme can be used for local-scale applications, such as corporate training and campus information systems, which rely on an intranet for data transmission.

4. MULITCAST TECHNIQUES

In a Multicast environment, videos are not broadcast repeatedly, but multicast on demand. In this section, we first discuss the *Batching* approach, and then present a more efficient technique called *Patching*.

4.1 BATCHING

In this environment, users requesting the same video, within a short period of time, are served together using the multicast facility. Since there could be several such batches of pending requests, a scheduler selects one to receive service according to some queuing policy. Some scheduling techniques for the Batching approach are as follows:

- **First-Come-First-Serve** [2][10]: As soon as some server bandwidth becomes free, the batch containing the oldest request with the longest waiting time is served next. The advantage of this policy is its fairness. Each client is treated equally regardless of the popularity of the requested video. This technique, however, results in a lower system throughput because it may serve a batch with few requests, while another batch with many requests is pending.

- **Maximum Queue Length First** [2]: This scheme maintains a separate waiting queue for each video. When server bandwidth becomes available, this policy selects the video with the most number of pending requests (i.e., longest queue) to serve first. This strategy maximizes server throughput. However, it is unfair to users of less popular videos.

- **Maximum Factored Queued Length First** [2]: This scheme also maintains a waiting queue for each video. When server resource becomes available, the video v_i selected to receive service is the one with the longest queue weighted by a factor $1/\sqrt{f_i}$, where f_i denotes the access frequency or the popularity of v_i. This factor prevents the system from always favoring more popular videos. This scheme presents a reasonably fair policy without compromising system throughput.

The benefit of periodic broadcast is limited to popular videos. In this sense, Batching is more general. It, however, is much less efficient than periodic broadcast in serving popular videos. A hybrid of these two techniques, called *Adaptive Hybrid Approach* (AHA), was presented in [17] offering the best performance. This scheme periodically assesses the popularity of each video based on the distribution of recent service requests. Popular videos are repeatedly broadcast using Skyscraper Broadcasting while less demanded ones are served using Batching. The number of channels used for periodic broadcast depends on the current mix of popular videos. The remaining channels are allocated to batching. The AHA design allows the number of broadcast channels allocated to each video to change in time without disrupting the on-going playbacks.

4.2 PATCHING

All the techniques discussed so far can only provide near-on-demand services. A multicast technique that can deliver videos truly on demand is desirable. At first sight, making multicast more efficient and achieving zero service delay seems to be two conflicting goals. We discuss in this subsection one such solution called *Patching*.

The *patching* technique [7][15][24] allows a new client to join an on-going multicast and still receive the entire video stream. This is achieved by receiving the missed portion in a separate *patching stream*. As this client displays data arriving in the patching stream, it caches the multicast stream in a buffer. When the patching stream terminates, the client switches to playback the prefetched data in the local buffer while the multicast stream continues to arrive. This strategy is illustrated in Figure 10. The diagram on the left shows a Client *B* joining a multicast *t* time units late, and must receive the first portion of the video through a patching stream. The right diagram shows *t* time units later. Client *B* has now just finished the patching stream, and is switching to play back the multicast data previously saved in the local buffer.

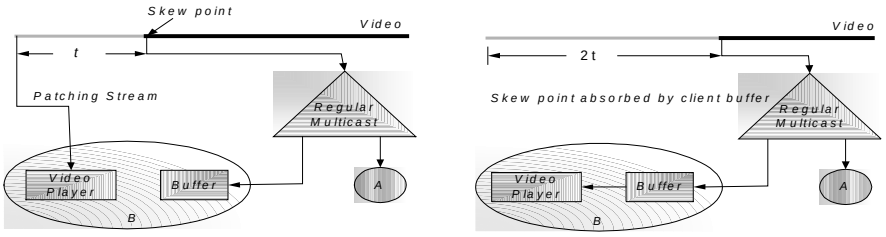

Figure 10. Patching

In a simple Patching environment, a new client can be allowed to join the current multicast if the client has enough buffer space to absorb the time skew; otherwise a new multicast is initiated. This strategy is too greedy in sharing the multicasts, and may result in many long patching streams. A better patching technique should allow only clients arriving within a *patching period* to join the current multicast. The appropriate choice of this period is essential to the performance of Patching. If the patching period is too big, there are many long patching streams. On the other hand, a small patching period would result in many inefficient multicasts. In either case, the benefit of multicast diminishes. A technique for determining the optimal patching period was introduced in [6]. A patching period is optimal if it results in minimal requirement on server bandwidth. In [6], clients arriving within a patching period are said to form a multicast group. This scheme computes D, the mean amount of data transmitted for each multicast group, and τ, the average time duration of a multicast group. The server bandwidth requirement is then given by D/τ which is a function of the patching period. The optimization can then be done by finding the patching period that minimizes this function. It was shown in [6] that under different request inter-arrival times, the optimal patching period is between 5 and 15 minutes for a 90-minute video.

5. BEYOND CONVENTIONAL MULTICAST AND BROADCAST

It has been recognized that standard multicast is inadequate for VOD applications. Patching addresses this drawback by supplementing each multicast stream with patching streams. In this subsection, we consider a new communication paradigm called *Range Multicast* (RM) [19].

The Range Multicast technique employs software routers placed at strategic locations on the *wide-area network* (WAN), and interconnected using unicast paths to implement an overlay structure to support the range multicast paradigm. As a video stream passes through a sequent of such software router nodes on the delivery path, each caches the video data into a fixed-size FIFO buffer. Before it is full (i.e., the first frame is still resident), such a buffer can be used to provide the entire video stream to subsequent clients requesting the same video.

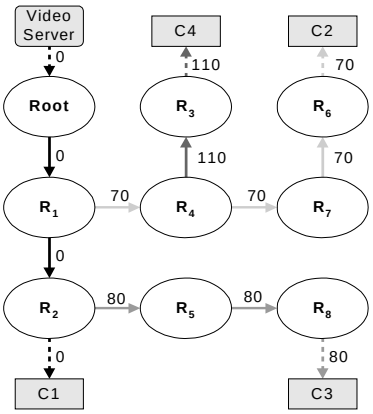

Figure 11. A range multicast example

A range multicast example is given in Figure 11. The root node is the front-end note for the server to communicate with the rest of the overlay network. We assume that each node has enough buffer space to cache up to 100 video blocks for the video stream passing through. The label on each link indicates the time stamp of a particular service requested by some client. For instance, label "0" indicates that Client C_1 requests the video at time 0. For simplicity, we assume that there is no transmission delay. Thus, C_1 can make a request at time 0 and receive the first block of the data stream instantaneously. Figure 11 illustrates the following scenario. At time 0, a node C_1 requests a video *v*. Since no node currently caches the data, the root has to allocate a new stream to serve C_1. As the data go toward C_1, all the non-root nodes along the way, R_1 and R_2, cache the data in their local buffer. At time 70, client C_2 requests the same video *v*. At this time R_1 has not dropped the first video block from its buffer, and can serve C_2. All the nodes along the path from the serving node R_1 to C_2 (i.e., R_4, R_7 and R_6) are asked to cache the video. Similarly, client C_3 requesting video *v* at time 80 can receive the service from node R_2 which still holds the first video block. At time 101, R_1 and R_2 cast out the first video block in their cache. Nevertheless, client C_4 can still join the multicast group at time 110 by receiving the full service from node R_4 which still has the first block of the

video. In this example, four clients join a range multicast at different times, but still receive the entire video stream. This is achieved using only one server stream. This characteristic is not possible with traditional multicast.

Range Multicast is a shift from conventional thinking about multicast where every receiver must obtain the same data packet at all time. In contrast, the data available from a range multicast at any time is not a "data point", but a contiguous segment of the video. In other words, a sliding window over the video is multicast to a range of receivers. This unique characteristic is important to video-on-demand applications in two ways:

- Better service latency: Since clients can join a multicast at their specified time instead of the multicast time, the service delay is zero.
- Less demanding on server bandwidth: Since clients can join an existing range multicast, the server does not need to multicast as frequently to save server resources.

Server bandwidth often dictates the performance of a VOD system. Range multicast enables such a system to scale beyond the physical limitation of the video server.

6. SUPPORTING VCR-LIKE INTERACTIONS

An important functionality of a VOD system is to offer VCR-like interactivity. This section, first introduces various interaction functions that a VOD system can support, then it discusses some techniques that handles such functions in both the multicast and the broadcast environments.

5.1 FORMS OF INTERACTIVITY

Videos are sequences of continuous frames. The frames being displayed to the monitor by a client is referred to as the current frame or the *play point*. The client watching the video can change the position of the play point to render another frame that is called the *destination point*. There exist two types of interactive functions, *continuous interactive functions* and *discontinuous functions*. In the first type of interactivity, a client continuously advances the play point one frame at a time at a speed that is different from the playback rate. The frames between the original play point and the destination point are all rendered to the monitor. Such interactions include the fast-forward and fast-reverse actions. In the second type of interactions, the user instantaneously changes the play point to another frame, such interactions are the jump forward and the jump backward interactions.
Furthermore, the interactions are divided into *forward* and *backward interactions*. If we denote by b the playback rate, Δt the time taken by the interaction, Δl the video length of the interaction in time unit, in other words the difference between the current play point and the destination point. The parameter x [11] that represents all type of interactions is defined as follow:

$$x = \frac{\Delta l / \Delta t}{b}$$

Table 1 gives the potential forward and backward interactions with the possible values of the parameter *x*.
A play action is therefore an action where its length over its duration equals to the playback rate. A fast forward action is when the length over the duration of the action is greater than the playback rate. In the case of a jump action, since the duration is zero, then the parameter x is infinite. A pause action is of length zero therefore the parameter x is zero. Fast reverse, jump backward, and play backward are the exact opposite of fast forward, jump forward, and play respectively.

Table 1. Backward and forward interactions

	Backward Interactions				Forward Interactions		
Action	Jump backward	Fast Reverse	Play backward	Pause	Play	Fast Forward	Jump Forward
x	-∞	[-x1,-1)	-1	0	1	(1,x1)	+∞

Figure 7 shows the position of the play point in the client buffer system after some interactions. The incoming frames fill up the buffer from one side while frames from the other side are discarded. After a play action of 2 frames the play point does not change its position. After a pause action, the play point follows the frame the user had paused in. After a fast forward action, the play point gradually moves toward the newest frame. After a jump backward, the play point instantly jumps towards an older frame in the buffer. From the figure, because a pause action is toward oldest frames, it is considered as a backward action. To complete the canonical world of interactions, there are two more actions, the *slow forward* with a parameter x ∈ (0,1) and the *slow backward* with x ∈ (-1,0). Both interactions are backward interaction for the same reason as the pause interaction.

Another parameter, *duration ratio* representing the degree of interactivity, is defined as the portion of time a user spends performing interactions over the time spent on normal play. *Blocked interactions* are interactions that cannot complete due to some limitation of resources.

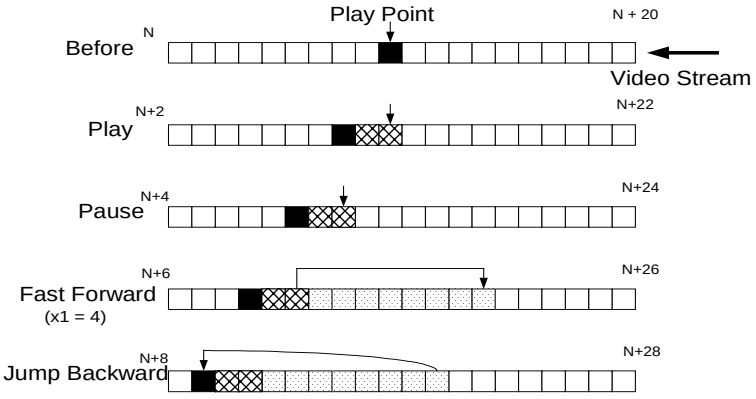

Figure 12. Play point position in the client buffer

5.2 VCR INTERACTION TECHNIQUES

This section presents some techniques that handle VCR-interactions in both the multicast and broadcast environment. These techniques try to take full advantage of the data sharing of multicast delivery while providing VCR functionality to each client individually. When a user request an interaction, the VOD system not only should it provide the interaction but it should also guarantee a jitter-free playback when the user resumes from the interaction. A good interaction technique is the one that provides a small degree of blocked interactions with a high duration ratio.

In the multicast environment, the *interactive multicast* technique [3][4] schedules the delivery of the video in some specified time slots, such time slots could range from 6 seconds to 30 minutes. The video is delivered only if some requests are pending in the queuing server. Requests during the same time slot form one multicast group. Because the period of a slot is known, users that jump forward or backward change their multicast group. A user jumping forward changes its multicast group to a group that has been scheduled earlier, while a user jumping backward changes to a multicast group scheduled after its current group. If such multicast group does not exist, an emergency channel is issued to provide the service. The continuous actions are handled in the client buffer, which contains a multicast threshold. When incoming frames exceed the threshold, a multicast change occurs. Therefore, the continuous actions are performed within the buffer's limitation.
This technique provides only limited discontinuous actions, for example if the period of the slots are 5 minutes, users cannot jump 7 minutes. Furthermore, using emergency channels to accommodate users degrades the performance of the multicast paradigm.

The *split and merge* [20] protocol offers an efficient way to provide VCR interactivity. This protocol uses two types of stream, an *S* stream for normal playback of the video, and an *I* stream to provide interactive actions. When a user initiates an interactive operation, the user split from its multicast group, and uses an *I* stream. After the completion of the interaction, the user merges from the *I* stream to a new multicast group. The split and merge scheme uses synchronized buffers located at the access node to merge smoothly the clients. If the merging fails, a new S stream is allocated to the user. The drawback of the split and merge protocol is that it requires an excessive number of I channels. Because all interactions are served using *I* streams, a blocked interaction is queued to be served as a new S stream. Thus causing a high blocking rate of VCR interactions and therefore degrading multicast scalability.

The technique proposed in [1] improve the split and merge protocol by disregarding I channels whenever they are not needed. Some interactions do not need an I stream to be performed and can join directly a new multicast.
In this technique the allocation of regular streams are independent from user's interaction behavior, an I channel is only allocated if it is available and the user's buffer is exhausted by the VCR operation.
A typical merging operation in this scheme is composed of three steps. First, some frames in the client buffer are disregarded to free some buffer space for future frames. Second, the client buffer prefetches frames from the targeted

S channel while displaying frames from the I channel. After an offset time, the client merge to the targeted channel and release the I channel.

In the broadcast situation, because the system can support unlimited number of users, using guard or emergency channels to support VCR interactivity violates the intention of using periodic broadcast. In this environment, it was observed in [11][12] that the play point can be maintained at the middle of the video segment currently in the prefetch buffer in order to accommodate interactive actions in either forward or reverse direction equally well. Because, the position of the play point profoundly influences future interactions (see figure 12), keeping it in the middle enhances the chance of their completion. This is accomplished in [11][12] by selectively prefetching the segments to be loaded depending on the current position of the play point. This scheme is called *Active Buffer Management*, in [12] it extends the staggered broadcasting scheme and in [11] it extends the client centric approach periodic broadcast. In general, Active Buffer Management can be set to take advantage of the user behavior. If the user shows more forward actions than backward actions, the play point can be kept near the beginning of the video segment in the buffer, and vice versa.

The technique proposed in [27] improves on the duration ratio of the active buffer management technique by extending the client centric approach periodic broadcast. The *broadcast-based interaction technique* improves on the overall duration ratio by periodically broadcasting a compressed version of the video. The compressed version of the video consists of only one frame out *f* frames. Clients watching the compressed segments at the playback rate will have the impression of fast playing the normal video. The broadcast-based interaction technique retains the most desirable property of the broadcast approach, namely unlimited scalability. The technique divides the client buffer space into two parts, one part holds the normal version of the video, and the second part holds the compressed version. The two play points of the two separate buffers are held on the same frame throughout the download of the entire video. When the user initiates a discontinuous action, the play point of the normal buffer fetches the destination frame, if such frame does not exist, the user downloads the appropriate segment of the normal video. However, when a continuous action is performed, the play point renders the next frame in the interactive buffer. Once the continuous interaction resumes the loaders fetches the appropriate normal segments to match the content of the destination point.

7. HANDLING RECEIVER HETEROGENEITY

Stream sharing techniques such as broadcast and multicast enables large-scale deployment of VOD applications. However, clients of such applications might use different forms of end devices ranging from simple palmtop personal digital assistants (PDA), to powerful PCs and high-definition television (HDTV) as receptors. Delivering a uniform representation of a video does not take advantage of the high-bandwidth capabilities nor does it adapt to the low-bandwidth limitation of the receivers. A heterogeneous technique that can adapt to a range of receivers is critical to the overall quality of service (QoS) of the VOD system.

One solution to the heterogeneity problem is the use of *layered media formats*. The basic mechanism is to encode the video data as a series of layers; the lowest layer is called the *base layer* and higher layers are referred to as *enhancement layers* [5][21]. By delivering various layers in different multicast groups, a user can individually mould its service to fit its capacity, independently of other users. In the *Receiver-Driven Layered Multicast* technique [21], a user keeps adding layers until it is congested, then drops the higher layer. Hence, clients look for the optimal number of layers by trying to join and leave different multicast groups.

Another approach is the use of *bandwidth adaptors* [13] between the server and receivers. This technique provides means to adapt to the receiving capabilities within the periodic broadcast framework. The main role of the adaptor is the conversion of incoming video segments into segments suitable for broadcast in the downstream at a lower data rate. Since the video arrives repeatedly at the adaptor at a higher speed than the data rate of the broadcast initiated at the adaptor, an *As Late As Possible* caching policy is used to ensure that an incoming segment is cached only if it will not occur again before it is needed for the broadcast in the downstream. Furthermore, an *As Soon As Possible* policy is used to cast out any segment after its broadcast if this segment will occur again before it will be needed for a future broadcast in the downstream. The adaptor, thus, stores only what it needs from a video, but never the video in its entirety. In this environment, clients can be routed to the right adaptor according to their capabilities. The adaptor sending the video data to the clients becomes their server in a transparent fashion. Compared to techniques relying on multi-resolution encoding, a major advantage of the Adaptor approach is that clients with lesser capability can still enjoy the same video with no loss of quality.

A different technique, called *Heterogeneous Receiver-Oriented* (HeRO) Broadcasting [14], proposes a trade-off between waiting time and service delay in a periodic broadcast environment. HeRO is derived from the observation that even though a periodic broadcast might be designed for a particular client bandwidth, a client, depending on its arrival time, might actually need less bandwidth than what the broadcast scheme was intended for. In fact, clients can do this most of the time in a carefully designed broadcast environment. In HeRO, data fragmentation is based on the geometric series $[1, 2, 2^2, \ldots 2^{K-1}]$, where K is the number of channels allocated to the video. Each channel i periodically broadcasts the segment of size 2^{i-1}. Users with different capacities are constrained to start their download at some specific times. That is, users with high bandwidth can start the download at the next occurrence of the first segment; but others with less bandwidth must wait for some specific time to start the download. This technique is illustrated in Figure 12, where the server repeatedly broadcasts 4 segments on the first set of channels. The numbers at the top indicate the minimum number of loaders a client needs to have in order to start the download at the beginning of that specific time slot. Clients with less than this number of loaders will have to wait for a future slot. For instance, a client with two loaders can only start its download at the beginning of time slots 2, 3, 4, 6, or 7, etc. in order to see a continuous playback. This pattern repeats every broadcast period of the largest

segment. Two such periods are shown in Figure 12. To reduce service latency for less capable clients, HeRO provides the option to broadcast the longer segments on a second channel with a phase offset equal to half their size. The example in Figure 12 uses two such channels. We note that clients with only enough bandwidth for one or two loaders now have more possible slots to join the broadcast. For instance, a client with one loader can now join the HeRO broadcast at the beginning of slot 6. We observe that the HeRO approach, unlike the multi-resolution encoding techniques, does not reduce the playback quality of low-bandwidth clients.

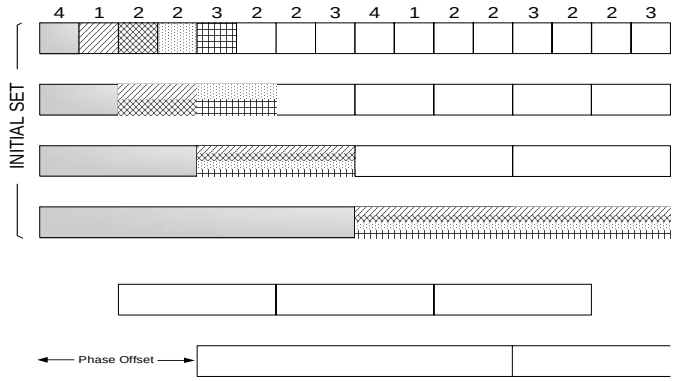

Figure 13. A Heterogeneous Receiver-Oriented (HeRO) broadcast example

8. SUMMARY

Video on demand is certainly a promising technology for many multimedia applications still to come. Unlike traditional data, delivery of a video clip can take up substantial bandwidth for a long period of time. This chapter describes some cost effective and scalable solutions for large-scale deployment of VOD systems.

A key aspect of designing a scalable VOD system is to leverage multicast and broadcast to facilitate bandwidth sharing. For popular videos or if the video database contains only few videos, a periodic broadcast technique achieves the best cost/performance. This approach, however, cannot deliver videos without some delay. Two alternatives have been considered for addressing this limitation: the *Server-Oriented Approach* (e.g., Skyscraper Broadcast, Striping Broadcast) reduces service delay by increasing server bandwidth; whereas the *Client-Oriented Approach* (e.g., *Cautious Harmonic Broadcast, Pagoda Broadcast*) requires client to equip with significantly more download bandwidth. For wide-area deployment, the *Server-Oriented Approach* is preferred because the cost of server bandwidth can be shared by a very large community of users, and therefore contributes little to the cost of the overall system. *Client-Oriented Approach* is limited to local deployment on an intranet where client bandwidth is abundant (e.g., receiving the video at five times the playback rate). Applications such as corporate training, campus information systems can benefit from these techniques.

For less popular videos, multicast on demand is a better solution then repeatedly broadcasting the videos. Standard multicast, however, makes users wait for the batching period. *Patching* resolves this problem by allowing the clients to join an ongoing multicast while playing back the missing part of the video arriving in a patching stream. For the best performance, a system should use both periodic broadcast and multicast. Techniques, such as AHA, can be used to monitor the popularity of the videos, and apply the best mechanism to deliver them. Range Multicast is a new concept in multicast. It enables clients to join a range multicast at their specified time, and still receive the entire video stream. Since many users actually receive their videos from the network, this new communication paradigm enables the VOD system to scale far beyond the physical limitation of the video server.

VCR-like interaction is a desirable feature for many VOD applications. It provides a convenient environment to browse and search for video content. Techniques such as *Split and Merge* can be used to provide such operations under multicast. For periodic broadcast, the *broadcast-based interaction* technique offers a highly scalable solution. In fact, the amount of server bandwidth required to support interactivity is independent of the number of users currently using this service.

Another important consideration in designing VOD systems is the capability to handle receiver heterogeneity. Multi-resolution encoding techniques provide a good solution for many applications. For those that demand the same high QoS for clients of various capabilities, techniques such as *Bandwidth Adaptor* and HeRO can be used in the periodic broadcast framework. A bandwidth adaptor, as the name implies, receives broadcast data from upstream at a high speed, and broadcast them to the downstream at a lower rate. Such a device can be placed on the server side to service a wide area of users, or on the client side to support a small group of users. HeRO is a different approach that handles differences in receiving bandwidths using only a single broadcast scheme. This is achieved by indicating in the broadcast when a client with a particular receiving capability can start its download. This solution is simple and effective.

References

[1] E. L Abram Profeta and K.G. Shin, “Providing unrestricted VCR functions in multicast video-on-demand servers,” in IEEE Int'l. Conf. on Multimedia Computing Systems (ICMCS'98), Austin, Texas, 1998.

[2] C.C Agarwal, J. L. Wolf, and P.S. Yu, “On Optimal batching policies for video on demand storage servers,” in Proc. of IEEE ICMCS'96, pp.253-258, Jun. 1996.

[3] K. C. Almeroth and M.H Ammar, “The use of multicast delivery to provide a scalable and interactive video-on-demand service,” in IEEE Journal of Selected Areas in Communications, vol 14, Aug. 1996.

[4] K.C. Almeroth and M. H. Ammar, “A scalable interactive video-on-demand service using multicast communication,” In Proc. of Int'l Conf. on Computer Communication Networks, pp. 292-301,1994.

[5] H. M. Briceo, S. Gortler, and L. McMillan, “Naïve-network aware internet video encoding,” in proc. of the 7th ACM Int’l Multimedia Conf., pp. 251-260, Oct. 1999

[6] Ying Cai and K. A. Hua, “Optimizing patching performance,” in Proc. of SPIE's Conf. on Multimedia Computing and Networking (MMCN'99), pp. 204-246, Jan. 1999.

[7] S. W. Carter and D.D.E. Long, “Improving bandwidth efficiency of video-on-demand servers,” Computer Networks and ISDN Systems, 31(1): 99-111, Mar. 1999.

[8] Asit Dan, Dinkar Sitaram, and Pervez Shahabuddin, “Scheduling policies for and on-demand video server with batching,” in Proc, of ACM Multimedia Conference, Oct. 1994.

[9] Asit Dan, Perwez Shahabuddin, Dinkar Sitaram, and Dow Towsley, “Channel Allocation under batching and VCR control in video-on-demand systems,” Journal of Parallel and Distributed Computing, 30(2): 168-79, Nov. 1995.

[10] A. Dan, D. Sitaram, and P. Shahabuddin, “Scheduling policies for on demand video server. Multimedia Systems,” 4(3): 112-121, Jun. 1996.

[11] Z. Fei, I. Kamel, S. Mukherjee, and M. Ammar, “Providing interactive functions through active client buffer management in partitioned video broadcast,” in Proc. of 1st Int’l workshop on networked group communication, (NGC'99), Nov. 1999

[12] Z. Fei, I. Kamal, S. Mukherje, M. Ammar, “Providing interactive functions for staggered multicast near video-on-demand systems,” Proc of the IEEE Int’l Conf on Multim Computing Systems, Jun. 1999

[13] K. A. Hua, Olivier Bagouet, and David Oger, “Bandwidth adaptors for heterogeneous broadcast-based video-on-demand systems,” Technical report #.

[14] K. A. Hua, Olivier Bagouet, and David Oger, “A periodic broadcast protocol for heterogeneous receivers,” SPIE Conf. On Multimedia Computing and Networking. Jan. 2003.

[15] K. A. Hua, Y. Cai, and Simon Sheu. Patching, “A multicast technique for true Video on Demand services,” in Proc. of ACM Multimedia Conf., Sept. 1998.

[16] K. A. Hua, Y. Cai, and S. Sheu, “Exploiting client bandwidth for more efficient video broadcast,” in Proc. of the Int’l Conf. on computer communication networks 1998.

[17] Kien A. Hua, J-H Oh and K. Vu, ``An Adaptive Video Multicast Scheme for Varying Workloads,'' *ACM-Springer Multimedia Systems Journal*, Vol. 8, Issue 4, August 2002, pp. 258-269.

[18] Kien A. Hua and Simon Sheu, “Skyscraper Broadcasting: a new broadcasting scheme for metropolitan video-on-demand systems,” in Proc. of SIGCOMM 97, pp. 89-100, Sept. 1997.

[19] Kien A. Hua, Duc A. Tran, and Roy Villafane, “Caching Multicast Protocol for On-Demand Video Delivery,” in the Proc. of ACM/SPIE Conf. on Multimedia Computing and Networking (MMCN 2000), pp. 2-13, Jan. 2000.

[20] W. Liao and V. O. Li, “The split and merge (SAM) protocol for interactive video on demand,” in IEEE multimedia, vol. 4 pp.51-62, Oct.-Dec. 1997.

[21] S. McCanne, V. Jacobson, and M. Vetterli, “Receiver-driven layered multicast,” ACM SiGCOMM'96, Aug. 1996.

[22] Jehan-Francois Paris, Steven W. Carter, and Darell D. E. Long, “Efficient broadcasting protocols for video-on-demand,” in Proc. of the int'l Symposium on Modeling, Analysis, and Simulation of Computing and Telecom Systems, pp. 127-32, Jul. 1998.

[23] Jehan-Francois Paris, “A simple low-bandwidth broadcasting protocol for video on demand,” in Proc. of the 8th Int'l Conf. on Computer Communications and Networks, 1998.

[24] S. Sen, L. Gao, J. Rexford, and Don Towsley, “Optimal patching schemes for efficient multimedia streaming,” in Proc. IEEE NOSSDAV'99, June 1999.

[25] Li-Shen Juhn and Li-Meng Tseng, “Harmonic Broadcasting for video-on-demand service,” IEEE Trans. on Broadcasting, 43(3): 268-71, Sept. 1997

[26] S. Sheu and K Hua,“Scalable Technologies for distributed multimedia systems” PhD Dissert. SEECS. Univ. of Central Florida. 1999.

[27] M. Tantaoui, K. A. Hua, S. Sheu, “Interaction in video broadcast,” in ACM Multimedia, Dec. 2002.