A Proactive Implementation of Interactive Video-on-Demand

Jehan-François Pâris
Department of Computer Science
University of Houston
Houston, TX 77204-3010

paris@cs.uh.edu

Darrell D. E. Long
Department of Computer Science
University of California
Santa Cruz, CA 95064

darrell@cse.ucsc.edu

Abstract

Most broadcasting protocols for video-on-demand do not allow the customer to pause, move fast-forward or backward while watching a video. We propose a broadcasting protocol implementing these features in a purely proactive fashion.

Our protocol implements rewind and pause interactions at the set-top box level by requiring the set-top box to keep in its buffer all video data it has received from the server until the customer has finished watching the video. It implements fast-forward by letting the video server transmit video data more frequently than needed by customers watching the video in sequence. As a result, any customer having watched the first x minutes of a video will be able to fast-forward to any scene within the first 2x or 3x minutes of the video. We show that this expanding horizon *feature can be provided at a reasonable cost.*

We also show how our protocol can accommodate customers connected to the service through a device lacking either the ability to receive data at more than two times the video consumption rate or the storage space required to store more than 20 to 25 percent of the video they are watching. While these customers will not have access to any of the interactive features provided by our protocol, they will be able to watch videos after the same wait time as all other customers.

1. INTRODUCTION

Broadcasting protocols offer the best solution for the successful deployment of metropolitan video-on-demand (VOD) services because they provide the most efficient way to distribute very popular videos to very large audiences and these so-called "hot" videos are expected to account for the majority of customer requests. Rather than reacting to individual viewer requests, broadcasting protocols distribute the contents of videos according to a fixed schedule guaranteeing that all customers will receive these contents on time. As a result, the number of customers watching a given video does not affect the server workload.

All recent VOD broadcasting protocols derive in some way from Viswanathan and Imielinski's *pyramid broadcasting* protocol [15]. Like it, they assume that most, if not all, users will watch each video in a strictly sequential fashion. These protocols also require customers to be connected to the service through a "smart" set-top box (STB) capable of (a) receiving data at rates exceeding the video consumption rate and (b) storing locally the video data that arrive out of sequence. In the current state of storage technology, this implies having a disk drive in each STB, a device already present in the so-called digital VCR's offered by TiVo [14], Replay [13] and Ultimate TV [14].

With the sole exception of staggered broadcasting, all broadcasting protocols share the common limitation of not offering any interactive action capability. Unlike VCRs, they do not provide controls allowing the viewers to pause the video and interrupt its viewing, to move fast-forward or backward (rewind). They require instead the viewers to watch each video in sequence as in a theater.

While staggered broadcasting provides some interactive control capability, it only allows viewers to jump from one staggered stream to another [1]. The sole advantage of this solution is its simplicity. Its major disadvantages are its high bandwidth requirements and its lack of precision: given a video of duration D distributed over k broadcasting channels, staggered broadcasting only allows users to move forward or backward in increments of D/k times units.

Two more recent works [4, 10] have proposed a better solution, namely adding interactive controls to an

[1] Supported in part by the Texas Advanced Research Program under grant 003652-0124-1999 and the National Science Foundation under grant CCR-9988390.

[2] Supported in part by the National Science Foundation under grant CCR-9988363.

existing broadcasting protocol and, preferably, to one having much lower bandwidth requirements than staggered broadcasting. Observe first that any efficient broadcasting protocol requires a disk drive in each customer STB. Today's cheapest disk drives have capacities of at least 10 gigabytes, giving them the possibility of storing at least three and a half hours of video in MPEG-2 format. One of the authors [10] proposed to keep in the customer STB all video data until the customer has watched the entire video. This would allow the STB to handle locally all pause and rewind commands while contingent streams would transmit on demand the missing video data. Hu [4] proposed to broadcast each video segment at a period that is x time units less than their maximum broadcasting period in order to allow fast forwards of up to x time units.

Both proposals have their disadvantages. Using contingent streams introduces a reactive component in the video server, complicating its design and making the whole scheme less scalable. Decreasing by a fixed quantity the broadcasting period of all segments could be quite costly unless we settle for a small decrement and a small fast forward horizon.

Our proposal does not suffer from these limitations. Like Hu's proposal, it is entirely proactive and does not require contingent streams. Our major difference is that we decrease the broadcasting period of all segments by a constant factor to allow any customer having watched the first x minutes of a video to fast forward to any scene within the first $2x$ or $3x$ minutes of the video. This *expanding horizon* approach, as we would like to call it, offers two major advantages. First, it provides users with a fast-forward horizon that will quickly exceed that provided by a protocol using a fixed horizon. Second, it is cheaper to implement.

The remainder of the paper is organized as follows. Section 2 reviews relevant previous work on broadcasting protocols. Section 3 presents a theoretical analysis of our approach. Section 4 presents a fixed-delay broadcasting protocol allowing fast forward to an expanding horizon and discusses its advantages and disadvantages. Finally, Section 5 has our conclusions.

2. PREVIOUS WORK

Given the large number of video broadcasting protocols that have been proposed since Viswanathan and Imielinski's *pyramid broadcasting* protocol, we will only mention those protocols that are directly relevant to our work. The reader interested in a more comprehensive review of broadcasting protocols for video-on-demand may want to consult reference [2].

First Channel	S_1	S_1	S_1	S_1	S_1	S_1
Second Channel	S_2	S_3	S_2	S_3	S_2	S_3
Third Channel	S_4	S_5	S_6	S_7	S_4	S_5

Figure 1. The first three channels for the FB protocol

The simplest broadcasting protocol is Juhn and Tseng's *fast broadcasting* (FB) protocol [5]. The FB protocol allocates to each video k data channels whose bandwidths are all equal to the video consumption rate b. It then partitions each video into 2^{k-1} segments, S_1 to $S_{2^{k-1}}$, of equal duration d. As Figure 1 indicates, the first channel continuously rebroadcasts segment S_1, the second channel transmits segments S_2 and S_3, and the third channel transmits segments S_4 to S_7. More generally, channel j with $1 \le j \le k$ transmits segments $S_{2^{j-1}}$ to S_{2^j-1}.

When customers want to watch a video, they wait until the beginning of the next transmission of segment S_1. They then start watching that segment while their STB starts downloading data from all other channels. Hence the maximum customer waiting time is equal to the duration of a segment. Define a *slot* as a time interval equal to the duration of a segment. To prove the correctness of the FB protocol, we need only to observe that each segment S_i with $1 \le i \le 2^{k-1}$ is rebroadcast at least once every i slot. Then any client STB starting to receive data from all broadcasting channels will always receive all segments on time.

The FB protocol does not require customer STBs to wait for any minimum amount of time. As a result, there is no point in requiring customer STBs to start downloading data while customers are still waiting for the beginning of the video. The newer *fixed-delay pagoda broadcasting* (FDPB) protocol [11] requires all users to wait for a fixed delay w before watching the video they have selected. This waiting time is normally a multiple m of the segment duration d. As a result, the FDPB protocol can partition each video into much smaller segments than FB with the same number of channels. Since these smaller segments can be packed much more effectively into the k channels assigned to the video, the FDPB protocol achieves smaller customer waiting times than an FB protocol with the same number of channels.

Table I summarizes the segment-to-channel mappings of a FDPB protocol requiring customers to wait for exactly 9 times the duration of a segment. Since customers have to wait for 9 times that duration, the first segment of the video will need to be broadcast at least once every 9 slots. Hence the protocol will use time division multiplexing to partition the first channel into $\sqrt{9}$ subchannels with each subchannel containing one third of the slots of the

Table I. The first five channels for a FDPB protocol with $m = 9$

Channel	*Subchannel*	*First Segment*	*Last Segment*
C_1	1	S_1	S_3
	2	S_4	S_7
	3	S_8	S_{12}
C_2	All 5 subchannels	S_{13}	S_{42}
C_3	All 7 subchannels	S_{43}	S_{116}
C_4	All 11 subchannels	S_{117}	S_{308}
C_5	All 17 subchannels	S_{309}	S_{814}

channel. The first subchannel will continuously broadcast segments S_1 to S_3 ensuring that these segments are repeated exactly once every 9 slots.

Observe that the next segment to be broadcast, segment S_4 needs to be broadcast once every 12 slots. Hence the second subchannel will transmit segments S_4 to S_7 ensuring that these segments are repeated exactly once every 12 slots. In the same way, the third subchannel will broadcast segments S_8 to S_{12} ensuring that these segments are repeated exactly once every 15 slots.

The process will be repeated for each of the following channels partitioning each channel into a number of subchannels close to the square root of the minimum periodicity of the lowest numbered segment to be broadcast by the channel. Hence channel C_2 will be partitioned into 5 subchannels because segment S_{13} needs to be repeated every 21 slots and $5 \approx \sqrt{21}$. As a result, the protocol will map segments S_{13} to S_{42} into the 5 subchannels of the second channel. Repeating the same process on channels C_3 to C_5, the protocol will be able to map 814 segments into five channels and achieve a deterministic waiting time of 9/814 of the duration of the video, that is, 80 seconds for a two-hour video.

Most research on interactive video-on-demand has focused on reactive video distribution protocols. Li *et al.* proposed in 1996 to use contingent streams to handle interactive VOD operations [7]. More recent work has focused on minimizing the duration of these contingent streams by merging them as soon as possible with other streams [3, 6, 8, 9]. Poon *et al.* have proposed a single-rate multicast double-rate unicast protocol supporting full VCR functionality [12].

3. THEORETICAL ANALYSIS

In this section, we derive lower bounds for the bandwidth requirements of fixed-delay broadcasting protocols allowing a limited amount of fast forwarding.

To compute these lower bounds, let us consider first the case of a broadcasting protocol not allowing any fast forwarding. Let D represent the duration of the video and w the duration of the fixed delay all customers must wait for before starting to watch the video. Consider t a small time interval Δt starting at an offset t within the video. To avoid STB underflow, the contents of this time interval must be broadcast at a minimum bandwidth $b/(t+w)$ where b is the video consumption rate.

Summing over all intervals as Δt approaches 0, we see that the bandwidth required to transmit the video is given by:

$$\int_0^D \frac{b}{t+w}\,dt = b(\ln(D+w) - \ln w) = b \ln \frac{D+w}{w} \quad (1)$$

Assume now that the protocol allows customers to fast-forward up to x time units ahead of their current position. The contents of each small interval Δt starting at a location t within the video will have to be broadcast at a minimum bandwidth $b/(t+w-x)$. The minimum bandwidth required to transmit the video is now given by:

$$\int_0^D \frac{b}{t+w-x}\,dt = b(\ln(D+w-x) - \ln w) = b \ln \frac{D+w-x}{w-x}$$

Observe that x cannot be greater than or equal to w and that the minimum bandwidth required to broadcast the video goes to infinity when x approaches w. Given that we expect w to be of the order of a few minutes, we can see that no broadcasting protocol will ever be able to provide a fixed fast forward horizon of any significant duration.

Consider now a broadcasting protocol allowing customers who have already watched the first x minutes of a video to fast forward to any scene within the first fx minutes of the video. The contents of each small interval Δt starting at an offset t within the video will have to be broadcast at a minimum bandwidth

$$\frac{b}{\frac{t}{f}+w} = \frac{bf}{t+fw}.$$

The minimum bandwidth required to transmit the video is now given by:

$$\int_0^D \frac{fb}{t+fw}\,dt = fb(\ln(D+fw) - \ln fw) = fb \ln \frac{D/f+w}{w} \quad (2)$$

Slots	1	2	3	4	5	6	7	8	9	10	11	12
First subchannel	S_1			S_2			S_3			...		
Second subchannel		S_4			S_5			S_6			...	
Third subchannel			S_7			S_8			S_9			S_{10}

Figure 2. How the protocol maps its first channel

In essence, broadcasting a video of duration D in a way that allows customers who have already watched the first x minutes of a video to fast forward to any scene within the first fx minutes of the video requires the same bandwidth as broadcasting a video of duration D/f with a video consumption rate fb.

Subtracting equation (1) from equation (2), we obtain the overhead of implementing a fast forward horizon growing at a rate f:

$$fb\ln\frac{D/f+w}{w} - b\ln\frac{D+w}{w} = b\ln\frac{(D/f+w)^f}{w^{f-1}(D+w)} < (f-1)b\ln\frac{D/f+w}{w}$$

This overhead will always be less than $f-1$ times the bandwidth required to broadcast a video of duration D/f with a video consumption rate b. This result is not as strong as it appears because the minimum bandwidth required to broadcast a given video is not that sensitive to the duration of that video. Consider, for instance a one-hour video and let us assume that we want a customer delay of 4 minutes. The minimum bandwidth required to broadcast this video will be given by $b\ln(64/4)$, that is, 2.77 times the video consumption rate. Broadcasting a two-hour video with the same customer delay would require a bandwidth equal to $b\ln(124/4)$, that is, 3.43 times the video consumption rate.

One way to decrease the cost of implementing fast-forward with an expanding horizon would be to disallow fast forward during the first few minutes of the video. This would allow us to broadcast the first few segments of the video at their normal frequency instead of at a multiple f of that frequency. The savings could be considerable as the first segments of a video are the ones that require the most bandwidth. Conversely, a broadcasting protocol with a fast forward horizon expanding at an increasing rate as the customer watches the video could be implemented at a reasonable additional cost as the later segments of a video require much less bandwidth as its first segments.

4. IMPLEMENTATION

We present in this section a broadcasting protocol allowing customers who have watched the first x minutes of a video to fast forward to any scene within the first $2x$ minutes of the video. In other words, its fast forward horizon will expand at a fixed rate $f = 2$. We decided to base our protocol on the fixed-delay pagoda broadcasting (FDPB) protocol discussed in section 2 because it has bandwidth requirements that are fairly close to the theoretical minimum.

We will consider a video of duration D to be broadcast over k channels C_j with $1 \le j \le k$. The bandwidths of these k channels will all be equal to the video consumption rate b. The protocol will partition each video into n equal-size segments of duration $d = D/n$. These n segments will be broadcast at different frequencies over the k channels, each segment transmission occupying a *slot* of duration d. The broadcast schedule will allow customers who have been watching the video for at least x minutes to fast forward to any scene within the first fx minutes of that video.

As in the example of section 3, we will assume $m = 9$, which means that each customer will have to wait for a time equal to the duration of 9 videos.

Consider segment S_1, that is, the first segment of the video. To guarantee its on-time arrival, it needs to be broadcast at least once every m slots. This will also allow any kind of fast forwarding within that segment. The next segment, segment S_2, must become accessible as soon as customers have finished watching the first half of segment S_1. Hence it will also need to be broadcast at least once every 9 slots. The following segment is segment S_3. It must become accessible as soon as customers have finished watching segment S_1 and will need to be broadcast at least once every 10 slots. Segment S_4 will need to be broadcast at the same frequency as segment S_3 since it must become accessible as soon as customers have finished watching the first half of segment S_2. More generally segment S_i with $1 \le i \le n$ will need to be broadcast at least once every $m + \lceil i/f - 1 \rceil$ slots, which will guarantee that customers will be able to fast forward to it as soon as they have watched the first i/f segments of the video.

Table II. The first eight channels for a FDPB protocol with $m = 9$ and a fast forward horizon expanding at a rate $f = 2$

Channel	Subchannel	First Segment	Last Segment
C_1	1	S_1	S_3
	2	S_4	S_6
	3	S_7	S_{10}
C_2	1	S_{11}	S_{17}
	2	S_{18}	S_{25}
C_3	1	S_{26}	S_{32}
	2	S_{33}	S_{40}
	3	S_{41}	S_{49}
C_4	1	S_{50}	S_{60}
	2	S_{61}	S_{73}
	3	S_{74}	S_{88}
C_5	All 6 subchannels	S_{89}	S_{151}
C_6	All 6 subchannels	S_{152}	S_{252}
C_7	All 7 subchannels	S_{253}	S_{417}
C_8	All 12 subchannels	S_{418}	S_{688}

As the original FDPB protocol, our protocol will partition each channel C_j into s_j *subchannels* in such a way that each subchannel will occupy $1/s_j$ of the slots of channel C_j. Looking at Figure 2, we see that the first channel is partitioned into 3 subchannels. The first of these subchannels broadcasts segments S_1 to S_3 ensuring that these segments will be repeated once every 9 slots. The first segment to be broadcast by the second subchannel is segment S_4, which needs to be broadcast every ten slots. Since the second subchannel has only one-third of the slots of its channel, it can only broadcast segments at periods that are multiples of three of the slot size. Hence it will broadcast segments S_4 to S_6 once every 9 slots. The first segment to be broadcast by the third subchannel is segment S_7, which needs to be broadcast once every 12 slots. As a result, the third subchannel will broadcast segments S_7 to S_{10}.

The first segment to be broadcast by the second channel is thus segment S_{11}, which needs to be broadcast every $9 + 5 = 14$ slots. Recall that the original FDPB protocol partitioned each channel into a number of subchannels close to the square root of the minimum period of all segments to be broadcast by this channel. Our new protocol uses a slightly different approach: the number of subchannels into which a channel will be partitioned is obtained by considering all possible values of s_k and selecting the one mapping the most segments into the channel. As a result, the second channel will be partitioned into two subchannels, one broadcasting segments S_{11} to S_{17} and the other segments S_{18} to S_{25}.

Table II summarizes the final segment-to-channel mappings for the first 8 channels. As one can see allocating 8 channels to a video allows us to partition the video into 688 segments and achieve a waiting time of 9/688 of the video duration, that is 94 seconds for a two-hour video. Recall that the original FDPB protocol with the same value of m only needed 5 channels to achieve a maximum waiting time of 80 seconds for the same two-hour. Hence allowing customers who have watched the first x minutes of a video to fast forward to any scene within the first $2x$ minutes of the video will require three extra channels.

We believe that broadcasting these three additional channels will require less computing and networking resources than implementing contingent streams with the client/server interactions these streams would require.

There are two additional benefits in implementing fast forward by transmitting more frequently video segments. First, we observe that most high-numbered segments will be transmitted twice to between the time the customers order the video and the time they actually watch that segment. Hence a STB receiving the first time a damaged segment would have a second chance to receive a working segment. Second, transmitting segments more frequently would also help customers who are connected to the service through a device lacking either the ability to receive data at more than two times the video consumption rate or the storage space required to store 40 to 50 percent of the video they are watching.

Limiting the Client Bandwidth to Two Channels

Let us show first how our protocol can handle customers connected to the video-on-demand service through a device that cannot receive data at more than twice the video consumption rate. Our protocol will let these customers watch videos after the same wait time as all other customers but will not let them fast-forward.

We will always start counting slots from the time customers order the video. Hence we will say that segment S_1 will need to be in the customer STB by the end of the 9^{th} slot and, more generally that segment S_i will need to be in the customer STB by the end of the $i + 8^{th}$ slot.

Recall that our protocol transmits all segments—but the first one—at a higher frequency than the original FDPB protocol. Hence the STB of a customer not interested in the fast forward feature will not need to receive data from all channels at the same time. Consider, for instance, the case of the second channel. As Tables II and III show, channel C_2 broadcasts segments S_{11} to S_{25}. Segment S_{11} is repeated every 14 slots and segment S_{25} is repeated every 16 slots. Note that segment S_{11} must reach

Table III. Minimum and maximum periodicities of the segments broadcast by a FDPB protocol with $m = 9$ and a fast-forward horizon expanding at a rate $f = 2$

Channel	*First Segment*	*Last Segment*	*Shortest Period (slots)*	*Longest Period (slots)*
C_1	S_1	S_9	9	12
C_2	S_{11}	S_{25}	14	16
C_3	S_{26}	S_{49}	21	27
C_4	S_{50}	S_{49}	33	45
C_5	S_{89}	S_{151}	48	78
C_6	S_{152}	S_{252}	84	120
C_7	S_{253}	S_{417}	133	203
C_8	S_{418}	S_{688}	216	336

the customer STB by the time the customer has finished watching segment S_{10}, that is, by the end of the 19th slot. Hence the customer STB can wait for 5 slots before starting to receive data from channel C_2. Since segment S_{25} is repeated every 16 slots, the STB will then stop receiving data from channel C_2 after 5 + 16 = 21 slots.

Consider now channel C_3. It broadcasts segments S_{26} to S_{25}. Segment S_{26} is repeated every 21 slots and needs to reach the customer STB before the end of the 34th slot. Hence it can be safely delayed by 13 slots, that is, after the STB will have received the first 10 segments of the video from channel C_1. Since segment S_{49} is repeated every 27 slots, the STB will then stop receiving data from channel C_2 after 13 + 27 = 21 slots.

We can apply the same reasoning to all successive channels and we will see the STB will never have to start receiving data from channel C_{i+2} before it has finished receiving data from channel C_i. Hence the STB of a customer not interested in fast-forwarding through the video will never have to receive data from the video server on more than two channels at a time.

Reducing the Client Buffer Size Requirements

Delaying segment reception from successive channels will also impact the maximum amount of video data to be stored in the STB buffer. As we stated earlier the STB of a customer wanting to experience the interactive features provided by our protocol will require a buffer capable of storing each video being watched in its entirety.

Disabling these interactive features and delaying as much as possible segment reception will result in much lower storage requirements because the STB will not have to keep in its buffer any segment that has been viewed by the customer. As a result, the number of segments kept in the STB buffer will reach its maximum when the STB finishes receiving data from the next to last channel. This number will remain constant as long as the STB receives data from all the subchannels of the last channel and will then start decreasing after that.

As shown in Table III, the first subchannel of any channel has always the shortest period of any subchannel in that channel. We can thus estimate the minimum storage requirements of our protocol by measuring the number of segments in the buffer when the STB has just finished receiving data from the first subchannel of the last channel.

Assume that the last channel is channel C_k and that it contains s_k subchannels. Let then S_z be the first segment to be broadcast by channel C_k. Since S_z must be repeated at least once every $m + \lceil z/f - 1 \rceil$ slots, the first subchannel of C_k will contain exactly $\lfloor (m + \lceil z/f - 1 \rceil)/s_k \rfloor$ segments. By the time the STB will finish receiving data from that subchannel, it will have in its buffer a total of $s_k \times \lfloor (m + \lceil z/f - 1 \rceil)/s_k \rfloor$ segments from all s_k subchannels of the last channel. Looking at Table III, we see that the first segments of the last channel are repeated once every 266 slots. Hence the STB will stop receiving data from the first subchannel of the last channel after having received 216 segments from that channel. The STB will thus never hold more than 216 of the 688 of the segments constituting the video, that is, 31.4 percent of the video. More generally, the STB will never have to hold more than 32 percent of the video when the video is broadcast on 7 or more channels.

Further reductions in client buffer size requirements could be achieved by limiting the number of segments that can be broadcast by any channel. If no channel broadcasts more than n_{max} distinct segments, each segment will be repeated at least once every n_{max} slot and the customer STB will never have to store more than n_{max} segments.

Consider, for instance a variant of our protocol not allowing any channel to broadcast more than 100 channels. Channels C_6 to C_8 would now only broadcast 100 channels each. Assuming the same values of the m and f parameters, the protocol would only be able to broadcast 451 segments over 8 channels. As a result it would only achieve a waiting time of 144 seconds but would require the customer STB to store less than 100/451 or 22.2 percent of the video.

5. CONCLUSION

Broadcasting protocols for video-on-demand typically require customers to watch videos in sequence and do not

allow them to pause, move fast-forward or backward while watching a video.

We have presented a pagoda broadcasting protocol overcoming these limitations without requiring contingent streams. Hence it does not suffer the same scalability limitations as protocols involving contingent streams. Our protocol implements rewind and pause interactions by requiring the set-top box to keep in its buffer all video data it has received from the server until the customer has finished watching the video. It implements fast-forward by letting the video server transmit video data more frequently than needed by customers watching the video in sequence. As a result, any customer having watched the first x minutes of a video will be able to fast-forward to any scene within the first $2x$ or $3x$ minutes of the video. As we have seen, this expanding horizon feature can be provided at the cost of three additional channels per video.

We have also shown how our protocol can accommodate customers connected to the service through a device lacking either the ability to receive data at more than two times the video consumption rate or the storage space required to store more than 20 to 25 percent of the video they are watching. While these customers will not have access to any of the interactive features provided by our protocol, they will be able to watch videos after the same wait time as all other customers.

REFERENCES

[1] K. C. Almeroth and M. H. Ammar, The use of multicast delivery to provide a scalable and interactive video-on-demand service. *IEEE Journal on Selected Areas in Communications*, 14(50):1110–1122, Aug. 1996

[2] S. W. Carter, D. D. E Long and J.-F. Pâris, Video-on-demand broadcasting protocols, In *Multimedia Communications: Directions and Innovations* (J. D. Gibson, Ed.), Academic Press, San Diego, 2000, pages 179–189.

[3] S. W. Carter, D. D. E Long and J.-F. Pâris, An efficient implementation of interactive video-on-demand, *Proc. 8th International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems*, pages 172–179, San Francisco, CA, Aug.-Sept. 2000.

[4] A. Hu, Video-on-demand broadcasting protocols: a comprehensive study. *Proc. IEEE INFOCOM 2001*, Vol. 1, pages 508–517, Anchorage, AK, April 2001.

[5] L. Juhn and L. Tseng. Fast data broadcasting and receiving scheme for popular video service. *IEEE Transactions on Broadcasting*, 44(1):100–105, March 1998.

[6] N. Kamiyama and V. O. K. Li, An efficient deterministic bandwidth allocation method in interactive video-on-demand systems. *Proc. 1998 Global Communication Conference*, vol. 2, pages 664–671, Nov. 1998.

[7] V. O. K. Li , W. Liao , X. Qiu , and E. Wang, ."Performance model of interactive video-on-demand systems. IEEE *Journal on Selected Areas in Communications*, 14(6):1099–1109, Aug1996.

[8] W. Liao and V. O. K. Li, The split-and-merge (SAM) protocol for interactive video-on-demand systems. *IEEE Multimedia* 4(4): 51–62, 1997.

[9] W. Liao, V. O. K. Li: Synchronization of distributed multimedia systems with user interactions, *Multimedia Systems* 6(3): 196–205, 1998.

[10] J.-F. Pâris, An interactive broadcasting protocol for video-on-demand, *Proc. 20th International Performance of Computers and Communication Conference* (IPCCC '01), pages 347–353,Phoenix, AZ, April 2001.

[11] J.-F. Pâris. A fixed-delay broadcasting protocol for video-on-demand, *Proc. 10th International Conference on Computer Communications and Networks* (ICCCN '01), pages 418–423, Scottsdale, AZ, Oct. 2001.

[12] W.-F. Poon, K.-T. Lo and J. Feng, "Design and analysis of multicast delivery to provide VCR functionality in video-on-demand systems," In *Proceedings of the 2nd International Conference on ATM*, pages 132–139, June 1999.

[13] ReplayTV. http://www.replay.com/.

[14] TiVo Technologies. http://www.tivo.com/.

[14] UltimateTV. http://www.ultimatetv.com/.

[15] S. Viswanathan and T. Imielinski. Metropolitan area video-on-demand service using pyramid broadcasting. *Multimedia Systems*, 4(4):197–208,