Using Previews to Reduce the Cost of Video-on-Demand Services

Jehan-François Pâris[1]

Department of Computer Science, University of Houston,
Houston, TX 77204-3010
paris@cs.uh.edu

Abstract. We propose to start each video-on-demand program by two or three short previews of coming attractions as it is customarily done in movie theaters and on videocassettes. These previews will be cheap to distribute for they can be viewed in any order. As we will show, they can significantly reduce the bandwidth requirements of most video distribution protocols, among which, stream tapping and most static and dynamic video broadcasting protocols. The sole exception seems to be fixed-delay broadcasting protocols that limit the receiving bandwidth of the customers' set-top boxes to two channels. In that case, the bandwidth used for distributing the previews temporarily reduces the bandwidth that remains available to the fixed-delay broadcasting protocol, which results in a significant increase of the server bandwidth required to achieve a given customer waiting time.

1 Introduction

When we go watch a movie in a theater, when we put a videocassette in our VCR, we are to watch first a few short previews of coming attractions. The main purpose of these previews is to entice us either to come back to the theater for another movie or to rent more videocassettes.

We think that video-on-demand (VOD) services should follow the same approach and start their programs by two to three short previews. First, it would allow them to advertise their current and future offerings. Second, it could significantly reduce the bandwidth requirements of the service.

This is an important issue because distributing videos on demand necessitates vast amounts of bandwidth and these high bandwidth requirements are the main reason for the very high cost of VOD services. Reducing this cost is an imperative because VOD has to compete against cheaper well-established rivals such as videocassette rentals or pay-per-view.

[1] Supported in part by the National Science Foundation under grant CCR-9988390

This situation has resulted in numerous proposals aiming at reducing the bandwidth requirements of VOD services. All these proposals can be broadly classified into three main categories. All proposals in the first group follow a strict *proactive* approach. They partition each video into *segments* and retransmit theses segments according to a fixed schedule guaranteeing that any customer having waited for a given maximum delay *w* will be able to watch the whole video without any interruption. The simplest proactive protocol is *staggered broadcasting.* It consists of broadcasting each video on *k* distinct channels with each broadcast starting an offset *D/k*, where *D* is the duration of the video. More sophisticated proactive protocols, such as *pyramid broadcasting* [17], can achieve shorter customer waiting times at much lower bandwidth costs. These protocols require customer set-top boxes that can receive video data at at least twice the video consumption rate and have enough local storage to store up to one half of the video duration.

Proactive protocols work well for videos that are in high demand such as the ten to twenty videos that are "hot" at any time. Reactive protocols take a less radical approach and do not require the video server to transmit any data in the absence of any request for the video. They try however to share as many data as possible among overlapping requests for the same video. Some of the earliest reactive solutions include *batching* [5] and *piggybacking* [8]. More recent proposals such as *stream tapping* [2, 3] or *hierarchical multicast stream merging* [6] save more bandwidth but also require customer set-top boxes that can receive video data at at least twice the video consumption rate and have enough local storage to store at least ten to twenty minutes of video.

A third group of solutions are reactive in nature but behave like proactive solutions once the customer request arrival rate reaches some saturation point [11, 18]. They work best for videos whose popularity varies over time.

As we will see, all three approaches can benefit from starting each video program with a few minutes of previews. Previews are cheap to distribute for they can be viewed in any order. They also provide the customer with something interesting to watch while waiting for the beginning of the video. This would allow us to stretch this delay and reduce the bandwidth requirements of nearly every video distribution protocol. We need however to qualify this statement: some broadcasting protocols require the customer set-top box to start accumulating video data while the customer is still waiting for the video. This is the case for the GEBB protocol [9] and the fixed-delay pagoda broadcasting protocol [15]. Whenever the receiving bandwidth of the customer set-top boxes is limited to two channels, the bandwidth used for distributing the previews will temporarily reduce the bandwidth that remains available to the protocol. This will result in a significant increase of the server bandwidth required to achieve a given customer waiting time.

The remainder of the paper is organized as follows. Section 2 reviews relevant previous work on video distribution protocols. Section 3 shows how to use previews to mask the latency of reactive protocols and reduce the bandwidth requirements of proactive protocols. Section 4 compares our approach with partial preloading. Finally, Section 5 has our conclusions.

Slots	1	2	3	4
First Channel	S_1	S_1	S_1	S_1
Second Channel	S_2	S_3	S_2	S_3
Third Channel	S_4	S_5	S_6	S_7

Figure 1. The first three channels for the fast broadcasting protocol

2 Previous Work

We will only mention those protocols that are the most relevant to our work. The reader interested in a more comprehensive review of proactive protocols may want to consult reference [4].

Proactive protocols

The simplest proactive protocol is Juhn and Tseng's *fast broadcasting* (FB) protocol [12]. It allocates to each video k 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 $1 \leq j \leq k$ 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 $1 \leq i \leq 2^k - 1$ 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 [15] 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 efficiently 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

Figure 2 displays the segment-to-channel mappings of a FDPB protocol requiring customers to wait for exactly 9 times the duration of a segment. Given that delay, the first

Subchannel	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1st	S_1	▨	▨	S_2	▨	▨	S_3	▨	▨	S_1	▨	▨	S_2	▨	▨
2nd	▨	S_4	▨	▨	S_5	▨	▨	S_6	▨	▨	S_7	▨	▨	S_4	▨
3rd	▨	▨	S_8	▨	▨	S_9	▨	▨	S_{10}	▨	▨	S_{11}	▨	▨	S_{12}

(a) First channel

Channel	*Subchannels*	*First Segment*	*Last Segment*
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 18 subchannels	S_{309}	S_{814}

(b) Channels C_2 to C_5

Figure 2. How the fixed-delay broadcasting protocol maps its first five channels.

segment of the video will need to be broadcast at least once every 9 slots. 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 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 equal 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. Applying 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.

Reactive protocols

Stream tapping [2, 3] or *patching* [10], assumes that each customer STB has a buffer capable of storing at least 10 minutes of video data. This buffer will allow the STB to

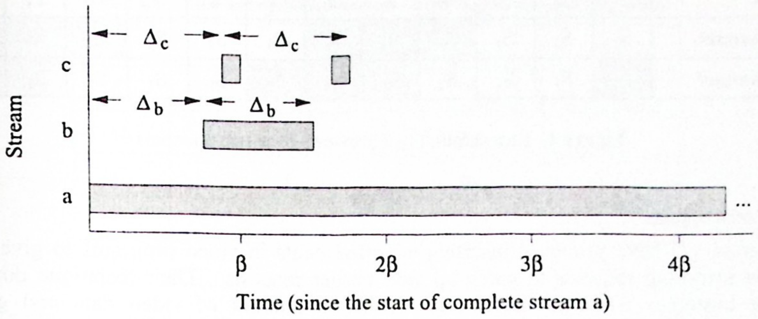

Figure 3. Stream tapping.

"tap" into streams of data on the VOD server originally created for other clients, and then store these data until they are needed. In the best case, clients can use most of the data from the existing stream, and greatly reduce the amount of time they need their own stream.

Stream tapping defines three types of streams. *Complete streams* read out a video in its entirely. These are the streams clients typically tap from. *Full tap streams* can be used if a complete stream for the same video started $\Delta \leq \beta$ minutes in the past, where β is the size of the client buffer, measured in minutes of video data. In this case, the client can begin receiving the complete stream right away, from its buffer, which will then always contain a moving Δ-minute window of the video. Stream tapping also defines partial tap streams, which can be used when $\Delta > \beta$. In this case clients must go through cycles of filling up and then emptying their buffer since the buffer is not large enough to account for the complete difference in video position. To use tap streams, clients only have to receive at most two streams at any one time. If they can actually handle a higher bandwidth than this, they can use an option to the protocol called extra tapping. Extra tapping allows clients to tap data from any stream on the VOD server, and not just from complete streams. Figure 3 shows some example streams from the VOD server's perspective. Stream *a* is a complete stream, and it must exist for the entirety of the video. Stream b is a full tap stream starting Δ_b minutes after stream a. It only has to exist for Δ_b minutes. Stream *c* is another full tap stream, but it is able to use extra tapping to tap data from stream *b*, and so its service time is much smaller than Δ_c.

Eager, Vernon and Zahorjan's *hierarchical multicast stream merging* (HMSM) protocol [6] is a reactive protocol that never requires the STB to receive more than two streams at the same time. *Selective catching* combines both reactive and proactive approaches. It dedicates a certain number of channels for periodic broadcasts of videos while using the other channels to allow incoming requests to catch up with the current broadcast cycle [7].

Slots	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	*11*	*12*
1st channel	S_1	S_2	S_3	S_1	S_2	S_3	S_1	S_2	S_3	S_1	S_2	S_3
2nd channel	S_1	S_1	S_1	S_2	S_1	S_1	S_2	S_1	S_1	S_2	S_1	S_1

Figure 4. Broadcasting two previews over two channels

Other techniques

Basu *et al.* [1] have proposed inserting advertisements in video programs to give more time to incoming requests to catch up with earlier requests. Their technique does not require customer STBs to buffer any significant amount of video data and greatly improves the performance of piggybacking protocols [8].

Partial preloading [13–14, 16] loads in the customer STB the first few minutes of the top 10 to 20 videos in order to provide zero-delay access to these videos and reduce the server bandwidth required by the remainder of the video. It applies to both proactive and reactive distribution protocols.

3 Implementation

We discuss first how previews can be distributed and show then how to use previews to mask the latency of a fixed-delay broadcasting protocols and reduce the bandwidth requirements of stream taping protocols.

Distributing previews

We will focus here on the simplest case where all customers watch the same previews. More targeted strategies are possible but they would also require more server bandwidth.

The cheapest way to distribute the previews is to broadcast them in sequence on a single channel. Since previews can be watched in any order, the average customer waiting time will be equal to one half of the average duration of a preview. A much better quality of service could be achieved by broadcasting the previews on two channels. As seen on Figure 4, adding an extra channel would allow us to partition each preview into three segments and achieve an average customer waiting time equal to one sixth of the average duration of a preview. Since most previews last around two minutes, this means an average waiting time of 20 seconds (and a maximum waiting time of 40 seconds).

Using previews to mask the latency of a fixed-delay broadcasting protocol

Fixed-delay broadcasting protocols, such as the GEBB protocol [9] and the FDPB protocol [15], could greatly benefit from our approach. Consider for instance the case of a FDPB protocol with $m = 9$. As we saw earlier, it requires 5 channels to achieve a waiting time of 40 seconds for a two-hour video. Assume now that each video is preceded by 4 minutes of previews. As shown on Figure 2, 4 channels suffice to partition the video into

Channel	*Number of Subchannels*	*First Segment*	*Last Segment*
C_2	3	S_1	S_{12}
C_2	5s	S_{13}	S_{42}
C_3	6	S_{43}	S_{95}
C_4	8	S_{96}	S_{193}
C_5	11	S_{193}	S_{369}
C_6	14	S_{370}	S_{674}

Figure 5. How the FDPB protocol maps its first six channels when the STB receiving bandwidth is limited to two channels.

308 segments and achieve a waiting time of 9/308 of the video duration. This waiting time would be completely masked by the 4 minutes of previews as long as the video lasts less than 4×308/9 = 137 minutes. Increasing the duration of the previews to five minutes would allow us to mask the latency of videos lasting up to 5×308/9 = 171 minutes, that is, almost three hours.

Hence we could reduce the bandwidth requirements of the FDPB by 20 percent by having customers watch between four and five minutes of previews before each video.

Handling set-top boxes that cannot receive more than two channels at the same time

Like most other broadcasting protocols, the FDBP protocol assumes that the STB can and will simultaneously receive data from all the channels on which the various segments of the video are broadcast. This requirement complicates the design of the STB and increases its cost.

Fortunately, the FDPB protocol can be easily modified to handle customers connected to the video-on-demand service through a STB that cannot receive data at more than twice the video consumption rate [15]. Consider the case of a FDPB protocol with $m = 9$ that interacts with STBs that cannot receive video data from more than two channels. As shown in Figure 5, the segment-to-channel mappings of the first two channels are unchanged. The first mappings to be affected are those of channel C_3 as the STB must now wait until it has received all data from the first channel before starting to receive data from channel C_3.

The last segment broadcast by the first channel is segment S_{12}. As shown on Figure 2, it is broadcast once every 15 slots. The first segment broadcast by channel C_3 is segment S_{43}. Recalling that the customer waiting time is equal to 9 slots, we see that segment S_{43} must now be broadcast at least once every 43 + 8 – 15 =36 slots. Similarly segment S_{44} has now to be broadcast at least once every 44 + 8 – 15 = 37 slots and so on. As a result, channel C_3 will now be partitioned into $\sqrt{36} = 6$ subchannels and will broadcast segments S_{43} to S_{95}. The process will be repeated on the remaining channels subject to the

Channel	*Number of Subchannels*	*First Segment*	*Last Segment*
C_2	3	S_1	S_{12}
C_2	3	S_{13}	S_{28}
C_3	5	S_{29}	S_{58}
C_4	6	S_{59}	S_{112}
C_5	9	S_{113}	S_{213}
C_6	12s	S_{214}	S_{391}

Figure 6. How the FDPB protocol maps its first six channels when the STB receiving bandwidth is limited to two channels and the customer watches previews while waiting for the video.

restriction that the STB cannot receive data from channel C_j for $j \geq 3$ until it has finished receiving data from channel C_{j-2}.

As we can see from Figure 5, the FDPB protocol will now need 6 channels to achieve a waiting time of 9/674 of the video duration, that is, 96 seconds for the same two-hour video.

Assume now that we want to mask this waiting time by having the customer watch one short preview. Since this preview will occupy one of the two channels the STB can listen to at any given time, the STB will not be able to receive data from channel C_2 until after the STB has finished receiving the previews. In our case, this means that the first segment broadcast by channel C_2, that is, segment S_{13} will now have to be repeated at least once every 12 slots instead of at least once every 12 + 8 = 20 slots. Consequently, channel C_2 will only be able to broadcast segments S_{13} to S_{28}, that is, 15 segments instead of the 30 segments it was previously broadcasting. This will result in much less efficient mappings for the each and every channel for all segments will now have to be repeated more frequently than before.

As we can see on Figure 6, having customers watching previews while waiting for the video of their choice has a dramatic impact on the performance of the FDPB protocol, which can only pack now 391 segments into six channels. As a result, the customer waiting time for a two-hour video will be equal to $7200 \times 9/391 = 203$ seconds.

We can thus characterize the benefits of having customers watching previews while waiting for the video of their choice as rather dubious. First and foremost, we will not save any bandwidth. Second, it is not clear that the customer will prefer waiting for almost three minutes while watching previews to waiting for slightly more than one minute and half in front of a blank screen.

Using previews to reduce the latency of a stream tapping protocol

Stream tapping and hierarchical multicast stream merging have two major advantages over broadcasting protocols. First, they provide true instant access to the video. Second, they require much less bandwidth than broadcasting protocols to distribute videos that are not requested more than ten times per hour for a two-hour video.

Unfortunately, the same is not true at higher request arrival rates. Since they handle all customer requests one by one, proactive protocols require much more bandwidth than most broadcasting protocols whenever the request arrival rate exceeds 20 requests per hour. One attractive solution would be to start each video program by four to five minutes of previews. This would have the same effect as batching requests together over a fixed batching interval equal to the duration of these previews and would greatly reduce the number of video streams that the server has to manage.

To estimate the impact of these previews on the protocol bandwidth requirements, we can observe that starting each video program by D_P minutes of previews is equivalent to limit the customer arrival rate to at most one arrival each D_P minutes. Previous simulation studies have shown that stream tapping with all its bandwidth savings options activated requires an average of 5.55 channels to satisfy 20 requests per hour for a two-hour video [14]. Hence starting each video with as little as three minutes of previews should allow us to satisfy any number of requests with an average bandwidth of 5.5 channels.

Increasing the durations of the previews to five to six minutes would result in further bandwidth reductions. For instance, starting each video with six minutes of previews would reduce the average bandwidth required to distribute a video to 4.09 channels, a figure quite similar to the bandwidth requirements of a FDPB protocol starting each video program with four minutes of previews.

Using previews with hybrid distribution protocols

These protocols, also known as dynamic broadcasting protocols, are reactive in nature but behave like proactive protocols once the customer request arrival rate reaches some saturation point [11, 18]. Hence, their bandwidth requirements are bounded, which distinguishes them from purely reactive protocols.

This distinction becomes blurred once we introduce previews. As we have just seen, adding previews to video programs puts an upper bound on the bandwidth requirements of a purely reactive protocol such as stream tapping. As a result, one of the motivations for using hybrid distribution protocols will disappear. We should focus instead on the actual performance of hybrid protocols all over the range of request arrival rates. An efficient hybrid protocol, such as the channel-based heuristic distribution protocol [18], would still perform as well as a stream tapping protocols at low request arrival rates while requiring significantly less bandwidth at high request arrival rates.

4 Comparison with Partial Preloading

Starting each video program by a few minutes of previews can have a dramatic impact on the bandwidth requirements of most video distribution protocols. In fact, it compares favorably with partial preloading [13–14, 16]. Starting each video program with x minutes of previews provides nearly the same results as having the first x minutes of each video preloaded in the customer STB.

Consider, for instance the case of the FDPB protocol. We have seen that having customers watch between four and five minutes of previews before each video would allow us to broadcast videos on four channels instead of five. This is not very different from what we could achieve by partitioning each video into 317 segments and by having the first 9 segments preloaded in the customer STB. As Figure 2 shows, the remaining 308 segments could then be broadcast on 4 channels. In both cases, we would have to factor the cost of the one or two additional channels required to broadcast either the previews or the initial segments of each video but the cost of these additional channels could be shared among all the videos being broadcast by the server.

There are however some important differences between the two techniques. First, prepending previews is much easier to implement for it does not require any negotiation with the customer STB. Second, applies to all videos while partial preloading only applies to a limited number of presumed popular videos.

On the other hand, partial preloading does not force customers to watch a few minutes of previews before watching the video of their choice. Hence the amount of previews we can force the customer to watch is quite limited.

Combining both techniques could be an attractive proposition. We do not believe it would make much sense to preload previews in the customers' STBs for it would not lead into any significant bandwidth improvements. A more attractive option would be to make partial preloading optional and have customers who do not have the first few minutes of the video preloaded on their STB watch more previews than those who have them.

5 Conclusion

Starting each video program by four to five minutes of previews would only require one to two additional video channels and could significantly reduce the bandwidth requirements of most video distribution protocols. In particular, it would reduce by 20 percent the bandwidth requirement of a fixed-delay pagoda protocol and would put an upper bound on the bandwidth requirements of stream taping protocols. Fixed-delay broadcasting protocols that limit the receiving bandwidth of the customer set-top boxes to two channels seem to be the sole exception. In their case, the bandwidth used for distributing the previews temporarily reduces the bandwidth that remains available to the fixed-delay broadcasting protocol, which results in a 20 to 25 percent increase of the server bandwidth required to achieve a given customer waiting time.

References

1. Basu, P., A. Narayanan, W. Ke, T. D. C. Little, and A. Bestavros. Optimal scheduling of secondary content for aggregation in video-on-demand systems. *Proc. 8th Int'l Conf. on Computer Communications and Networks*, pp. 104–109, Oct. 1999.

2. Carter, S. W. and D. D. E. Long. Improving video-on-demand server efficiency through stream tapping. *Proc. 6th Int'l Conf. on Computer Communications and Networks*, pp. 200–207, Sep. 1997.
3. Carter, S. W. and Darrell D. E. Long, Improving bandwidth efficiency of video-on-demand servers," *Computer Networks*, 31(1-2):99–111, 1999.
4. Carter, S. W., 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, pp. 179–189, 2000.
5. Dan, A., D. Sitaram, and P. Shahabuddin. Dynamic batching policies for an on-demand video server. *ACM Multimedia Systems Journal*, 4(3):112–121, June 1996.
6. Eager, D. L., M. K. Vernon and J. Zahorjan. Minimizing bandwidth requirements for on-demand data delivery. *IEEE Trans. on Knowledge and Data Engineering*, 13(5):742–757, Sept./Oct. 2001.
7. Gao, L., Z.-L Zhang and D. Towsley. Catching and selective catching: efficient latency reduction techniques for delivering continuous multimedia streams. *Proc. 1999 ACM Multimedia Conf.*, pp. 203–206, Nov. 1999.
8. Golubchik, L., J. Lui, and R. Muntz. Adaptive piggybacking: a novel technique for data sharing in video-on-demand storage servers. *ACM Multimedia Systems Journal*, 4(3): 140–155, 1996.
9. Hu, A., I. Nikolaidis, and P. van Beek. On the design of efficient video-on-demand broadcast schedules, *Proc. 7th Int'l Symp. on Modeling, Analysis, and Simulation of Computing and Telecommunication Systems*, pp. 262–269, Oct. 1999.
10. Hua, K. A., Y. Cai, and S. Sheu. Patching: a multicast technique for true video-on-demand services. *Proc. 1998 ACM Multimedia Conf.*, pp. 191–200, Sep. 1998.
11. Hua, K. A., J. H. Oh, and K. Vu. An adaptive hybrid technique for video multicast. *Proc. 7th IEEE Int'l Conf. on Computer Communications and Networks*, pp. 227–234. Oct. 1998
12. Juhn, L. and L. Tseng. Fast data broadcasting and receiving scheme for popular video service. *IEEE Trans. on Broadcasting*, 44(1):100–105, March 1998.
13. Pâris, J.-F., D. D. E. Long and P. E. Mantey. A zero-delay broadcasting protocol for video on demand. *Proc. 1999 ACM Multimedia Conf.*, pp. 189–197, Nov. 1999.
14. Pâris, J.-F. A stream tapping protocol with partial preloading. *Proc. 9th Int'l Symp. on Modeling, Analysis and Simulation of Computer and Telecommunication Systems*, pp. 423–430, Aug. 2001.
15. Pâris, J.-F. A fixed-delay broadcasting protocol for video-on-demand. *Proc. 10th Int'l Conf. on Computer Communications and Networks*, pp. 418–423, Oct. 2001.
16. Sul, H.-K., H.-C. Kim, and K. Chon, A hybrid pagoda broadcasting protocol: fixed-delay pagoda broadcasting protocol with partial preloading. *Proc. 2003 IEEE Int'l Conf. on Multimedia and Expo*, July 2003.
17. Viswanathan, S. and T. Imielinski. Metropolitan area video-on-demand service using pyramid broadcasting. *ACM Multimedia Systems Journal*, 4(4):197–208, 1996.
18. Zhang, Q. and J.-F. Pâris. A channel-based heuristic distribution protocol for video-on-demand. *Proc. 2002 IEEE Int'l Conf. on Multimedia and Expo*, Vol. 1, pp. 245–248, Aug. 2002.