Practical Low Delay Broadcast of Compressed Variable Bit Rate Movies *

Neva Cherniavsky and Richard E. Ladner †
University of Washington

Abstract

H.264 is currently the best way to compress media to achieve high quality at low bandwidth. Since its inception, technologies such as video-on-demand are increasingly realizable. Periodic broadcast is a popular way of implementing video-on-demand, yet most current methods do not work on H.264 because they assume a constant bit rate stream and do not account for B frames. In this paper, we describe a new protocol for periodic broadcast of video-on-demand that works for variable bit rate streams with B frames. We map the periodic broadcast problem into the generalized windows scheduling problem of arbitrary length jobs on parallel machines. Our method is lossless and practical, in that it does not require channels of differing bandwidth. We apply our method to H.264 encoded video traces and achieve a delay of under 10 seconds on a 1.5 Mbps channel.

1 Introduction

With the advent of H.264, movies can be compressed at higher quality with far fewer bits than in earlier standards [25]. Technologies that were previously impractical can now be realized. Video-on-demand (VOD), a service whereby a customer interactively selects and downloads movies and other programming, is one such technology. A popular way to implement VOD is via periodic broadcast, where the customer incurs some delay and then can play the movie from start to finish. However, H.264 makes many current methods infeasible with its use of bidirectional (B) frames and constant quality video. B frames are predicted from previous and upcoming reference frames, and thus cannot be decoded until their reference frames are received. A constant quality video, in which each frame is encoded to nearly the same PSNR, requires a variable bit rate per frame. Most current VOD methods ignore the use of B frames and require a constant bit rate. The protocols that work for variable bit rate require

*This work was supported by NSF under grant number CCF-0514353 and by the NSF graduate research fellowship.

†Address: Department of Computer Science and Engineering, Box 352350, University of Washington, Seattle, WA 98195–2350, Tel: +1-206-543-1695; Fax: +1-206-543-2969. E-mail: *nchernia,ladner@cs.washington.edu.*

dividing the stream into logical channels of differing bandwidth, which is impractical for many applications.

In this paper, we introduce a new method for video-on-demand that is flexible enough to support H.264 compressed video, practically implementable, and yet reduces both bandwidth and delay compared to previous methods. The key insight is to model the problem as windows scheduling of arbitrary length jobs on parallel machines [2]. The jobs correspond to frames of the movie. The parallel machines correspond to multiple logical channels. In the same way that using multiple parallel machines can service more jobs than just using one fast machine, using multiple channels reduces delay compared to just increasing bandwidth.

For example, suppose we have two frames of different lengths, and suppose it takes 2 time slots to play a frame. Call the delay D. Our goal is to schedule the frames so that, no matter when the user tunes in, she will only need to wait D time slots before playing the movie. We also must ensure that she experiences no further delay once the movie starts.

In order to achieve this, we need to schedule the first frame so that it is received in every window of D time slots and the second frame so that it is received in every window of $D + 2$ time slots. With this schedule, no matter when the user tunes in, she will have completely received frame 1 after D time slots. She can play that frame and know that she will have completely received frame 2 once she has played the first frame.

Suppose the length of the first frame is 2 time slots and the length of the second frame is 3 time slots (based on the bandwidth of our channel). The minimum delay D for one channel is 5 time slots. The schedule would be a round robin between frames 1 and 2. To see why the minimum delay cannot be any smaller, consider Figure 1 and suppose the delay were 4. If the user is lucky and tunes in at the beginning of the round robin at the first arrow, she will not experience any problems. But if she is unlucky and tunes in at the second arrow, frame 1 will not be completely received after 4 time slots. We cannot change the schedule from a round robin, or else she will not receive frame 2 in time. On the other hand, if the delay is 5, she will always receive each frame within the corresponding window.

Now suppose we send the frames over two logical channels. Each channel has half of the bandwidth of the original. This means that our lengths are doubled, because it takes twice as long to receive the same number of bits. The length of the first frame is now 4 time slots and the length of the second frame is now 6 time slots. The minimum delay on two logical channels is $D = 4$. The schedule is simply to send the first frame on the first channel and the second frame on the second channel. The second frame takes 6 time slots to receive, but does not need to be received before the first frame is played, which takes 2 time slots. Thus, a delay of 4 time slots guarantees the second frame will be received in every window of size $D + 2$.

The rest of the paper is organized as follows. In the next section, we describe some related work on periodic broadcast VOD as well as the windows scheduling problem. In section 3 we present our algorithms. Section 4 contains our results, and section 5 the conclusion.

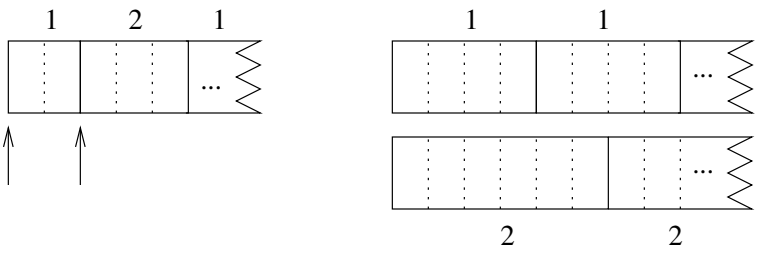

Figure 1: Schedule on one channel and two channels

2 Related Work

Video-on-demand has been studied extensively since the early nineties. There are two main ways in which a video service can provide VOD. A *reactive* service responds to requests from users. Reactive methods include unicast, in which there is a stream between the service and each new user. This does not scale well with the number of users, so researchers have explored multicast methods such as batching users together [5] and merging streams [6]. A *proactive* service anticipates requests. Periodic broadcast is a proactive service that requires the users to wait for some delay before they can begin watching the movie. This is essentially what many VOD services do today. It is the only option for satellite television, which has high bandwidth available for transmitting downstream but none available for transmitting upstream [9].

The primary advantage of periodic broadcast is that it takes much less bandwidth than unicast, and that in turn could allow a video service to offer more movies. Furthermore, research has shown that 80% of requests for movies are for the 20 or so most popular [4]. If these were sent over a periodic broadcast channel, the video service could meet the other 20% of requests via unicast, saving bandwidth. The main disadvantages of periodic broadcast are the need for a large buffer on the receiving end, lack of some VCR functionality, and the delay. Most periodic broadcast protocols assume that there is plenty of space available on the user's set top box, which is not unreasonable considering the cost of memory. Most protocols also do not allow for the VCR functionality of fast forward, though rewind and pause are easily implemented with a large buffer. Therefore, research in the field focuses on achieving a small delay with a minimal bandwidth. If the delay is on the order of minutes, then a user might turn to another service, such as a regular video rental store.

Perhaps the most natural way to implement periodic broadcast is via *Staggered Broadcast* [5], in which the movie is broadcast in its entirety over several channels at different intervals. If the length of the movie is N and there are k channels, the user experiences a maximum delay of N/k. A lower delay protocol is to send the movie in different segments over k channels, one for each segment. The channels may have differing bandwidth. Most of the research in the field uses this technique [24, 11, 12, 18, 20]. For a detailed survey, see [10]. These methods assume that the movie can be broken up in arbitrary places, or at the very least, along frame boundaries, which is not the case with any coding standard that includes B frames. Furthermore, they assume that the movie is received at some constant bit rate (often

the playback rate). It is possible to encode a constant quality movie at a constant bit rate, but the bandwidth requirement will be much higher than for the variable bit rate version [3].

Methods for variable bit rate VOD include lossy methods that use smoothing, server buffering and client prefetching [22, 14]. There are also several lossless methods. *Variable Bandwidth Harmonic Broadcasting* [17] changes the bandwidth per channel depending on the movie; each channel has differing bandwidth. The *Loss-Less and Bandwidth-Efficient* (LLBE) protocol [16] divides the movies into segments that respect frame boundaries. Each segment is broadcast in its own channel, at differing bandwidths per channel. The segments' divisions are chosen based on a dynamic program that returns the division that gives rise to the minimum total bandwidth. *General Frame Level Segmentation* (GFLS) [26] modifies LLBE to work on MPEG videos with B frames. The differing bandwidths used in these lossless methods are not co-factors, i.e., it is not generally the case that one channel's bandwidth is a multiple of another's. This renders them impractical for current video service providers.

The technique most closely related to ours is harmonic broadcasting[12, 18, 19]. Harmonic broadcasting divides the movie into segments and broadcasts segment i at bandwidth proportional to $1/i$. The worst case delay for bandwidth b asymptotically approaches $1/(e^b - 1)$. This is optimal for constant bit rate periodic broadcast [7, 8, 10]. Harmonic broadcasting has a nice mapping to windows scheduling, explored in [1]. That work showed that the optimal delay can be approached in the limit using windows scheduling techniques on channels of equal bandwidth.

3 Algorithm

3.1 Conversion to windows scheduling

We wish to schedule a periodic broadcast of a variable bit rate H.264 video so that no matter when users tune in to the broadcast, they experience a delay of D before they can begin playing the movie. We model the problem as windows scheduling of arbitrary length jobs on parallel machines.

The windows scheduling problem takes as input a sequence of n positive integer pairs $I = \langle (w_1, \ell_1), (w_2, \ell_2), \ldots, (w_n, \ell_n) \rangle$, representing n jobs. The ith job has length ℓ_i and must be executed within every window of size w_i. The goal is to schedule the jobs on the minimum number of parallel processors. Solving the problem optimally is NP-hard, but an 8-approximation is known, as well as a practical greedy algorithm [2].

We convert the video-on-demand problem to a windows scheduling problem as follows. The processors correspond to logical channels and the lengths correspond to the frame sizes. The window sizes correspond to our guarantee that the users experience only a small fixed delay before they can begin playing the movie.

Specifically, if L is the time it takes to play one frame, we must guarantee that frame 1 is received in every window of size D, that frame 2 is received in every window of size $D + L$, and that frame i is received in every window of size $D + (i - 1)L$. Since

the window size is in units of time, the frame lengths must be converted to time, and this will depend on the bandwidth and the number of logical channels.

Let the bandwidth in bits per second be denoted B, the number of logical channels k, the delay in seconds D, and the time it takes to play one frame L. Let b_i be the size in bits per frame i. Then the bandwidth per channel is B/k and the length of each frame in seconds is $b_i k/B$. The job sequence is then

$$\langle (D, b_1 k/B), (D + L, b_2 k/B), (D + 2L, b_3 k/B), \ldots, (D + (n-1)L, b_n k/B) \rangle .$$

Each window must be at least as large as the length of the job scheduled on it, so in particular, $D \geq b_1 k/B$.

3.2 Solving windows scheduling

Though the windows scheduling problem is NP hard, there is a greedy algorithm that works quite well in practice [2]. The algorithm is easier to visualize with the help of the tree representation of a schedule. Each channel is represented by a directed tree. The nodes of the tree consist of (w, ℓ) pairs. A (w, ℓ) node represents a window of size w and a length of size ℓ, so any (w', ℓ') scheduled on this node must have $w \leq w'$ and $\ell \geq \ell'$. There are two ways to schedule jobs on (w, ℓ):

1. (w, ℓ) may be split into k children of size (wk, ℓ). This corresponds to a round robin schedule over the children.

2. (w, ℓ) may be split into k children of size $(w, \ell_1), (w, \ell_2), \ldots, (w, \ell_k)$ such that $\sum_{i=1}^{k} \ell_i = \ell$. This corresponds to subdividing the window and allocating the appropriate number of slots to jobs.

For example, suppose we had a playback rate of $L = 3$ and converted frames lengths of $\ell_1 = 2, \ell_2 = 1, \ell_3 = 1$. Then to play on one channel, our minimum delay would be 3, giving $\langle (3, 2), (6, 1), (9, 1) \rangle$. The tree representation of the schedule is in Figure 2. In every window of size 3, the schedule would visit the $(3, 2)$ node, then one of the $(6, 1)$ or $(9, 1)$ nodes, as shown in the figure.

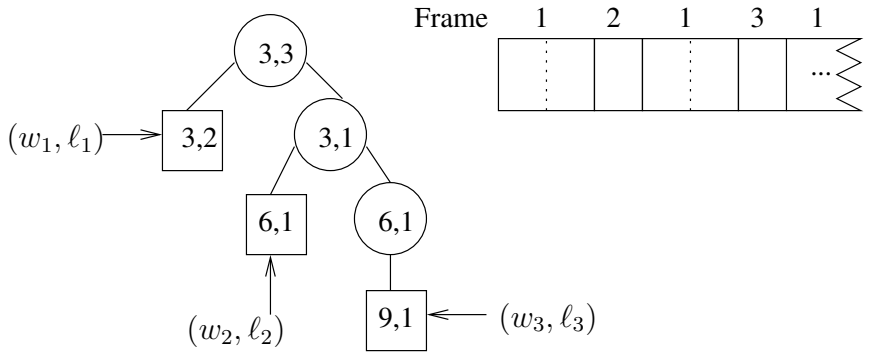

Figure 2: Tree representation and corresponding schedule. Boxes represent jobs.

To find the tree, the algorithm simply starts with one node of size (w_1, w_1) and schedules the first job (w_1, ℓ_1) as a child. That leaves a node of size $(w_1, w_1 - \ell_1)$. The next job, (w_2, ℓ_2), is scheduled as a child of this node by one of the two methods listed above. Note that our window sizes are always increasing. If the ith node cannot be scheduled, a new tree starts with (w_i, w_i) as its root.

Our goal is different than that of windows scheduling; we are interested in minimizing the delay or the bandwidth for a given set of frame lengths. To find the minimum delay, we set the bandwidth size and the number of channels and convert the frame lengths. We know that the minimum delay is at least the converted size of the first frame. We run our greedy algorithm with the delay set to this and the number of trees capped at the number of channels. If the algorithm reaches the cap, we run it again with double the delay. We continue doubling until we find a feasible delay given the number of channels; a delay equal to the sum of lengths is always feasible. Once we have a feasible delay, we perform a binary search between it and the last infeasible delay in order to determine the smallest delay that is feasible.

To find the minimum delay over any number of channels, we run the algorithm with the number of channels set from 1 to n, where n is the maximum number of channels. This adds a multiplicative factor of n to the running time. The algorithm for finding the minimum bandwidth is similar; we set the delay at the beginning and run the greedy algorithm for differing values of bandwidth. In this case, we set a maximum bandwidth of 100 Mbps. Since a window can never be smaller that its length, we can throw out some infeasible bandwidths right away (e.g. those in which $D \leq \ell_1 k/B$ where k is the number of channels).

3.3 H.264 modification

The above model is too simplistic in that it does not take B frames into account. In H.264, there are three frame types: I frames, P frames, and B frames. I frames are coded independently of the other frames in the video. P frames are predicted from previous I or P frames. B frames are predicted from previous or future I or P frames. If we used the above model based only on frames, we could no longer guarantee that the user could play the movie without interruption.

Our solution is to combine the frames into segments that do not depend on future segments. For example, a common H.264 coding sequence is IBBPBBPBBPBBI... Our algorithm divides this into segments:

$$\text{I, BBP, BBP, BBP, BBI, } \ldots$$

Each segment no longer depend on future segments. The windows scheduling algorithm will guarantee that all preceding segments are received before the current segment, and that the current segment is received in its entirety before it must be played.

Since different segments will be scheduled on the same channel, we include a unique identifier at the beginning and end of each segment. The number of segments will certainly be smaller than 2^{32}, so we use a 32-bit integer for our unique identifier. This adds a total of 64 bits to each segment.

4 Results

In order to demonstrate the applicability of our algorithm, we ran it on several video traces. The first is from [23] and uses H.26L, the working version of H.264, to encode the movie "Starship Troopers" at constant quality. Here we use the trace file with QP=20. The coding sequence is IBBPBBPBBPBBI. The number of frames is 90,000 and the length of the movie is 60 minutes. Figure 3 shows the minimum delay at different bandwidths and also the minimum bandwidth at different delays, plotted against the number of channels. The most gain occurs when we move from one channel to two. Though the results continue to improve as we add more channels, the returns are diminishing. Since the complexity of reading and reassembling the movie increases as the number of channels increase, we limit the number of channels in the graph to 25.

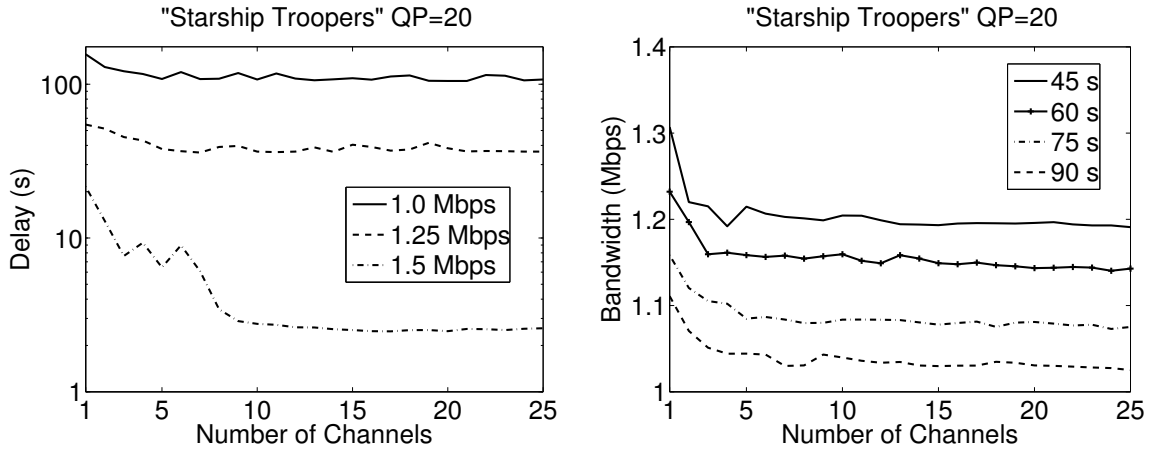

Figure 3: Delay at varying bandwidths and bandwidth at varying delays for "Starship Troopers"

These results demonstrate that a practical VOD system using H.264 is feasible, even for satellite television, where interactivity is not possible. Note that the delay for a 1.5 Mbps logical channel is less than 10 seconds. Since the bandwidth available on one cable channel is at least 30 Mbps [13], the provider could divide the channel into 20 subchannels, each of which would play one movie. Each subchannel of 1.5 Mbps would be further divided into the number of logical channels that guaranteed a low delay; in the case of "Starship Troopers", 5 channels would suffice. The division process is simple since the bandwidth is distributed equally. When the user requests one of the 20 movies, the set top box tunes into the appropriate channel and the movie begins playing in less than 10 seconds.

We also compare our results to LLBE/GFLS (the results for the two algorithms are quite similar [26]). The results from LLBE are expanded upon in a technical report [15], in which they plot delay against optimal server bandwidth on 7 channels for the MPEG-1 traces from [21]. In Table 1, we compare our minimum bandwidth using windows scheduling (WS) on 7 logical channels to theirs for traces mtv_1, news_2, and soccer_1 (fuss). Because MPEG-1 is a poor compression standard compared to H.264,

the bandwidth requirements are much higher. Though their method uses different bandwidths for different channels, our algorithm, with equal bandwidth channels, obtains comparable results.

	Delay (s)	LLBE	WS
mtv_1	15	3.8	3.3
	30	3.2	3.0
	60	2.6	2.6
	90	2.3	2.4
news_2	15	2.4	2.2
	30	2.0	1.9
	60	1.5	1.6
	90	1.4	1.5
soccer_1 (fuss)	15	4.1	3.6
	30	3.5	3.3
	60	2.8	2.9
	90	2.4	2.6

Table 1: Minimum bandwidth (Mbps) for given delay

5 Conclusion

We have presented an algorithm for periodic broadcast of variable bit rate movies. Our method is practical, in that it does not require channels of differing bandwidth to achieve low delay. In the future, we will look for better windows scheduling algorithms for this application. We would also like to explore the advantages of pre-caching the first few frames of a movie on the user's set top box at some point before the movie is requested. This could provide significant improvement in minimum bandwidth for a given delay, without taking up too much storage space. Finally, we would like to explore other problems in the space, such as taking user bandwidth limitations into consideration.

References

[1] A. Bar-Noy, R. E. Ladner, and T. Tamir. Scheduling techniques for media-on-demand. In *Proc. of the 14th ACM-SIAM Symposium on Discrete Algorithms (SODA '03)*, pages 791–800, 2003.

[2] A. Bar-Noy, R. E. Ladner, T. Tamir, and T. VanDeGrift. Windows scheduling of arbitrary length jobs on parallel machines. In *Proc. of the 17th annual ACM symposium on Parallelism in algorithms and architectures*, pages 56–65, 2005.

[3] I. Dalgic and F. A. Tobagi. Characterization of quality and traffic for various video encoding schemes and various encoder control schemes. Technical Report CSL-TR-96-701, Stanford University, 1996.

[4] A. Dan, D. Sitaram, and P. Shahabuddin. Scheduling policies for an on-demand server with batching. In *Proc. of ACM Multimedia*, pages 15–23, 1994.

[5] A. Dan, D. Sitaram, and P. Shahabuddin. Dynamic batching policies for an on-demand video server. *ACM Multimedia Systems Journal*, 4(3):112–121, 1996.

[6] D. L. Eager, M. K. Vernon, and J. Zahorjan. Optimal and efficient merging schedules for on-demand data delivery. In *Proc. of the 7th ACM Int'l Multimedia Conference*, pages 199–203, 1999.

[7] L. Engebretsen and M. Sudan. Harmonic broadcasting is optimal. In *Proc. of the 13th Annual ACM-SIAM Symposium On Discrete Mathematics (SODA '02)*, 2002.

[8] L. Gao, J. Kurose, and D. Towsley. Efficient schemes for broadcasting popular videos. *Multimedia Systems*, 8(4):284–294, 2002.

[9] T. Gnoffo. How hot is vod? *The Philadelphia Inquirer*, page C01, Oct 2005.

[10] Ailan Hu. Video-on-Demand broadcasting protocols: a comprehensive study. In *Proc. of the Twentieth Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM-01)*, pages 508–517, 2001.

[11] K. A. Hua and S. Sheu. Skyscraper broadcasting: A new broadcasting scheme for metropolitan video-on-demand systems. In *Proc. of the ACM SIGCOMM Conference : Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM-97)*, pages 89–100, 1997.

[12] L. Juhn and L. Tseng. Harmonic broadcasting for video-on-demand service. *IEEE Transactions on Broadcasting*, 43(3):268–271, Sept 1997.

[13] H. Kalva and B. Furht. Techniques for improving the capacity of video-on-demand systems. In *Proc. of the 29th Annual Hawaii Int'l Conference on System Sciences*, pages 308–315, 1996.

[14] F. Li and I. Nikolaidis. Trace-adaptive fragmentation for periodic broadcast of VBR video. In *Proc. of 9th Int'l Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV '99)*, 1999.

[15] F. Li and I. Nikolaidis. An inherently loss-less and bandwidth-efficient periodic broadcast scheme for vbr video. Technical Report TR00-16, University of Alberta, Canada, Jun 2000.

[16] I. Nikolaidis, F. Li, and A. Hu. An inherently lossless and bandwidth efficient periodic broadcast scheme for vbr video. In *Proc. of ACM SIGMETRICS 2000*, pages 116–117, 2000.

[17] J.-F. Pâris. A broadcasting protocol for compressed video. In *Proc. of Euro-media '99 Conference*, pages 78–84, 1999.

[18] J.-F. Pâris, S. W. Carter, and D. D. E. Long. Efficient broadcasting protocols for video on demand. In *Proc. of the 6th Int'l Symposium on Modeling, Analysis and Simulation of Computer and Telecommunications Systems (MASCOTS '98)*, pages 127–132, 1998.

[19] J.-F. Pâris, S. W. Carter, and D. D. E. Long. A low bandwidth broadcasting protocol for video on demand. In *Proc. of the IEEE Int'l Conference on Computer Communications and Networks (IC3N '98)*, 1998.

[20] J.-F. Pâris, S. W. Carter, and D. D. E. Long. A hybrid broadcasting protocol for video on demand. In *Proc. of the 1999 Multimedia Computing and Networking Conference (MMCN '99)*, 1999.

[21] O. Rose. Statistical properties of MPEG video traffic and theft impact on traffic modeling in ATM systems. Technical Report 101, University of Wuerzburg, Germany, 1995. Trace available at `http://www3.informatik.uni-wuerzburg.de/MPEG/`.

[22] D. Saparilla, K. Ross, and M. Reisslein. Periodic broadcasting with VBR-encoded video. In *Proc. of IEEE Infocom '99*, pages 464–471, 1999.

[23] P. Seeling, M. Reisslein, and B. Kulapala. Network performance evaluation using frame size and quality traces of single-layer and two-layer video: A tutorial. *IEEE Communications Surveys and Tutorials*, 6(2):58–78, 2004. Trace available at `http://trace.eas.asu.edu/mirrors/h26l/1462.html`.

[24] S. Viswanathan and T. Imielinski. Pyramid broadcasting for video on demand service. In *IEEE Multimedia Computing and Networking Conference*, volume 2417, pages 66–77, 1995.

[25] T. Wiegand, G. J. Sullivan, G. Bjntegaard, and A. Luthra. Overview of the H.264/AVC video coding standard. *IEEE Transactions on Circuits and Systems*, 13(7):560–576, Jul 2003.

[26] S. Wu. General frame level segmentation for periodic broadcast of vbr videos. In *Proc. of the 12th Int'l Conference on Computer Communications and Networks (ICCCN '03)*, pages 143–148, 2003.