Improving Video-on-Demand Server Efficiency Through Stream Tapping

Steven W. Carter and Darrell D. E. Long[†]
Department of Computer Science
University of California, Santa Cruz
Santa Cruz, CA 95064

Abstract

Efficiency is essential for Video-on-Demand (VOD) to be successful. Conventional VOD servers are inefficient; they dedicate a disk stream for each client, quickly using up all available streams. However, several systems have been proposed that allow clients to share streams. We present a new system called stream tapping that allows a client to greedily "tap" data from any stream on the VOD server containing video data the client can use. This is accomplished through the use of a small buffer on the client set-top box and requires less than 20% of the disk bandwidth used by conventional systems for popular videos. We present a description and analysis of the stream tapping system as well as comparisons between it and other efficiency-improving systems.

1 Introduction

Video-on-Demand (VOD) allows a client to connect to a VOD server using a television set-top box (STB), use the STB to make a selection from the server's video library, and begin watching the selected video after a short interval of time—ideally, instantaneously.

Unfortunately, VOD has run into some problems. Many companies jumped into the market only to quickly jump back out. Others entered with high expectations only to drastically cut back on what they had originally intended. The reason is cost: VOD is an expensive business to start up. Any system that can use existing hardware more efficiently or that can reduce the amount of hardware needed is very valuable.

Market tests suggest that VOD will be competitive with video rentals, cable movie channels, pay-per-view channels, and satellite television [1]. Thus it is also important for the VOD server to be run as efficiently as possible so it can support the large number of clients expected to use it.

There are two primary measures of VOD server efficiency: *latency*, the average time a client must wait before it can begin viewing its request, and *bandwidth*, the amount of disk (or network) resources used by the server. This poses difficulties for the VOD provider since minimizing one does not necessarily minimize the other. For example, the VOD provider might have the option of showing a particular video every five minutes or every ten minutes. The first option has half the latency but requires twice the bandwidth of the second option. Any strategy that can reduce both latency and bandwidth is valuable to VOD providers.

[†]This research was supported by the Office of Naval Research under Grant N00014–92–J–1807.

Two terms are important for understanding stream tapping and other efficiency-improving systems: *display stream*, a stream of data a client receives at its STB, and *disk stream*, a stream of data the VOD server reads from local (disk) storage. The number of simultaneous disk streams a VOD server can support while maintaining Quality-of-Service (QOS) is limited, and so the careful management of these streams is essential.

Conventional VOD systems use no strategy at all when it comes to their disk streams. They simply reserve a disk stream for each display stream. While this is the simplest to implement, it is also the least efficient.

Other systems, including stream tapping, attempt to service multiple display streams from each disk stream. This makes more efficient use of the available disk bandwidth on the VOD server, and with more clients able to use the server at any one time, latencies are usually lower as well.

What makes stream tapping unique is how it goes about increasing the number of display streams for each disk stream. The client STB initially receives its own disk stream, but then it is allowed to aggressively "tap" into other disk streams from the VOD server. This tapped data is then stored in a small local buffer until the STB needs it. Notice that every time the STB is able to tap data, the initial disk stream, which had only one display stream, is needed for less time, and the tapped disk stream gains another display stream over the time the STB is able to tap data from it. This increases the average number of display streams per disk stream.

2 Related Work

Several other systems for improving VOD server efficiency have been proposed, and we describe some of them below. Some of the systems use several techniques. We distinguish them by using the most fundamental idea of the system.

2.1 Batching

A simple but effective technique for improving VOD server efficiency is known as *batching* [2, 3]. When the VOD server has multiple requests for the same video in its request queue, it may service them all (that is, batch the requests together) by multicasting the video to all of the requesting clients. However, this strategy will not attempt to make efficient use of the server's disk streams until all of the streams are in use.

2.2 Staggered Broadcasting

In this system [4, 5], a disk stream for a video is only started at regular intervals (such as every 10 minutes), and all requests for the video received during the current interval are batched together. While this makes very good use of the server's available bandwidth, it guarantees a non-zero average latency.

2.3 Pyramid Broadcasting

Pyramid broadcasting [6, 7] is a variation on staggered broadcasting. It also reserves a certain number of disk streams for particular videos, but rather than having the disk streams read the entire video, it has the streams read multiplicatively increasing segments of the video. The client STB must then jump from stream to stream in order to receive the entire video.

Pyramid broadcasting reduces the latency found in staggered broadcasting, but in order for the STB to receive each segment in time, the system requires that the video data be transferred at a rate much higher than it is consumed. The STB also requires a local buffer of moderate size: for MPEG-1, the buffer must be between 250 and 600 MB, depending on the version of pyramid broadcasting used.

2.4 Piggybacking

In this system [8, 9], the display rates of videos are changed by ±5% (little enough so human observers should not notice) so that two existing disk streams can be "merged" into one. However, the slow merging rate (two streams 3 minutes apart take 30 minutes to merge) limits this system's potential.

2.5 Asynchronous Multicasting

Asynchronous multicasting [10, 11] allows a client to join a multicast group for a video after the video has started. The VOD server accomplishes this by breaking up the video into segments of length S and sending out a segment every S minutes—but using a transfer rate N times the consumption rate of the video, so the transfer takes only S/N minutes. This allows a client to join a multicast group late, store the segments that are current for the other members of the group in a local buffer until they are needed, and use the gaps between the segments to receive segments which it missed.

The buffer for asynchronous multicasting must be able to hold at least NS minutes of video data. Using $N = 3$ and $S = 6$ [11], this is larger than 200 MB. Also, since the client can only receive one segment at a time (due to the high transfer rate), in order to join an existing multicast group it must receive its first segment before the group receives the N^{th} segment of the video. That means a client can only join a multicast group that started less than $(N-1)S - S/N$ minutes in the past.

Stream tapping uses a variation on asynchronous multicasting (see §3.1), but it does not break the video into segments, does not make any assumptions about the transfer rate, makes more efficient use of the client buffer, and requires a lower data rate at the client.

3 Stream Tapping

The key idea behind stream tapping is that clients are not restricted to their assigned disk stream. If other disk streams for the same video are active on the VOD server, clients are allowed to "tap" into them, storing the tapped data in a local buffer until it is needed. By using existing disk streams as much as possible, the clients minimize the amount of time they require their own disk streams. The rest of this section elaborates upon how this strategy works.

3.1 Definitions

Several of the parameters used by stream tapping are defined below:

β	the size of the STB buffer, measured in minutes of video data. Measuring in time allows us to ignore the particulars of the video encoding used. We assume that all STB's have the same buffer size, although this is not required.
N	the number of videos offered by the VOD server.
L_i	the length of video i, in minutes, for $1 \leq i \leq N$.
S	the maximum number of disk streams the VOD server can support.
C	the maximum number of disk streams the client STB can support.
λ	the arrival rate of requests at the VOD server, measured in requests per hour.
Δ	the difference in time, in minutes, between the current request for a video and the last request for the same video which required an original disk stream.

Of the above values, only C, N, and S are required to have integer values.

Stream tapping also divides disk streams into three types. These types are defined as follows:

1. *Original Streams*
   With an original disk stream, the video is read from disk in its entirety. We can calculate the number of *disk minutes* for these streams—that is, the amount of time the stream reads data from local (disk) storage—based on the video being requested:
   $$D_o(i) = L_i \tag{1}$$

2. *Full Tap Streams*
   A full tap disk stream can only be used when the display stream for the requested video starts within β minutes of an original disk stream for the same video (that is, when $\Delta \leq \beta$). When this happens, the requesting STB will be able to "tap" into the original disk stream and receive data from it and its assigned full tap stream. In particular, during the first Δ minutes, the STB will receive video data from both streams. The full tap stream will be displayed live, and the original stream will be stored in the STB's buffer. After Δ minutes the STB will be able to release the full tap and receive the rest of the video data direct from its buffer, which is continually filled from the original stream and always contains a moving Δ-minute window of new video data. We can calculate the number of disk minutes required by full tap streams based on Δ:
   $$D_f(\Delta) = \Delta \tag{2}$$

3. *Partial Tap Streams*
   A partial tap disk stream can be used in any situation where a full tap stream cannot be used, as long as an original stream for the video is active (that is, when $\Delta > \beta$). This type of disk stream is similar to a full tap in that the requesting STB will also "tap" into the original stream, but unlike the full tap, it will not be able to completely release the partial tap stream until the video is complete. In particular, during the first β minutes, the STB will receive data from both streams (receiving minutes Δ to $\Delta + \beta$ of the video from the original stream and storing it in its buffer), and then it will repeat the following until the video is finished:
   - The STB's buffer is full, and the video data it contains is $\Delta - \beta$ minutes away from the STB's current position in the video. The STB will reacquire (if necessary) the partial tap stream and receive the next $\Delta - \beta$ minutes of video data from there.
   - The STB will then temporarily release the partial tap stream for the next β minutes. During that time it will display the video data in its buffer and receive data from the original stream once again, filling up its buffer while simultaneously emptying it.

   In total, the partial tap stream will only need to exist for the first β minutes and then only for the first $\Delta - \beta$ minutes of every succeeding Δ-minute interval. We can calculate the number of disk minutes required by partial tap streams in terms of the video being requested and Δ:
   $$D_p(\Delta, i) = \beta + \lfloor \frac{L_i - \beta}{\Delta} \rfloor (\Delta - \beta) + \min(\Delta - \beta, (L_i - \beta) \bmod \Delta) \tag{3}$$

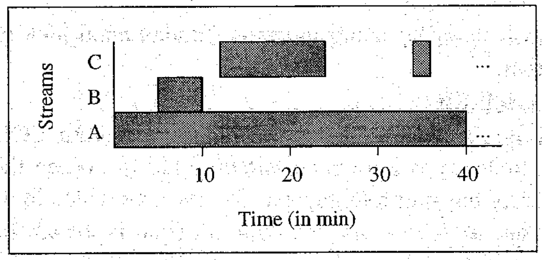

Figure 1: Three types of disk streams: A is an original stream, B is a full tap stream, and C is a partial tap stream. The shaded areas indicate when the streams are active.

Figure 1 gives an example of each of the three types of disk streams when the buffer size is 10 minutes. The original stream starts at time T_0, the full tap stream at time $T_0 + 5$, and the partial tap at time $T_0 + 12$.

3.2 The Algorithm

Using the definitions from §3.1, we can now describe the stream tapping algorithm. Every time the VOD server can service requests, it must first assign each request in its request queue one of the three disk stream types.

1. If no instance of the requested video is currently being read from disk using an original stream, then the request requires an original stream.

2. If an original stream for the requested video started within the last β minutes, then the request can use a full tap stream.

3. If an original stream for the requested video started over β minutes in the past, then a decision must be made about the type of disk stream (either original or partial tap) that the request should receive.

This decision can be made based on the request's *video group*. A video group is the set of disk streams for the requested video, starting with the most recent original stream and including all subsequent tap streams. With a minimal amount of extra storage (one counter for each video), the VOD server can keep track of λ_g, the scheduling rate of streams in the group.

Given λ_g and Δ, the VOD server estimates two values:

- The average disk usage of a video group which exists for $\Delta + 1/\lambda_g$ minutes and has a scheduling rate of λ_g.
- The optimal average disk usage of a video group which exists for less than or equal to Δ minutes and has a scheduling rate of λ_g.

The first is the average usage of the group with the request, and the second is the best average usage of the group without the request. If the first value is less than the second, the request is assigned a partial tap stream; otherwise it is assigned an original stream.

Once all of the requests have been assigned stream types, the VOD server will know deterministically the disk scheduling and usage required by each (for this iteration). It can then use this information to order the requests in the queue and to determine which requests can be serviced.

3.3 Other Options

With the basic part of the stream tapping system, the client STB need only receive at most two disk streams at any one time. If it can receive more without affecting the QOS requirements, then two other options can be used.

The first of these options is called *extra tapping*. It allows a stream to tap data from any disk stream on the video server, not just from the original stream in its video group. This can only be performed under two conditions: the new video data does not displace any data the STB expects to be in its buffer, and the new video data will still be in the buffer when it is needed. Together these conditions mean that extra tapping can be used only during the first β minutes of full and partial taps.

An example of extra tapping is shown in Figure 2. The buffer size is 10 minutes, and B and C are full tap streams starting, respectively, 5 and 7 minutes after original stream A. The before part of the figure shows the video data (light gray) that the STB receiving stream C can tap from stream B. The after part shows the only parts of the full tap streams that need to be reserved on the VOD server.

The second option is called *stream stacking*. When an STB has data in its buffer to which it is trying to "catch up," and when it also has extra space in its buffer, it can use any available disk streams on the VOD server to help load in more quickly the data it needs. This does not change the

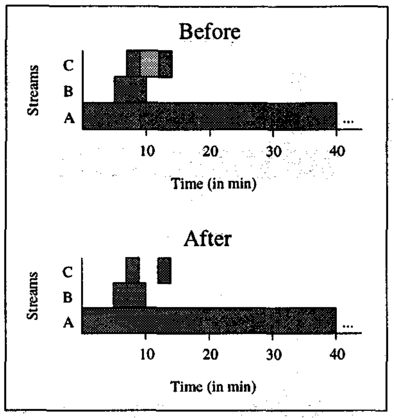

Figure 2: Extra tapping: A is an original stream and B and C are full tap streams to A. The lightly shaded area indicates the part of B which C can tap.

number of disk minutes required by the stream, but it rearranges when they take place, potentially avoiding future bandwidth contention.

Figure 3 provides an example of stream stacking. The buffer size is 10 minutes, and B is a full tap stream starting 5 minutes after original stream A. Since the STB receiving B only needs to reserve half of its buffer for stream A's data, it can use the rest of the buffer to more quickly load the first five minutes of the video. In this example, we assume stream E is available, and that the STB receiving B is able to use it for two minutes before another stream reserves it. The before part of the figure shows the part of stream B (light gray) that is read by stream E, and the after part shows how stream B becomes available two minutes earlier than it would have otherwise.

4 Simulation

We analyzed the stream tapping system using simulation. Each run of the simulation consisted of a two-hour warm-up period followed by a twelve-hour interval during which statistics were kept. Each data point presented in the next section is the mean average of five such runs. This kept the variance of the values to (typically) less than 1%.

4.1 Videos

The length of each video was modeled using a normal distribution with a mean of 110 minutes and a standard deviation of 10 minutes. These lengths were capped at a minimum of 90 minutes and a maximum of 180 minutes to keep the values realistic.

The probability of each video was modeled using a Zipf-like distribution. This distribution was recommended by Drapeau *et al.* [12], and as in other studies [2, 9] we configured the distribution to more closely fit empirical

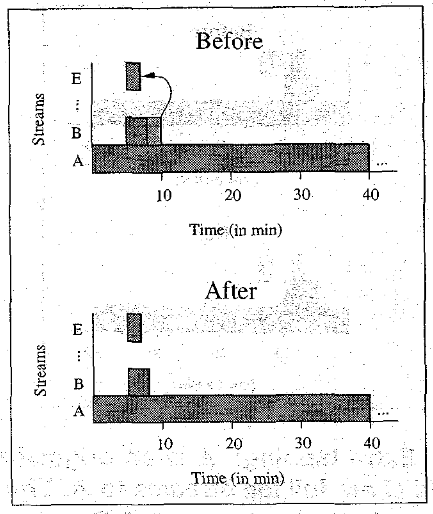

Figure 3: Stream stacking: A is an original stream and B is a full tap stream. The lightly shaded area indicates the part of B that is "stacked" by E.

video rental patterns [13] using Zipf parameters $N = 92$ and $\theta = 0.271$.

4.2 Clients

Client requests were generated based on a Poisson arrival process with interarrival time $1/\lambda$. Upon arrival, clients selected a video based on the distribution described above. Clients never *reneged* on their requests, and they did not interact with the video through pause, rewind, and other VCR-like controls. We will explore these issues in the future.

4.3 VOD Server

The VOD server acted like an ordinary server: it received requests, serviced requests if it had available resources, queued requests if it did not, and tried to service requests in its queue every time an event occurred that made a disk stream available.

The amount of time it takes a VOD server to access a video depends on the particular server architecture being used (for example, if it uses tertiary storage for unpopular videos). Since we did not want to make any assumptions about the server, and since we also wanted to remove the effects of the server from the results of the simulation, we took this latency to be zero.

4.4 Network

Any network can be used to transport video data for VOD, but the characteristics of the network drastically influence its effectiveness. For this reason we did not want to make any assumptions about the characteristics of the network, and as with the VOD server, we also wanted to remove its effects from the results of the simulation. We assumed that the data was able to traverse the network with zero latency and that the network always had bandwidth available to the VOD server. While the assumptions made here and in §4.3 may seem overly simplistic, they are consistent with the approach taken by other researchers [2–11].

5 Results

This section gives several results for the stream tapping simulation. A more complete description can be found in another report [14].

In a default configuration:

- The VOD server has 300 disk streams and 92 videos ($S = 300$, $N = 92$).
- The client STB has a 10-minute buffer and can receive up to 4 disk streams at any one time ($\beta = 10$, $C = 4$).

Unless otherwise specified, the results presented in this section used the values from this configuration. Also, unless otherwise specified, the simulation used both stream stacking and extra tapping.

Figure 4 shows how the size of the client buffer affects disk usage on the VOD server. In order to determine the disk usage on the server without any bias introduced due to contention for bandwidth, we gave the server an unlimited number of disk streams. To make the resulting disk usage more meaningful, we restricted the server to a single video. Thus, Figure 4 also shows that with a 10-minute buffer on the client STB, the server needs less than 15 disk streams on average to provide a latency of zero for a video, even if the video is requested 120 times per hour.

Note that a larger client buffer improves the disk usage on the VOD server for all arrival rates except for the largest, $\lambda = 120$. This is because stream tapping was designed with a small buffer in mind, and when the buffer size and arrival rate become large enough, it is not always best for the VOD server to assign a request a full tap stream. Changing the algorithm so that a stream decision is made for all tap streams (instead of just partial tap streams) is a trivial matter, and it is something we will explore in the future.

Figure 5 shows how the size of the client buffer affects latency. This graph is very encouraging; even when there were twice as many requests per hour as there were disk streams on the VOD server, the server was able to handle the load and and give reasonable latencies, even for the 5-minute buffer. Note that a conventional system, which reserves a disk stream for each client, can only handle about 164 requests per hour when it has 300 disk streams; any more and its request queue will grow to infinity. Stream tapping can handle almost twice that many requests per hour before it even begins to generate non-zero latencies (with $\beta \geq 10$).

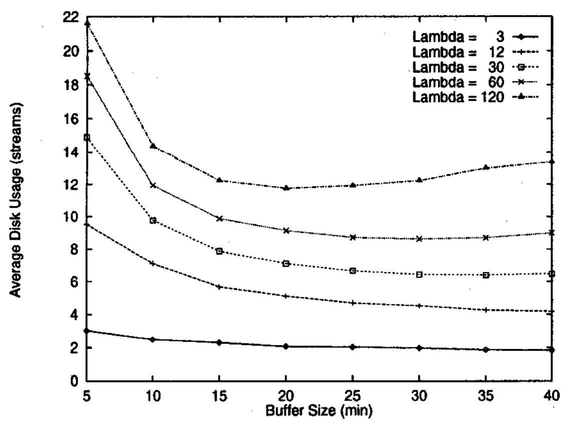

Figure 4: Effects of the STB buffer size on disk usage ($N = 1, S = \infty$).

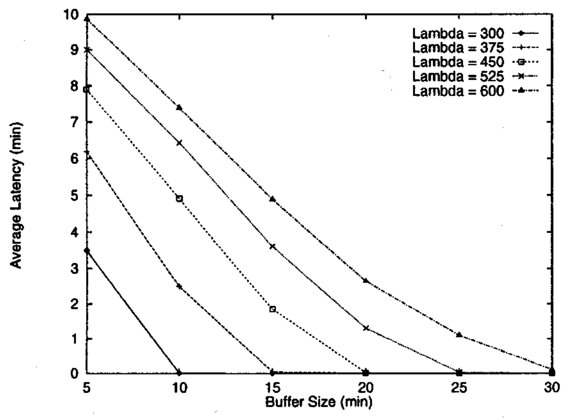

Figure 5: Effects of the STB buffer size on latency.

Figure 6 shows how the number of disk streams on the VOD server affects latency. The arrival rates are given as a percentage of those streams per hour. This allows each arrival rate in the figure to be meaningful. Unsurprisingly, the VOD server performs much better as it receives more disk streams. This is simply because increasing the number of requests for a video by some factor does not also increase the bandwidth used by the video by the same factor. This is one reason that stream tapping scales well.

Figure 7 shows how much disk bandwidth is saved by using stream tapping instead of a conventional system. We could compare latencies as well, but for any arrival rate that gives non-zero latencies for stream tapping, a conventional system generates an infinite queue. Note that stream tapping saves over 80% when the interarrival time is 2 minutes or less (that is, when the video is popular), and even saves 15% when the interarrival time is 60 minutes.

Figure 8 compares stream tapping to the two broadcasting systems. Because of their deterministic nature, it is

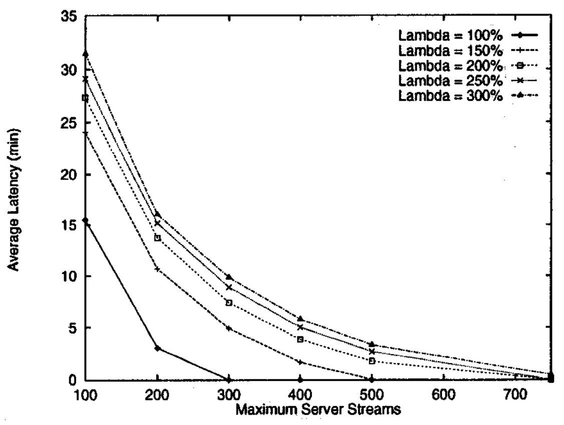

Figure 6: Effects of the number of disk streams on the VOD server latency.

possible to write functions for latency based on the disk bandwidth (measured in streams) provided the two broadcasting systems. Given a video i, we used

$$\mathcal{L}_s(S) = \frac{L_i}{2S}$$

for staggered broadcasting and

$$\mathcal{L}_p(S) = \left(\frac{L_i}{3}\right)\left(\frac{1}{2^{S/3} - 1}\right)$$

for pyramid broadcasting [7]. Note that even with the high arrival rate (a request every ten seconds) stream tapping outperformed both broadcasting systems given sufficient disk streams.

Figures 9 and 10 compare stream tapping to batching and asynchronous multicasting. We estimated the performance of asynchronous multicasting by modeling it as stream tapping with only full tap streams. This provides an upper bound on its performance since, using a 10-minute buffer, a request in asynchronous multicasting can only join a multicast group for a video that started less than 6–7 minutes in the past. Our model increases this to 10 minutes.

Figure 9 compares the three systems using disk bandwidth. It is probably a little misleading since by allowing the VOD server unlimited disk streams to measure usage without contention, the server never has any requests in its queue, and thus batching in this case performs exactly the same as a conventional system. However, in both Figures 9 and 10, stream tapping handily outperforms the other systems.

The only system that we could not compare stream tapping to directly was piggybacking. However, we note that the "simple merging policy" [8] is essentially the same as stream tapping using only full taps and no options, but

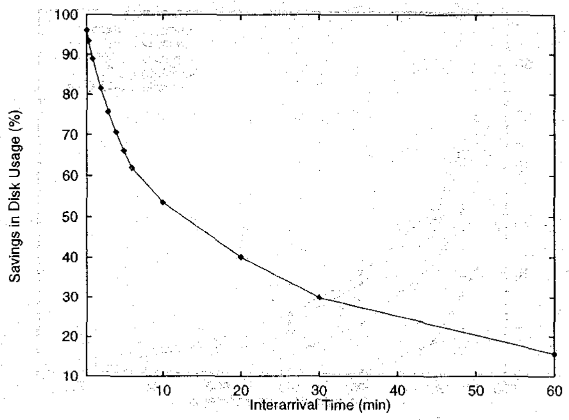

Figure 7: Savings in disk usage by using stream tapping instead of a conventional system ($N = 1$, $S = \infty$).

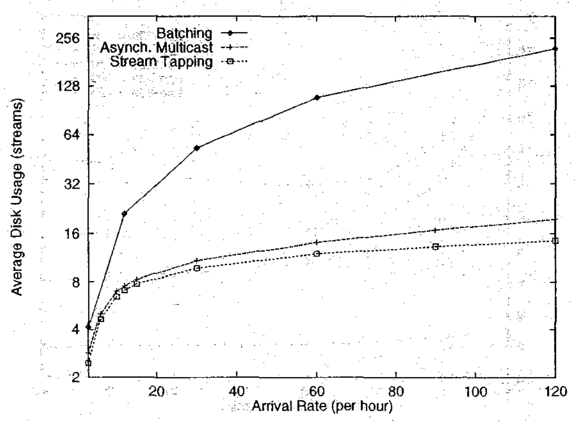

Figure 9: Using disk usage to compare batching, asynchronous multicasting, and stream tapping ($N = 1$, $S = \infty$).

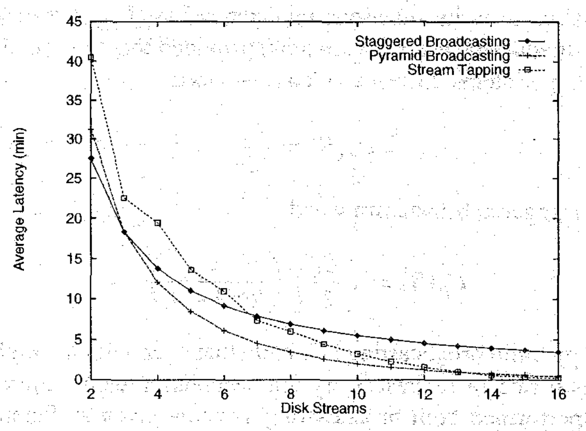

Figure 8: Using latency to compare staggered broadcasting, pyramid broadcasting, and stream tapping ($N = 1, \lambda = 360$).

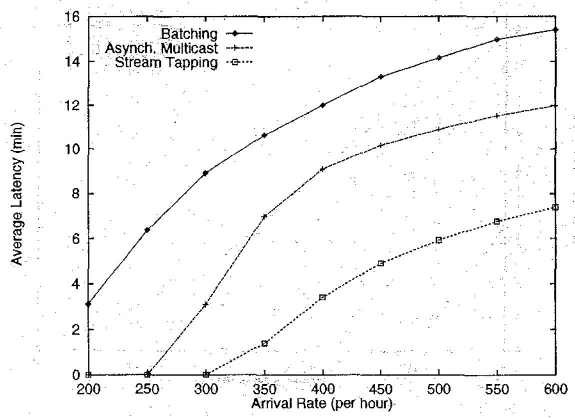

Figure 10: Using latency to compare batching, asynchronous multicasting, and stream tapping.

where the number of disk minutes required by the full tap is 10Δ rather than Δ. Thus we are confident stream tapping would give better results than this system as well.

6 Conclusions

We have described a system called stream tapping that can improve the disk bandwidth utilization and decrease the average client latency for a VOD server. In particular, it

- Requires less than 20% of the disk bandwidth used by conventional systems for popular videos;
- Easily handles arrival rates for which conventional systems generate infinite queues; and
- Performs as well or better than several other VOD systems under a variety of conditions.

These results are possible because stream tapping adds complexity to the client STB: the STB must be able to receive data at a rate higher than the consumption rate for the video, and it must have a local buffer. This sounds like a significant drawback to the system, but if MPEG-1 encoding is used for the videos, then the data rate will always be less than 6 Mbps and the (10-minute) buffer will be less than 120 MB. These are not outstanding requirements to place on the STB—especially when STB's are expected to have prices similar to VCR's—and stream tapping can adapt to much more restrictive conditions.

7 Acknowledgments

We would like to thank T. Sconyers, T. Haining, T. Kroeger, and J. F. Pâris for their insights and helpful comments. We would also like to acknowledge the support of the Office of Naval Research under Grant N00014–92–J–1807.

References

[1] James R. Allen, Blaise L. Heltai, Arthur H. Koenig, Donald F. Snow, and James R. Watson. VCTV: a video-on-demand market test. *AT&T Technical Journal*, 72(1):7–14, January 1993.

[2] Asit Dan, Dinkar Sitaram, and Perwez Shahabuddin. Dynamic batching policies for an on-demand video server. *Multimedia Systems*, 4(3):112–21, June 1996.

[3] Charu C. Aggarwal, Joel L. Wolf, and Philip S. Yu. On optimal batching policies for video-on-demand storage servers. In *Proceedings of the International Conference on Multimedia Computing and Systems*, pages 253–8, Hiroshima, Japan, June 1996. IEEE Computer Society Press.

[4] Kevin C. Almeroth and Mostafa 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(5):1110–22, August 1996.

[5] Tzi-cker Chiueh and Chung-ho Lu. A periodic broadcasting approach to video-on-demand service. *Proceedings of the SPIE – The International Society for Optical Engineering*, 2615:162–9, 1996.

[6] S. Viswanathan and T. Imielinski. Metropolitan area video-on-demand service using pyramid broadcasting. *Multimedia Systems*, 4(4):197–208, August 1996.

[7] Charu C. Aggarwal, Joel L. Wolf, and Philip S. Yu. A permutation-based pyramid broadcasting scheme for video-on-demand systems. In *Proceedings of the International Conference on Multimedia Computing and Systems*, pages 118–26, Hiroshima, Japan, June 1996. IEEE Computer Society Press.

[8] Leana Golubchik, John C. S. Lui, and Richard R. Muntz. Adaptive piggybacking: a novel technique for data sharing in video-on-demand storage servers. *Multimedia Systems*, 4(30):140–55, June 1996.

[9] Charu C. Aggarwal, Joel L. Wolf, and Philip S. Yu. On optimal piggyback merging policies for video-on-demand systems. In *Proceedings of the International Conference on Multimedia Systems*, pages 253–8, Hiroshima, Japan, June 1996. IEEE Computer Society Press.

[10] Heekyoung Woo and Chong-Kwon Kim. Multicast scheduling for VOD services. *Multimedia Tools and Applications*, 2(2):157–171, March 1996.

[11] Hari Kalva and Borko Furht. Techniques for improving the capacity of video-on-demand systems. In *Proceedings of the 29th Annual Hawaii International Conference on System Sciences*, pages 308–15, Wailea, HI, USA, January 1996. IEEE Computer Society Press.

[12] Ann L. Drapeau, David A. Patterson, and Randy H. Katz. Toward workload characterization of video server and digitial library applications. *1994 ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems*, 22(1):274–5, May 1994.

[13] *Video Store Magazine*, December 13, 1992.

[14] Steven W. Carter and Darrell D. E. Long. Stream tapping: a system for improving efficiency on a video-on-demand server. Technical Report UCSC–CRL–97–11, University of California, Santa Cruz, April 1997.