# Stream Tapping: a System for Improving Efficiency on a Video-on-Demand Server

**Steven W. Carter and Darrell D. E. Long**[†]

**UCSC–CRL–97–11**
**November 2, 1997**

Department of Computer Science
University of California, Santa Cruz
Santa Cruz, CA 95064

ABSTRACT

Video-on-Demand (VOD) allows clients to view selected videos at any time, and it is essential for VOD servers to be run as efficiently as possible. Conventional VOD servers are not efficient; they dedicate a disk stream for each client, and this strategy quickly uses up all available streams. Other systems, such as batching, interval caching, and pyramid broadcasting, have been studied. These systems make more efficient use of the VOD server's disk streams.

In this report we present a new VOD system called stream tapping. Stream tapping allows clients to aggressively "tap" into any disk streams on the VOD server that are reading data the client can use. This can be accomplished through the use of a small buffer—as small as 115 MB for MPEG-1 encoding—on the client's set-top box, and it can save over 80% of the disk bandwidth used by a conventional system. This report includes a description and analysis of the stream tapping system, and comparisons between it and other efficiency-improving systems.

**Keywords:** Video-on-Demand, efficiency, bandwidth

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

# 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 then begin viewing the selected video in a short amount of time.

VOD is not yet commercially available. Many companies have run trials for VOD over the last five years [1–3], but by and large these companies have then either scaled back what they intended or left the VOD business altogether. The reason is cost. VOD—from upgrading or creating new networks to developing new software to buying and maintaining hardware —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.

Also, market tests suggest that VOD will be competitive with video rental stores and pay-per-view channels [1, 4]. With this large number of potential clients, it is important for the VOD server in particular to be run efficiently. It must be able to handle a variety of loads without (greatly) sacrificing performance, and it must be able to scale well as more and more clients attempt to use its services.

The three main hardware components of a VOD service are the VOD server, the client STB, and the network that connects the two. See Figure 1. Their role in VOD is described below.

- *VOD Server*
  Many VOD server architectures have been explored in the literature [5–10], and they range from distributed systems that contain a hierarchy of file system and storage nodes to stand-alone file servers that operate using only local storage. However, regardless of the architecture, the VOD server must perform three roles:

  1. Act as a repository for a large number of videos,
  2. Be able to support multiple, simultaneous accesses to videos, and
  3. Be able to communicate with clients so clients can select and view videos.

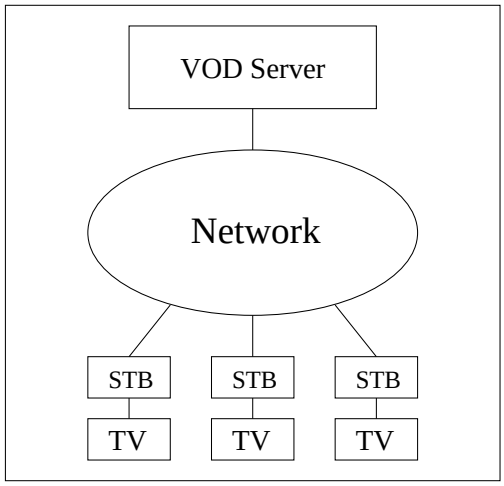


Figure 1: The hardware components of a VOD service.

- *Client STB*
  The role of the client set-top box is usually dependent on the specific needs of a particular VOD provider [11, 12] but some trends are becoming clear [13, 14]. The STB must:

  1. Be able to communicate with the VOD server so that it can navigate possible video selections and receive video data,
  2. Be able to decode the data it receives, both for descrambling and MPEG decompression purposes,
  3. Have a certain amount of processing power, and
  4. Have at least a small amount of local buffer, both to keep enough video frames on hand to prevent *jitter*[1] and for possible network transport strategies [15, 16].

  These requirements are enough that many VOD providers are using stripped-down computers for the STB [3, 11].

[1] *Jitter* occurs when the next frame to be shown is not available, and the STB must wait for the frame to arrive or skip the frame (and probably other frames) in order to resynchronize.

- *Network*
  The network is simply the means of communication between the VOD server and the client STB. It does not have any other requirements per se, but it should provide such things as high bandwidth, guaranteed quality-of-service[2] (QOS), and multicasting. This makes ATM an ideal networking choice [17, 18] although even the Internet can be used [8].

Given these three hardware components, there are many issues involved in their implementation. In this report we concentrate on the VOD server—in particular, how to make it more efficient. The two measures of efficiency we will use are:

- *Latency:* the average time a client must wait before it can begin viewing its request.
- *Bandwidth:* the amount of disk (or network) resources used by the server.

Although these two measures appear to correlate well, it is not necessarily the case. For example, a 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.

There are two terms that are important for understanding stream tapping and other efficiency-improving systems:

- *Display stream:* a stream of data a client receives at its STB.
- *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 the necessary QOS of the data is limited, and so the careful management of these streams is important.

[2] *Quality-of-service* is simply a commitment to a particular level of performance. For the network, this relates to the amount of time it takes a video frame to go from the VOD server to the client STB.

Conventional VOD systems do not use any strategy at all when it comes to their disk streams. They simply reserve a disk stream for each display stream. While this is the easiest strategy 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, storing the tapped data in a local buffer until it is needed. Every time the client is able to tap data, its assigned stream (which only has one display stream) will not be needed for as long, and the other disk stream will be able to increase its display streams by one for the amount of time the STB is able to tap data from it. This increases the average number of display streams per disk stream.

The remainder of this report is organized as follows. We present the stream tapping system in §2 and then remark on some of its hardware requirements in §3. In §4 we describe other VOD systems that have been presented in the literature. The simulation used to test the stream tapping system is described in §5, and in §6 we discuss the results from that simulation. In §7 we outline some future plans with stream tapping. Finally, in §8, we provide some concluding remarks.

## 2 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.

## 2.1 Definitions

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

- $\beta$ 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.
- $N$ 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 simultaneous disk streams that the VOD server can support.
- $C$ the maximum number of simultaneous disk streams that the client STB can receive.
- $\lambda$ the arrival rate of requests at the VOD server, measured in requests per hour.
- $\Delta$ the difference in time, in minutes, between the current request for a video and the last request for the same video that required an original disk stream.

Note that $\beta$, $L$, $\lambda$, and $\Delta$ are not required to have integer values.

Stream tapping also divides disk streams into three types:

1. *Original Streams*
   Original disk streams can be used at any time. However, they require that the requested video be read from disk in its entirety, which means they are busy (*i.e.* reading from storage) for

$$B_o(i) = L_i \tag{1}$$

   minutes, where $i$ is the index of the requested video.

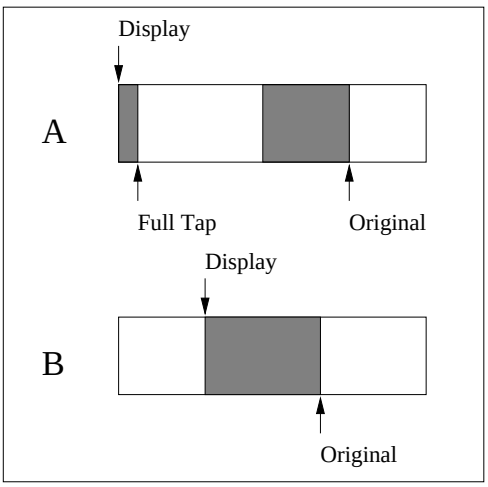


Figure 2: Examples of an STB's buffer while it is receiving a full tap stream (A) and after the full tap stream has been released (B).

2. *Full Tap Streams*
   A full tap disk stream can only be used when the requested video starts within $\beta$ minutes of an original disk stream for the same video (*i.e.* when $\Delta \leq \beta$). This allows the full tap stream to work in tandem with the original stream so that the full tap stream can be released well before the video is complete.

   In particular, the requesting STB will receive both the full tap and original streams for $\Delta$ minutes. During that time the full tap stream will read the first $\Delta$ minutes of the video, and the STB will display it live. The data from the original stream will be stored in the STB's buffer. After $\Delta$ minutes, the STB will be able to release the full tap stream and receive the rest of the video from its buffer, which will be continually updated from the original stream and contain a moving $\Delta$-minute window of the video. That means a full tap stream will be busy for

$$B_f(\Delta) = \Delta \tag{2}$$

   minutes.

   Figure 2 gives two examples of the state of an STB's buffer when the STB is assigned a full tap stream. The shaded areas

of the buffer indicate video data the STB still needs to display, with the most recent data on the right. The first part of the figure shows the buffer during the first $\Delta$ minutes when the STB is receiving two disk streams. The full tap stream is being displayed, and just enough of it is kept in the buffer to prevent jitter. The second part shows the buffer after the full tap stream has been released. Video data from the original stream is being received and displayed (consumed) at the same rate, leaving a constant window of data in the buffer.

3. *Partial Tap Streams*
A partial tap disk stream can be used for a requested video in any situation where a full tap stream cannot be used, as long as there is an original stream for the video currently active (*i.e.* when $\Delta > \beta$). As with the full tap stream, a partial tap stream can work in conjunction with the original stream, but unlike the full tap stream, the partial tap cannot be fully released until the video is complete.

   In particular, during the first $\beta$ minutes, the STB will receive both streams. The partial tap stream will read the first $\beta$ minutes of the video, and the original stream will read minutes $\Delta$ to $\Delta + \beta$ of the video. After that, the STB will repeat the following until the video is complete:

   - The STB's buffer is full, and the video data it contains is $\Delta - \beta$ minutes away from the STB's current place in the video. The STB will reacquire (if necessary) the partial tap stream and receive the $\Delta - \beta$ minutes of video data from there.
   - The STB will then temporarily release the partial tap stream for the next $\beta$ 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.

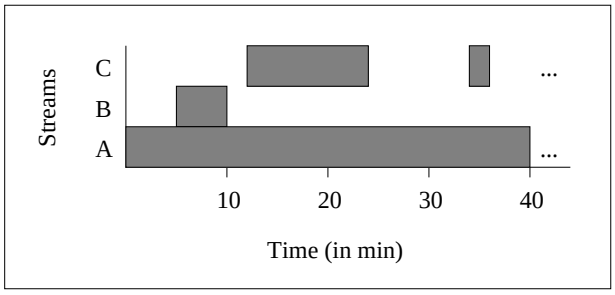


Figure 3: Three types of disk streams: $A$ is an original stream, $B$ is full tap stream, and $C$ is a partial tap stream.

   The partial tap stream will only need to exist for the first $\beta$ minutes of the display and then for the first $\Delta - \beta$ minutes of every succeeding $\Delta$-minute interval. That means it will be busy for

$$B_p(\Delta, i) = \beta + \lfloor \frac{L_i - \beta}{\Delta} \rfloor (\Delta - \beta) + \min(\Delta - \beta, (L_i - \beta) \bmod \Delta) \quad (3)$$

   minutes, where $i$ is the index of the requested video.

Figure 3 gives an example of the three types of disk streams when $\beta = 10$. The original stream starts at time $T_0$, the full tap stream starts at time $T_0 + 5$, and the partial tap stream at time $T_0 + 12$. The shaded area indicate when the streams are busy.

## 2.2 Algorithm

Using the definitions from §2.1, 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.

- If no instance of the requested video is currently being read from disk using an original stream, then that request is assigned an original stream.
- If an original stream for the requested video started less than or equal to $\beta$ minutes in the past, then that request is assigned a full tap stream.

- If an original stream for the requested video started over $\beta$ 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 be assigned.

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 $\lambda_g$, the scheduling rate of streams in the group.

Given $\lambda_g$ and $\Delta$, the VOD server estimates two values:

- $C_g$, the average disk usage of a video group that exists for $\Delta + 1/\lambda_g$ minutes and has a scheduling rate of $\lambda_g$.

- $O_g$, the optimal average disk usage of a video group that exists for less than or equal to $\Delta$ minutes and has a scheduling rate of $\lambda_g$.

$C_g$ is the average usage of the group with the request, and $O_g$ is the best average usage of the group without the request.

The VOD server then requires a parameter to the algorithm, $\alpha$, the *tap limit*. If $C_g \leq \alpha O_g$ 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 check which requests can be serviced.

### 2.3 Other Options

In the main part of the stream tapping algorithm described above, the client STB need only receive at most two disk streams at any one time. If the STB has the capability to receive more than this without sacrificing QOS, then two more options can be used.

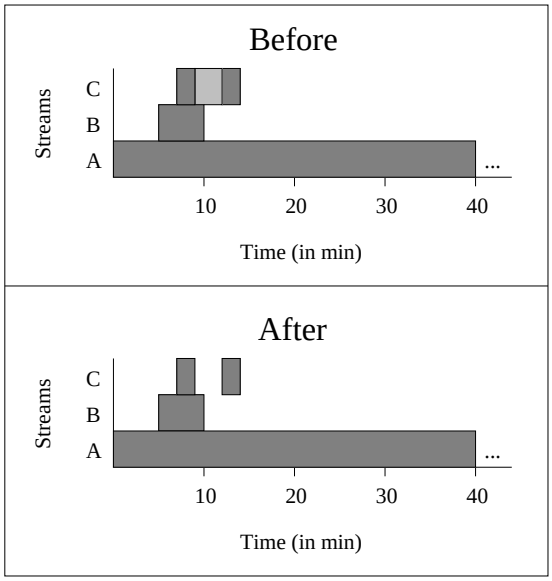


Figure 4: Extra tapping: $A$ is an original stream and $B$ and $C$ are full tap streams to $A$.

The first of these options is called *extra tapping*. This option allows an STB receiving a full or partial tap stream to tap data from any disk stream on the video server, not just from an original stream.

Extra tapping 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.

In other words, the STB is not allowed to undo any positive work or to do any unnecessary work.

An example of extra tapping is shown in Figure 4. 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 (in lighter 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

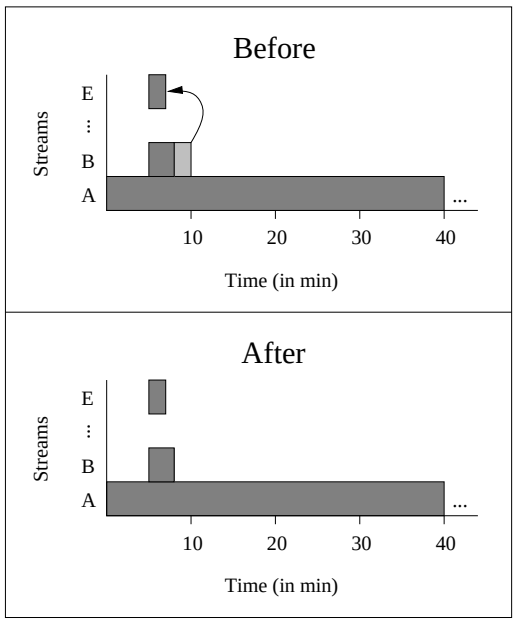


Figure 5: Stream stacking: $A$ is an original stream and $B$ is a full tap stream.

is trying to "catch up," and when it also has extra space in its buffer, it can use whatever disk streams are currently available on the VOD server to help load in the data it needs more quickly. This does not change the amount of time the stream is busy, but it rearranges when data is read from the server, potentially preventing future bandwidth contention.

Figure 5 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 $B$ (in lighter 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.

Although it might not have been obvious from the descriptions above, stream stacking and extra tapping can only be used during the first $\beta$ minutes of full and partial tap streams. For a proof that this extra data does not overflow the client's local buffer, please see Appendix B.

# 3 System Requirements

Stream tapping is more complicated than conventional systems, and because of this it imposes some extra requirements on the hardware components of a VOD service. (These are the same components shown in Figure 1.) Some of the extra requirements are discussed below.

## 3.1 VOD Server

The main difference between a VOD server using stream tapping and one running a conventional scheme is that the stream tapping server requires a stronger software solution.

- The server must be able to reserve disk bandwidth in advance. This also makes disk scheduling more difficult.
- Stream tapping requires a richer protocol between the server and the client. The client must be informed about the identity of the data it receives, and the server must provide that information.
- If extra tapping is used, the server must be able to make quick decisions about available bandwidth, construct and break down short-duration multicast groups, and keep track of which parts of the videos the clients have received.

However, because the software is more complicated, the server will require less hardware, and thus money spent on software development will be mitigated by money saved on hardware. If VOD providers run multiple servers, they should save even more.

## 3.2 Network

Unlike conventional systems, stream tapping requires that the network be able to reserve bandwidth in advance (assuming a network that can

provide QOS assurances is used). This sort of capability can be added to standard reservation protocols, such as RSVP [19] and ST-II [20], with minor modifications [21,22]. Also, because stream tapping reduces the amount of network bandwidth used by the VOD server, it can function on networks with much lower bandwidth capabilities than conventional systems require.

### 3.3 Client STB

In order to perform stream tapping, the client STB must have a local buffer that can store minutes of video data. This buffer does not need to be large. Using MPEG-1 video encoding (at 1.5 Mbps), a 10-minute buffer is only 115 MB; using MPEG-2 encoding (at 4 Mbps), the same buffer is only 300 MB. These small sizes mean the buffer should be relatively cheap to add to the STB. Even the 300 MB buffer should cost far less than $100, and this is not excessive when STB's are expected to have prices similar to VCR's[3].

## 4 Related Work

Other researchers have developed systems for improving the efficiency of VOD servers, and we briefly describe some of these systems below. The systems are distinguished by their most fundamental or unique idea.

### 4.1 Batching

A simple but effective technique for improving VOD server efficiency is known as *batching* [23,24]. 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.

The problem with batching is that it does not attempt to make efficient use of the VOD server's disk bandwidth until the server is in an overloaded state (*i.e.* when it is putting a large percentage of the video requests into its request queue). As long as the server's request queue is small, batching will function essentially the same as a conventional system.

### 4.2 Delayed Batching

Delayed batching [25,26] works much like standard batching, except that instead of servicing a request as soon as possible, the VOD server will wait a certain amount of time (called the *batching interval*) in an effort to increase the number of requests batched together. The batching interval is allowed to change for each video and is usually based on the popularity of the video.

Delayed batching solves the problem found in standard batching in that it is able to batch requests regardless of the load on the VOD server. However, it creates two new problems: it guarantees that the average client latency will be nonzero, and the only way for it to minimize latency is to maximize disk bandwidth (and vice versa).

### 4.3 Staggered Broadcasting

With staggered broadcasting [27–29], a disk stream for a video is only started at regular intervals (such as every ten minutes), and all requests received for the video during the current interval are batched together. This is similar to delayed batching except in how the interval timer is started (a video request versus a regular interval). Thus staggered broadcasting also has the same two problems as delayed batching.

### 4.4 Pyramid Broadcasting

With pyramid broadcasting[4] [30,31], the VOD server reserves a certain number of disk streams for a set of (popular) videos. However, rather than having each stream read out an entire video, the streams read out multiplicatively increasing segments of the videos. The client STB must

[3] In fact, STB's may cost far more than this. Time Warner used a scaled-down Indy workstation, costing over a thousand dollars, in its Orlando trial [3].

[4] For the rest of this report, when we refer to pyramid broadcasting, we will mean the unconstrained, permutation-based version presented by Aggarwal *et al.* [30]. This is the version that produced the lowest average client latency for a given bandwidth.

then jump from stream to stream in order to receive an entire video.

Pyramid broadcasting gives a much better latency than that found in staggered broadcasting (the latency is exponential in the bandwidth rather than linear), but in order for the STB to receive each segment as it is needed, the video data must be transferred at a rate about three times the consumption rate. Because of this, the STB must also have a local buffer, and the size of this buffer must be at least 300 MB (and perhaps as large as 900 MB) for MPEG-1 encoding. This is much larger than the buffer required by stream tapping.

### 4.5 Piggybacking

In piggybacking [32, 33], the display rates of videos are changed by $\pm 5\%$ (little enough so human observers should not notice) so that two existing disk streams can be "merged" into one. That is, the rate of one disk stream can be increased while the rate of another is decreased, and once the streams reach the same position in the video, one can be released.

This strategy works well in that, except for one possible stream jump by the client STB, all of the work is done by the VOD server. However, this gain is heavily offset by the amount of time it takes for two streams to merge; if the streams start $\Delta$ minutes apart, piggybacking will take $10\Delta$ minutes to merge them. This is 10 times longer than it takes stream tapping to achieve a similar effect. Also, if the video rate changes cannot be made on the fly, piggybacking will require at least two versions of each video (one fast and one slow—a normal speed isn't necessarily needed), increasing the storage requirements.

### 4.6 Interval Caching

With interval caching [34,35], the VOD server keeps a local cache of some large size (perhaps in the GB range). When a new video request is to be serviced, the server checks if another disk stream has been assigned for that video, and, if so, attempts to keep the interval of video data between the two display streams in its cache. It does this by placing data in the cache as it is read for the first display stream, leaving it there until the second display stream needs it, and then freeing it after the second display stream has used it. In this way, the second display stream only requires a disk stream for the small amount of time it needs to "catch up" to the cached data (since the other disk stream's data is not placed in the cache until the second display stream starts).

Clearly, the effectiveness of interval caching is dependent on the size of the VOD server's cache, but bigger is not necessarily better. The best size is influenced by the relative costs of disk streams and cache space. For servers of the size used in this report and for MPEG-1 encoding, Dan and Sitaram found that the most cost effective cache size was only 250 to 500 MB; yet even using a cache large enough to hold 8 videos (roughly 10 GB), they only found a disk bandwidth savings of 40–50% over conventional systems [34]. Stream tapping can achieve this for any video with an average interarrival time of 20 minutes or less (see §6). Also, while interval caching is able to save disk streams on the VOD server, it does nothing to save bandwidth on the network. It has the same network requirements as conventional systems.

### 4.7 Asynchronous Multicasting

Asynchronous multicasting [36,37] 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 only takes $S/N$ 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 that it missed.

In particular, the buffer must be able to hold some multiple $M$ of the segment size $S$. This allows the client to join a multicast group as long as the $M^{\text{th}}$ segment has not yet been sent to the

rest of the group, or within $(M-1)S$ minutes of the start of the video. Using an example of $M = 3$ and $S = 6$ [37], this means the client's buffer must be able to hold 18 minutes of video data, but it can only catch up to videos that started less than 12 minutes in the past.

Stream tapping was developed independently of asynchronous multicasting, and although the two systems share some similarities (especially with full tap streams), there are some important differences. Stream tapping:

- 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 STB.

# 5 Simulation

We analyzed the stream tapping system using simulation. Each run of the simulation consisted of a 2-hour warm-up period followed by a 12-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%.

## 5.1 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 truncated to 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. (See Appendix A for details.) This is the distribution recommended by Drapeau *et al.* [38] and used by many others [23, 27, 33, 34, 39]. Also, like many of the papers just cited, we decided to configure the distribution to more closely fit empirical video rental patterns [40], and therefore used $N = 92$ and a $\theta = 0.271$. (See Figure 6.)

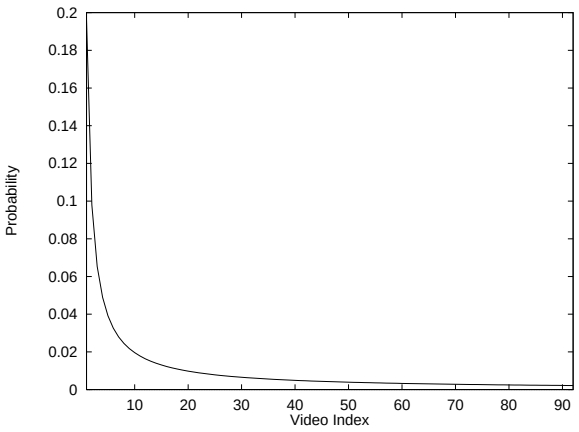


Figure 6: A Zipf-like distribution with $N = 92$ and $\theta = 0.271$.

## 5.2 Clients

Clients were generated using a Poisson arrival process with an interarrival time of $1/\lambda$. Clients were only allowed to select a video based on the distribution described above. Adding more complexity to the client—such as allowing it to give up waiting (or *renege*) on its request, or allowing it to interact with the video—is planned for the future.

## 5.3 VOD Server

Overall, 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 only issue left to model was the amount of time it took the server to access a video. This latency depends on the particular VOD server architecture being used (*e.g.* 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.

### 5.4 Network

The characteristics of a network—its size, bandwidth, protocol, and medium—can drastically influence its effectiveness in supporting VOD. For this reason we did not want to make any assumptions about the network, and as with the VOD server, we also wanted to remove its effects from the results of the simulation. Therefore 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 in this and the previous subsection are certainly unrealistic, they are consistent with the approaches taken by other researchers in the field. Also, even if the latencies caused by video start-up and the network were included, they are small enough when compared to the latency caused by contention for VOD server resources that it is likely the results would not be very different.

### 5.5 Client STB

We assumed that the client STB's were all exactly the same in terms of their buffer size and maximum data rate. This is not required to be the case, but it simplified the simulation.

### 5.6 Scheduling Policy

Stream tapping does not work well with standard scheduling policies. First-come-first-served (FCFS) is fair, but it does not recognize the savings presented by full and partial tap streams. Maximum queue length (MQL), by scheduling based on the current number of outstanding requests for each video, favors the tap streams, but it can cause starvation for unpopular videos. Minimum service time (MST) recognizes the potential for tap streams better than the others, but it can cause starvation as well.

For this reason, we created our own scheduling policy called WSA, the weighted, scaled average of FCFS, MQL, and MST. In this policy, each request receives its normal priority from the three standard policies, but then the values are scaled to put them in common terms, weighted for tuning purposes, and then added together for the final priority.

The exploration of other scheduling policies, and their effect on a stream tapping VOD server, is planned for the future.

## 6 Results

In this section we will present simulation results for stream tapping. These results are for disk bandwidth and latency, our two primary metrics for VOD server efficiency. We will first explore how variations in the parameters and options to the system affect its performance, and then we will compare (where possible) stream tapping to other systems.

We drew upon two standard configurations—one for measuring latency and one for measuring disk bandwidth—when running the simulation. Unless otherwise specified, we used these configurations and both options to the system when simulating results.

The *latency configuration* is one that might exist in real life:

- 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 ($\beta = 10$, $C = 4$).

This represents a moderately-sized VOD server, one that could service a community of several thousand people.

A configuration for measuring disk bandwidth cannot be based on a realistic setting. This is for two reasons: it is more informative to know the bandwidth required by requests for a single video (which can then be extrapolated for multiple videos, if needed), and the measurements should not be influenced by contention for resources. Therefore, the *disk bandwidth configuration* has the same client STB as the latency configuration, but its VOD server has an infinite number of disk streams and only a single video.

Figures 7 and 8 show how the tap limit affects VOD server efficiency. Recall that the tap limit helps decide whether the server should select a partial tap stream or an original stream, with a value of 1.0 meaning it should choose whichever requires the least amount of disk time. Therefore the results in Figure 7 are not surprising; the lowest bandwidth came when the tap limit was 1.0.

Figure 8, which shows the tap limit's effect on latency, is more intriguing. The best limit here came at 1.05, a value that caused the VOD server to favor partial tap streams. We conjecture this happened because partial tap streams cause disk stream "gaps" of size $\beta$, and these allow stream stacking to be performed more often, and they allow full tap streams to be scheduled more often. (We allowed the modeled server to continue checking on requests even after it found one that could not be serviced. Giving extra opportunities for full tap streams causes the scheduling policy to behave more like MST, which reduces the average latency.)

Given the slight disparity between the optimal tap limits for latency and disk bandwidth, we decided to use a tap limit of 1.0 for the rest of the results in this section. This value was the most intuitive choice, and it should work well regardless of the scheduling policy used. We did not want to tune stream tapping to the WSA policy and give it an unfair advantage over the other systems during later comparisons. Also, by using a value greater than or equal to 1.0, we were able to prove the server will never make a bad stream choice. See Appendix C.

Figure 9 shows how the size of the client buffer affects disk usage on the VOD server. As expected, a larger buffer improves the disk usage on the VOD server for all arrival rates except for the largest, $\lambda = 120$. This happens 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

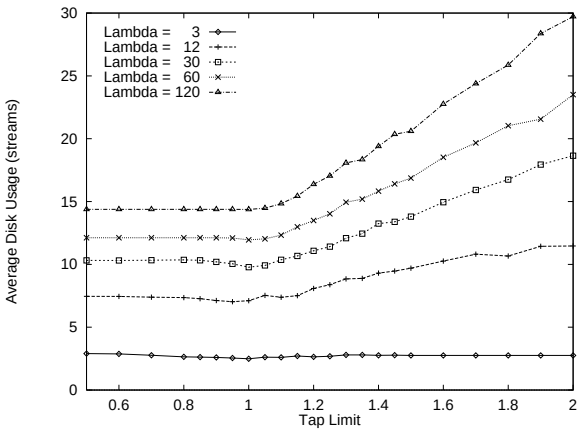


Figure 7: Using disk bandwidth to determine the best tap limit ($N = 1$, $S = \infty$).

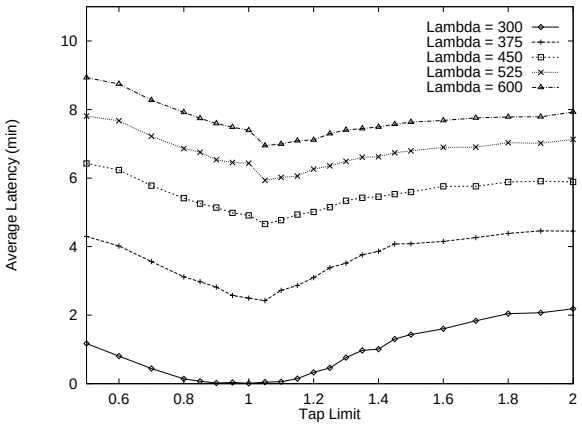


Figure 8: Using latency to determine the best tap limit ($N = 92$, $S = 300$).

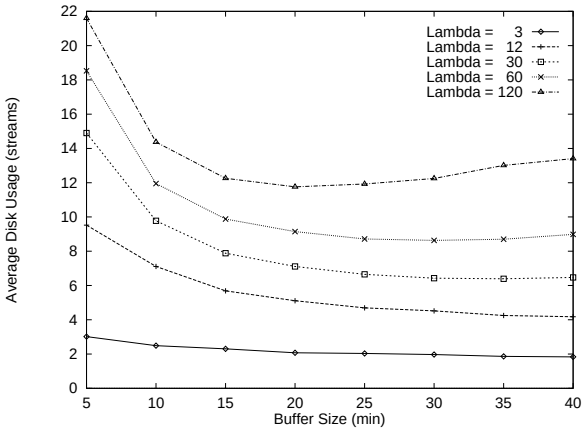


Figure 9: STB buffer size versus disk usage ($N = 1$, $S = \infty$).

it is something we will explore in the future.

Figure 9 also shows that with a 10-minute buffer on the client, the VOD server only requires about 15 disk streams on average to give a zero latency for a particular video, even if the video is requested 120 times per hour. If staggered broadcasting were to use 15 disk streams for a video, it would give a latency of over 3.5 minutes.

The effect the buffer size has on client latency is shown in Figure 10. 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.

It is also worth noting that a conventional system, by reserving a disk stream for each display stream, can only handle about 164 requests per hour when it has 300 disk streams; any more than that and it will begin to generate an infinite request queue. 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 11 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 the arrival rates to be meaningful for each server size.

Figure 9 already provided examples showing that when the request rate for a video is doubled, the bandwidth required by the video does not double. (For example, for $\beta = 10$, when the request rate jumped from 30 to 60 per hour, the bandwidth only increased from about 10 to 12 disk streams.) Thus the results from Figure 11 should not be surprising. Increasing the disk bandwidth and the request rate by the same factor allows the VOD server to perform more efficiently. This is why stream tapping scales well.

The effect of the VOD server's library size on latency is shown in Figure 12. Stream tapping works best when it receives a high rate of requests for a single video; that allows most clients to receive a significant amount of their video data through tapping. Thus we would expect the

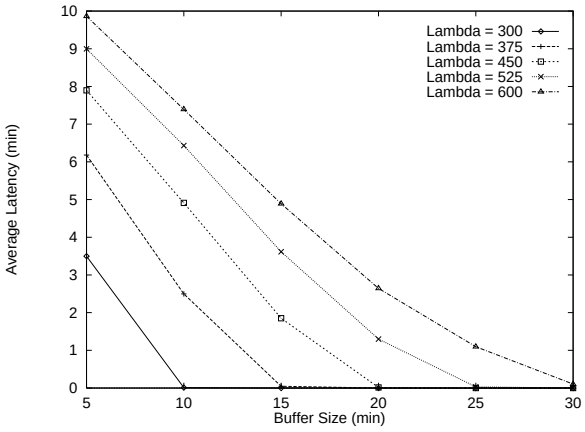


Figure 10: STB buffer size versus latency ($N = 92$, $S = 300$).

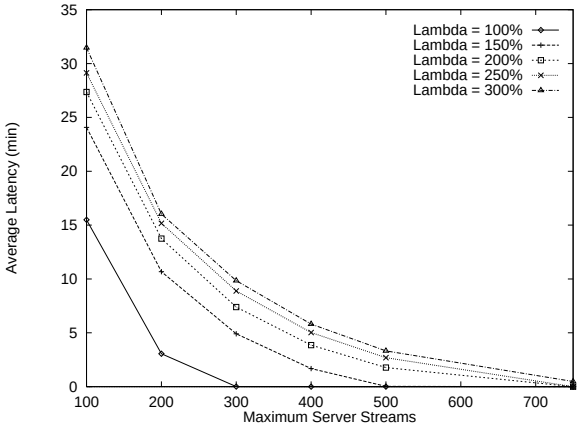


Figure 11: VOD server disk streams versus latency ($N = 92$, $S = 300$).

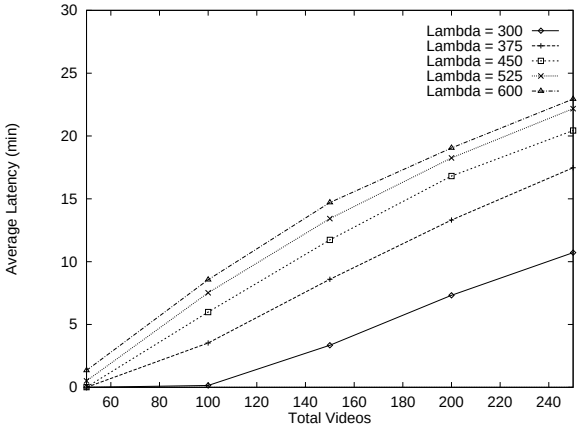


Figure 12: VOD server library size versus latency ($N = 92$, $S = 300$).

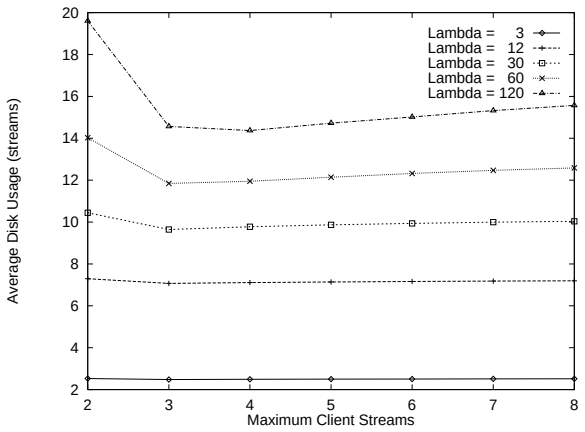


Figure 13: Maximum client streams versus disk bandwidth ($N = 1$, $S = \infty$, $\lambda = 60$).

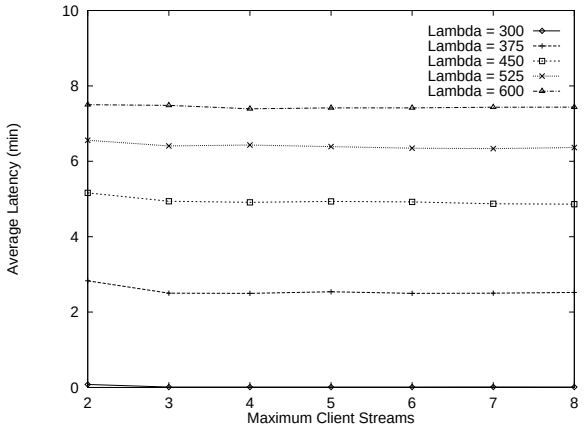


Figure 14: Maximum client streams versus latency ($N = 92$, $S = 300$, $\lambda = 525$).

VOD server to show lower latencies when there are fewer videos (but the request rate remains the same), and that is the case in this figure.

Figures 13 and 14 show how the maximum number of client disk streams affects VOD server efficiency. The basic part of the stream tapping system requires two client streams, and only the options to the system make use of more. Thus we expected *a priori* that the performance gains from increasing the number of client streams would be small, and that is what we found. Both disk usage and latency improve when moving from 2 to 3 client streams, but after that the gains are minimal.

Figure 13 might be a little misleading in that it

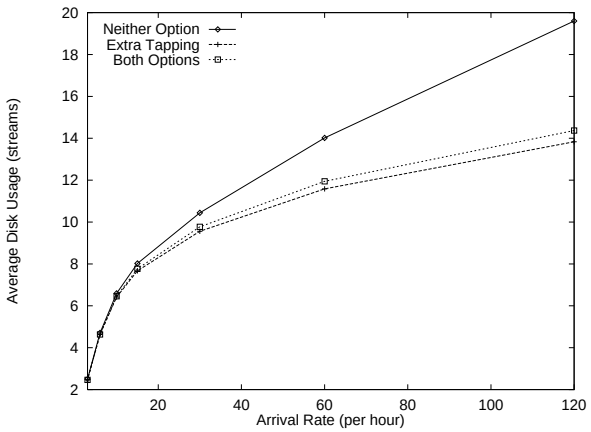


Figure 15: Effects of the stream tapping options on disk usage ($N = 1$, $S = \infty$).

indicates the server uses more bandwidth when the clients can accept more streams. This is because, by allowing the server to have infinite disk streams when measuring disk usage, the clients are able to perform stream stacking to its fullest extent. The more clients can perform stream stacking, the less they can perform extra tapping (since the "stacked" data can no longer be tapped by later-arriving requests), and the less efficient the VOD server becomes with regard to disk bandwidth. This is also a problem with other graphs that measure disk usage, but to a smaller extent since the default value of $C$ is only 4.

Figures 15 and 16 show how the extra tapping and stream stacking options affect the VOD server. Note that stream stacking is not included in Figure 15. When it is used alone, it simply rearranges when video data is read from disk; it does not change the actual disk usage.

These results were slightly disappointing: while the options appear to improve disk usage to a significant extent, they only decreased latency by a slight amount (roughly 20 seconds for each data point). This appears to be caused by two things:

- Extra tapping causes gaps in disk streams that reduce the overall disk usage, but the gaps are not large enough and do not line up well enough to allow other streams to be scheduled.

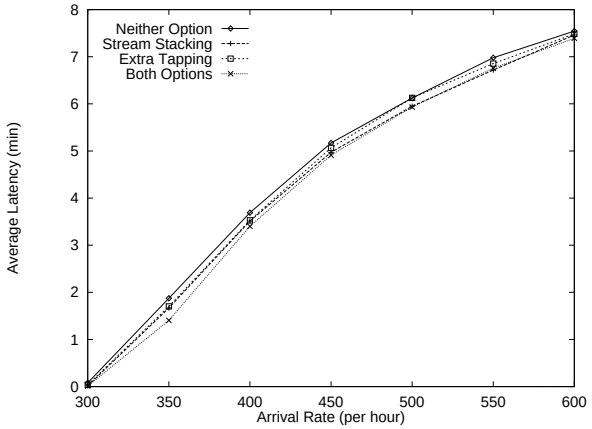


Figure 16: Effects of the stream tapping options on latency ($N = 92$, $S = 300$).

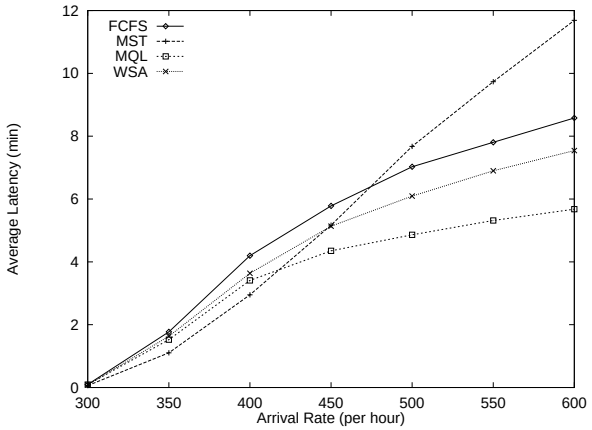


Figure 17: Average latencies for four scheduling policies ($N = 92$, $S = 300$).

- Stream stacking only exists to improve latency (it does not help disk usage), but it requires available streams in order to function. That is, in most of the situations when it can help, it is not needed.

However, it is possible that with a larger VOD server (which gives more disk streams to work with and allows more interactions between streams) or with a different strategy for using the options (such as changing how available streams are split up during stream stacking) the options might perform better.

Figures 17 and 18 show how WSA compared to other scheduling policies like FCFS, MQL, and MST. We found these results heartening: WSA

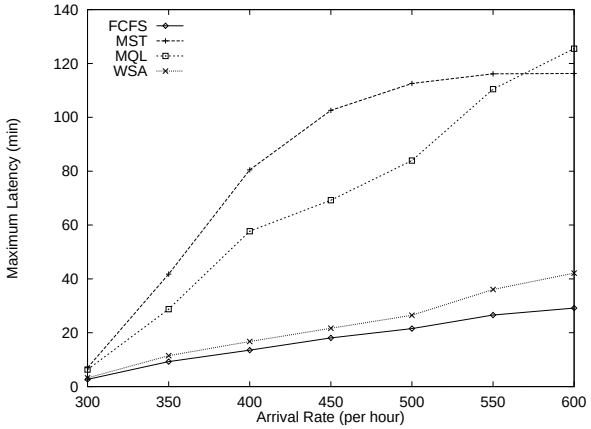


Figure 18: Maximum latencies for four scheduling policies ($N = 92$, $S = 300$).

performed well—with an average latency between FCFS and MQL and a maximum latency close to FCFS—even though we did not take a great deal of time to study different scheduling policies or tune the WSA policy. We expect that with more analysis we will be able to find a scheduling policy that will work even better with stream tapping than WSA does now.

Figure 19 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 creates nonzero 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 20 compares stream tapping to the two broadcasting systems. Because of their deterministic nature, it is 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

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

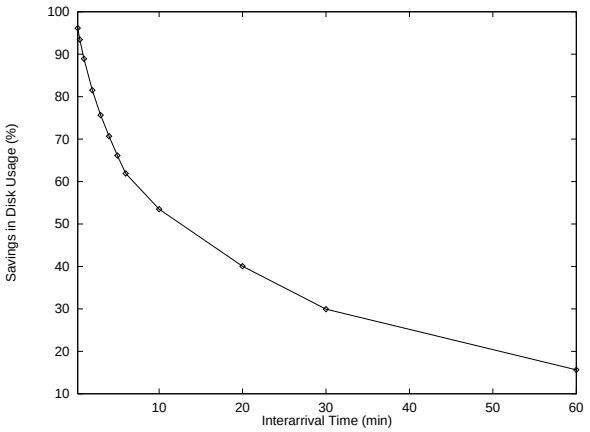


Figure 19: Savings in disk usage: stream tapping over conventional systems ($N = 1$, $S = \infty$).

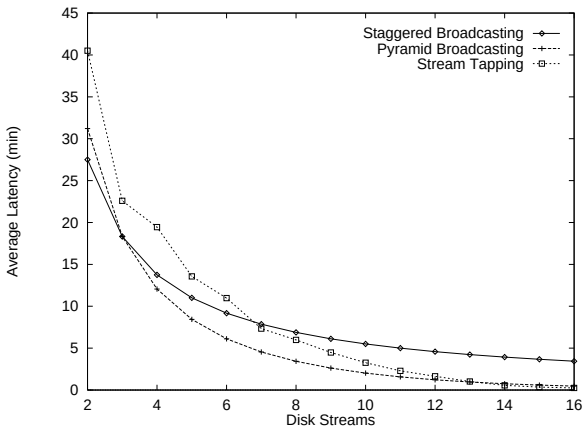


Figure 20: Average latency for three VOD systems ($N = 1$, $\lambda = 360$).

for pyramid broadcasting [30]. Note that even with the high arrival rate (a request every ten seconds) stream tapping was very competitive and outperformed both broadcasting systems given enough disk streams.

Figures 21 and 22 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 should provide 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 21 compares the three systems using disk bandwidth. This is not particularly fair to batching; by allowing the VOD server unlimited disk streams to measure usage without contention, the server never had any requests in its queue, and thus batching in this case performed exactly the same as a conventional system. In both Figures 21 and 22, stream tapping handily outperforms the other systems.

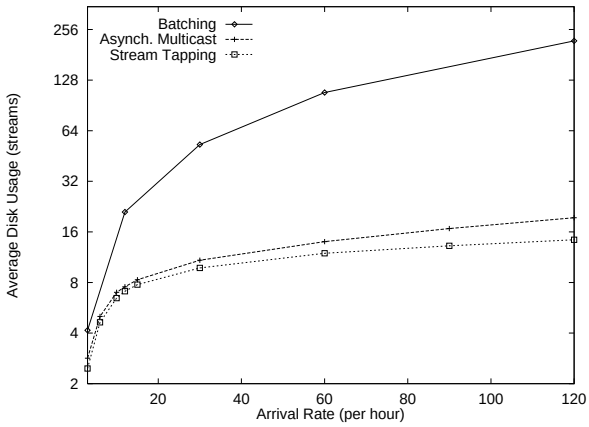


Figure 21: Average disk usage for three VOD systems ($N = 1$, $S = \infty$).

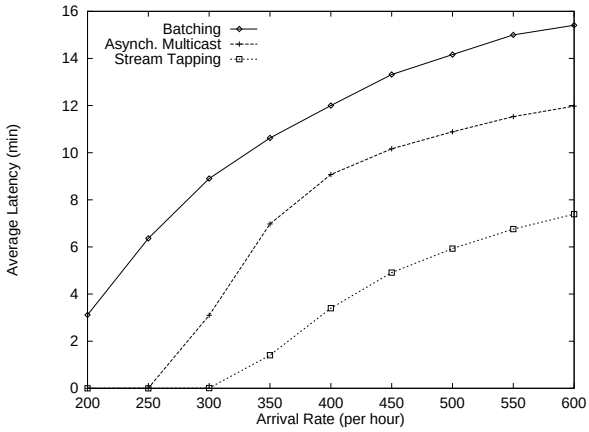


Figure 22: Average latency for three VOD systems ($N = 92$, $S = 300$).

# 7 Future Work

There are a variety of topics we have not yet explored at all, or have not explored as fully as we should. Some of the more important topics are listed in the subsections below.

### 7.1 Scheduling Policies

As we mentioned before (see §5.6), our WSA scheduling policy is very simplistic and can more than likely be improved upon. Others have looked at scheduling policies for VOD servers but have not considered the service time of the request [23, 24]. This is a very important factor with stream tapping since service times can range from a few seconds to hundreds of minutes.

### 7.2 Client Reneging

There are three important ways of rating a scheduling policy:

- Average latency, which we have discussed in this report.
- Client reneging rate, which is the percentage of time clients renege on their video request because it has taken too long to be serviced.
- Fairness, which is a measure how "fair" the scheduling policy is in regard to the different videos offered by the VOD server. A perfectly fair policy has an equal reneging rate for each video.

In this report, we have only considered the first of these three methods (while giving hints about the latter two through the use of the maximum latency). By adding client reneging to the simulation, we will be able to study the reneging rate and fairness as well.

### 7.3 Stream Decision Process

While working with stream tapping, we found there were situations when it was not always best to assign a request a full tap stream. For example, if $\beta = 30$, $L = 110$, and a disk stream is started for a video every minute, then a video group will contain an original stream and 30 full tap streams, and it will require

$$110 + \sum_{j=1}^{30} j = 575$$

minutes of disk time. However, if an original stream was used instead of the 15th full tap stream from above, the same requests would only require

$$110 + \sum_{j=1}^{14} j + 110 + \sum_{j=1}^{15} j = 445$$

minutes of disk time. Changing the decision process so that an original stream can be assigned at any time is an important step in making stream tapping more versatile.

### 7.4 Server Library Size

We chose a library size of 92 videos because that allowed us to model our workload against empirical data [40]. This size is probably much smaller than will actually be used: Drapeau *et al.* [38] estimate that VOD servers will contain around 500 (MPEG-1 encoded) videos, while some real life VOD trials feature servers that can hold over 1000 videos [1, 2]. Large library sizes are a good selling point for VOD providers, but it is not clear to what extent they affect VOD server simulations. Even with our distribution, the 92nd video is only selected 0.4% of the time. If hundreds of more videos are added, they will all have very small selection probabilities, and it may turn out that during a 12-hour simulation, not many more than 92 videos are selected anyway. However, this is something we plan to explore.

### 7.5 VCR Control

VCR control includes functions such as pause, fast forward, and rewind. Systems that start videos after semi-regular intervals (such as staggered broadcasting) can perform these functions very efficiently in a *discontinuous* manner [25, 27, 28]. That is, the functions can be mimicked by allowing the client STB to jump forward a stream (for fast forward), jump backward a stream (for rewind), or wait for a trailing stream to reach the same point in the video (for pause). However, this implementation does not allow for *cuing* (viewing the video while using fast forward or rewind) and it might not give

the client much precision in deciding the duration of the function (depending on how often new streams are started). And thus *continuous* VCR functions are preferred.

Systems such as stream tapping, piggybacking, and batching can support continuous VCR functions, but as far as we have been able to find, no such studies have been performed. This is probably because the systems will encounter serious degradation in performance when VCR controls are added: each VCR action taken by a client can potentially move the client out of its current multicast group and into a new group where it is the only member. This would, in effect, break down all the work the system went through to create a large multicast group.

But stream tapping might have an edge:

- A client moved out of its current group would still be able to tap data from other disk streams. This might allow it to be assigned the equivalent of a full or partial tap stream and save on disk bandwidth.

- Because stream tapping requires a buffer on the client STB, not all VCR functions would require the client to leave its group. For example, a client receiving an original stream can pause up to $\beta$ minutes before it must jump to a new group.

These points will help to mitigate the effects of VCR actions, but obviously they will not alleviate them entirely. However, a study must be done to see how much the VCR controls will hurt stream tapping performance, and how this interactive version of stream tapping will rate compared to other systems providing VCR controls.

# 8 Conclusion

Efficiency is very important for VOD servers; it reduces the amount of hardware a server requires in order to function, and it reduces the amount of time clients must wait before their requests can be serviced. The latter might be enough to determine whether a VOD service succeeds or fails.

In this report we presented a system called stream tapping that can improve the efficiency of VOD servers. It allows clients to tap into all disk streams on the VOD server so they can minimize the amount of new disk bandwidth they require for their requests. This reduces the total amount of bandwidth required by the server, and that in turn leads to lower latencies for the clients.

Stream tapping does not make any assumptions about its environment. It can work with VOD servers of any size, does not require excessive network bandwidth, and can be scaled to fit the desired complexity of the VOD server or client STB. Stream tapping also does not require any *a priori* knowledge by the VOD provider, such as which videos will be popular and which will not.

We tested stream tapping through the use of simulation. Using MPEG-1 encoding, videos with an average length of 110 minutes, and a VOD server with 300 disk streams available, we found that even when the client buffer was as small as 115 MB stream tapping gave less than 5-minute latencies even when as many as 450 requests were made each hour. We also compared stream tapping to a variety of other VOD systems and found that it performed as well or better than all of them. In particular, it required less than 20% of the disk bandwidth used by conventional systems (which dedicate a disk stream on the server for each client) for popular videos.

# Appendix

## A

Given a parameter $N$, the number of objects to be considered, a Zipf distribution assigns object $i$ the frequency

$$f(i) = \frac{1}{i}$$

Then the probability for object $i$ is given by

$$p(i) = \frac{f(i)}{\sum_{j=1}^{N} f(j)}$$

A Zipf-like distribution modifies the above slightly by using a second parameter $\theta$. With $\theta$ the frequency for object $i$ becomes

$$f(i) = \frac{1}{i^{(1-\theta)}}$$

and the probability for $i$ is calculated in the same fashion.

# B

When a client STB is assigned a full or partial tap stream, it will receive data from

- Its assigned stream,
- The original stream it is tapping, and
- Any streams it can use for stacking or extra tapping

during its first $\Delta$ (for full taps) or $\beta$ (for partial taps) minutes. We will use this appendix to prove that the client's buffer can hold all of this data without overflowing.

Let $d$ be the length of the interval ($\Delta$ or $\beta$), and note that $d \leq \beta$. Also note that the original stream being tapped is sending data that the STB buffer must hold for the entire $d$ minutes, but the other streams are only sending data for the first $d$ minutes of the video. Now consider any time $t$ between $T_0$, the starting time of the tap stream, and $T_0 + d$. The client buffer will have $t$ minutes of data from the original stream in its buffer, but it will only have at most $d-t$ minutes of data from the other sources. That means the buffer will have at most $d - t + t = d \leq \beta$ minutes of video data in it during the time interval. And so the variety of sources will not cause the STB buffer to overflow.

# C

When the stream tapping algorithm decides between a partial tap stream and an original stream, it does not consider the effect of future streams on the current video group. It makes a simple greedy decision based on the optimal average usage for the group—up until the time of the current request—and the average usage for the group if the request is assigned a partial tap stream. That leaves open the possibility that the algorithm could make a bad choice. That is, it is possible for the algorithm to decide on an original stream when a series of future partial tap streams would bring the average usage within the tap limit of the optimal usage. In this appendix we will prove that when the tap limit is greater than or equal to 1.0, the algorithm never makes a bad choice.

We will first define some notation. Let $\alpha$ be the tap limit, let $\delta = 1/\lambda_g$ be the interarrival time for streams for the video, and let $U_{i,j}$ be the total usage for the $i^{\text{th}}$ through $j^{\text{th}}$ streams to the video group. (So, for example, $U_{1,1}$ is the usage of the original stream in the group.) Let $r$ be the index of the stream that causes the optimal average usage, and let $s > r$ be the index of the first request that the algorithm assigns an original stream.

Then we know

$$\frac{U_{1,s}}{s\delta} > \alpha \frac{U_{1,r}}{r\delta} \tag{4}$$

and we need to prove that for any $t > s$,

$$\frac{U_{1,t}}{t\delta} > \alpha \frac{U_{1,r}}{r\delta}$$

Let us first look at the average group usage for the $s^{\text{th}}$ request (assuming it was assigned a partial tap stream rather than an original stream).

$$\begin{aligned}\frac{U_{1,s}}{s\delta} &= \frac{U_{1,r} + U_{r+1,s}}{s\delta} \\ &= \frac{U_{1,r}}{r\delta} \cdot \frac{r}{s} \\ &\quad + \frac{U_{r+1,s}}{(s-r)\delta} \cdot \frac{s-r}{s}\end{aligned}$$

The average usage is simply the weighted average of the optimal average usage and the average usage of the streams arriving after $r$. Hence, by Equation 4 we know

$$\frac{U_{r+1,s}}{(s-i)\delta} > \alpha \frac{U_{1,r}}{r\delta}$$

But

$$\frac{U_{r+1,s}}{(s-i)\delta} = \frac{1}{(s-r)}\left(\frac{U_{r+1,r+1}}{\delta} + \frac{U_{r+2,r+2}}{\delta} + \cdots + \frac{U_{s,s}}{\delta}\right)$$

which is a weighted average as well. Since the usage of each request increases the farther away the request gets from the original stream (*i.e.* with increasing index), we have

$$\frac{U_{s,s}}{\delta} > \alpha\frac{U_{1,r}}{r\delta} \tag{5}$$

Now consider any $t > s$. The average usage of the video group is

$$\begin{aligned}
\frac{U_{1,t}}{t\delta} &= \frac{U_{1,s} + U_{s+1,t}}{t\delta} \\
&= \frac{U_{1,s}}{s\delta}\cdot\frac{s}{t} \\
&\quad + \frac{U_{s+1,t}}{(t-s)\delta}\cdot\frac{t-s}{t} \\
&> \alpha\frac{U_{1,r}}{r\delta}\cdot\frac{s}{t} \\
&\quad + \frac{(t-s)U_{s,s}}{(t-s)\delta}\cdot\frac{t-s}{t} \qquad \text{by (4)} \\
&> \alpha\frac{U_{1,r}}{r\delta}\cdot\frac{s}{t} \\
&\quad + \alpha\frac{U_{1,r}}{r\delta}\cdot\frac{t-s}{t} \qquad \text{by (5)} \\
&= \alpha\frac{U_{1,r}}{r\delta}
\end{aligned}$$

QED.

# 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] Bruno Suard, Leopold Verbist, and Dirk De Schoenmacker. Update on VOD trials. In *Proceedings of ICCT '96*, pages 1033–6, Beijing, China, May 1996. IEEE Computer Society Press.

[3] Tekla S. Perry. The trials and trevails of interactive TV. *IEEE Spectrum*, 33(4):22–8, April 1996.

[4] David Tobenkin. Customers respond to video on demand. *Broadcasting & Cable*, 123(48):16, November 1993.

[5] Susan T. Whitehead. Time Warner Cable's Full Service Network—program management of the FSN virtual organization. In *Proceedings of the 2nd International Workshop on Community Networking Integrated Multimedia Services in the Home*, pages 291–9, Princeton, NJ, USA, June 1995. IEEE Computer Society Press.

[6] Milind M. Buddhikot and Gurudatta M. Parulkar. Efficient data layout, scheduling and playout control in MARS. In *Proceedings of the 5th International Workshop on Network and Operating System Support for Digital Audio and Visual*, pages 318–29, Durham, NH, USA, April 1995. Springer.

[7] Roger L. Haskin and Frank B. Schmuck. The Tiger Shark file system. In *Proceedings of COMPCON '96*, pages 226–231, Santa Clara, CA, USA, February 1996. IEEE Computer Society Press.

[8] Kunihiro Taniguchi, Hitoya Tachikawa, Takeshi Nishida, and Hiroshi Kitamura. Internet video-on-demand system architecture: MINS. *IEICE Transactions on Communications*, E79–B(8):1068–75, August 1996.

[9] Shunichiro Nakamura, Harumi Minemura, Tomohisa Yamaguchi, Hiroshi Shimizu, Takashi Watanabe, and Tadanori Mizuno. Distributed RAID style video server. *IEICE Transactions on Communications*, E79-B(8):1030–8, August 1996.

[10] Huib Eggenhuisen and Sjir Van Loo. Video-on-demand server. *Philips Journal of Research*, 50(1–2):201–8, 1996.

[11] Alec Livingstone. BT interactive TV. In *Proceedings of the 3rd International Workshop on Community Networking*, pages 111–5, Antwerpen, Belgium, May 1996. IEEE Computer Society Press.

[12] Kiyoshi Kohiyama, Hideaki Shirai, Kiyotaka Ogawa, Akio Manakata, Yuzuru Koga, and Masayuki Ishizaki. Architecture of MPEG-2 digital set-top-box for CATV VOD system. *IEEE Transactions on Consumer Electronics*, 42(3):667–72, August 1996.

[13] Qiang Tan, Mengchu Zhou, Jingjian Li, and Dingkang Yao. A brief overview of current TV set-top box developments. In *IEEE International Conference on Systems, Man and Cybernetics*, pages 2127–32, New York, NY, USA, October 1996. IEEE Computer Society Press.

[14] Ajith N. Nair. Interactive television set-top terminal architectures. In *Proceedings of COMPCON '96*, pages 233–8, Santa Clara, CA, USA, February 1996. IEEE Computer Society Press.

[15] Jean M. McManus and Keith W. Ross. Video on demand over ATM: constant-rate transmission and transport. *IEEE Journal on Selected Areas in Communications*, 14(6):1087–98, August 1996.

[16] Jian Ni, Tao Yang, and Danny H. K. Tsang. CBR transportation of VBR MPEG-2 video traffic for video-on-demand in ATM networks. In *IEEE International Conference on Communications*, pages 1391–5, Dallas, TX, USA, June 1996. IEEE Computer Society Press.

[17] Gert Van der Plas, Raf Smets, Bruno Suard, and Willem Verbiest. Demonstration of an ATM-based passive optimal network in the FTTH trial in Bermuda. In *Proceedings of GLOBECOM '95*, pages 988–92, Singapore, November 1995. IEEE Computer Society Press.

[18] Koichi Shiga. ATM technology meets VOD systems requirements. *Journal of Electronic Engineering*, 33(352):28–9, April 1996.

[19] Lixia Zhang, Stephen Deering, Deborah Estrin, Scott Shenker, and Daniel Zappala. RSVP: a new resource reservation protocol. *IEEE Network*, 7(5):8–18, September 1993.

[20] C. Topolcic. Experimental internet stream protocol, version 2 (ST-II). RFC 1190, October 1990.

[21] Mikael Degermark, Torsten Köhler, Stephen Pink, and Olov Schelén. Advance reservations for predictive service. In *Proceedings of the 5th International Workshop on Network and Operating System Support for Digital Audio and Visual*, pages 3–15, Durham, NH, USA, April 1995. Springer.

[22] W. Reinhardt. Advance reservation of network resources for multimedia applications. In *Second International Workshop on Advanced Teleservices and High-Speed Communication Architectures*, pages 23–33, Heidelberg, Germany, September 1994. Springer-Verlag.

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

[24] 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.

[25] Victor O. K. Li, Wanjiun Liao, Xiaoxin Qiu, and Eric W. M. Wong. Performance model of interactive video-on-demand systems. *IEEE Journal on Selected Areas in Communications*, 14(6):1099–109, August 1996.

[26] Hadas Shachnai and Philip S. Yu. The role of wait tolerance in effective batching: A paradigm for multimedia scheduling schemes. Technical Report RC 20038, IBM Research Division, T.J. Watson Research Center, April 1995.

[27] Asit Dan, Perwez Shahabuddin, Dinkar Sitaram, and Don Towsley. Channel allocation under batching and VCR control in video-on-demand systems. *Journal of Parallel and Distributed Computing*, 30(2):168–79, November 1995.

[28] 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.

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

[30] 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.

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

[32] 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.

[33] 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.

[34] Asit Dan and Dinkar Sitaram. Buffer management policy for an on-demand video server. Technical Report RC 19347, IBM Research Division, T.J. Watson Research Center, January 1993.

[35] Mohan Kamath, Krithi Ramamritham, and Don Towsley. Continuous media sharing in multimedia database systems. Technical Report 94-11, University of Massachusetts, 1994.

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

[37] 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.

[38] 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.

[39] Kevin C. Almeroth and Mostafa H. Ammar. The role of multicast communication in the provision of scalable and interactive video-on-demand service. In *Proceedings of the 5th International Workshop on Network and Operating System Support for Digital Audio and Visual*, pages 251–4, Durham, NH, USA, April 1995. Springer.

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