Title

Improving bandwidth efficiency of video-on-demand servers1This extends work previously reported at Sixth International Conference on Computer Communications and Networks [3].1

Journal

Computer Networks, 31(1-2)

ISSN

1389-1286

Authors

Carter, Steven W
Long, Darrell DE

Publication Date

1999

DOI

10.1016/s0169-7552(98)00233-5

Peer reviewed

ELSEVIER

Computer Networks 31 (1999) 111–123

COMPUTER NETWORKS

Improving bandwidth efficiency of video-on-demand servers [1]

Steven W. Carter, Darrell D.E. Long [*,2]

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

Abstract

Video-on-demand (VOD) servers have a limited amount of bandwidth with which to service client requests. Conventional VOD servers dedicate a unique stream of data for each client, and that strategy can quickly allocate all of the available bandwidth on the server. We describe a system called stream tapping that allows clients to ''tap'' into existing streams on the VOD server. By using existing streams as much as possible, clients can reduce the amount of new bandwidth they require, and that allows more clients to use the server at once, reducing client latency. Stream tapping uses less than 20% of the bandwidth required by a conventional VOD server for popular videos, and it performs better than many other strategies designed to improve VOD servers. © 1999 Elsevier Science B.V. All rights reserved.

Keywords: Video-on-demand; Efficiency; Bandwidth

1. Introduction

At some point in the future, video-on-demand (VOD) will allow clients to turn on their television sets, connect to a VOD server by way of a television set–top box (STB), select a video from the server's video library, and then begin viewing the video instantaneously.

That point is still some time off. Although numerous companies have conducted VOD trials and sponsored market tests that show the public is very receptive to the idea of VOD [12,13], these companies have not been able to create a commercially available system. The reason, almost unanimously, is cost. Time Warner, for example, spent over $250 million on its Full Service Network (which provided VOD and more) before ending the project because the system just was not yet economically viable [7].

So while it is always important to improve efficiency, it is critical with VOD. Strategies that can make existing hardware more efficient and reduce the need for additional hardware (and cost) may be enough to help VOD companies succeed where they are currently failing.

One such strategy is called *stream tapping* [3], and it improves the bandwidth efficiency of a VOD server. Stream tapping accomplishes this by allowing clients to ''tap'' into any stream of data on the VOD server that is displaying data the client can use. By using existing streams as much as possible, clients require their own stream of data for shorter period of time, making them less of a burden to the VOD server, and the existing streams are able to service more clients, making them more efficient. However,

* Corresponding author. E-mail: darrell@cs.ucsc.edu.

[1] This extends work previously reported at Sixth International Conference on Computer Communications and Networks [3].

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

PII: S0169-7552(98)00233-5

since client STB's will be able to receive data out of order, they will require a small amount of buffer space in which to store the data until it is needed. This buffer space need not be large, and it should not add greatly to the price of the STB.

A *conventional system*, on the other hand, does not make efficient use of the VOD server. In fact, it does not allow any sharing of data between clients; each client simply receives its own unique stream of data, and this strategy can quickly allocate all of the streams available on the VOD server.

When measuring efficiency, we will use two metrics:

- *Server bandwidth:* the average number of streams used by the VOD server. Alternatively, we may use *average service time* (AST) which is the average bandwidth, in streams, used by the VOD server multiplied by the amount of time the server has been running and divided by the number of clients.
- *Client latency:* the average amount of time a client must wait before its request is serviced by the VOD server. In some ways this is the most important metric of the two because it is the only one clients will see.

We classify VOD servers in two ways. A server provides *true VOD* if it never delays a client request when it has the bandwidth available to service it. If the server delays requests, then it provides *near VOD*. A server can also provide *interactive VOD* if it allows clients to manipulate the display of the video through the use of VCR controls such as pause, fast forward, and rewind. We have not yet modified stream tapping to support interactivity, but it is a topic we plan to pursue in the future.

A VOD server can also be *reactive* or *proactive*. If the server requires some amount of a priori information about the videos and their popularity then it is called proactive. For example, with some systems the service provider must reserve a specific amount of bandwidth in advance for popular videos. That means the provider must know which videos are going to be popular and how much bandwidth should be reserved for each. If no information about the videos is required, the server is called reactive.

We contend that true VOD is better than near VOD, and a reactive system is better than a proactive one. These classifications represent, respectively, the ideal nature of VOD and the simplest form of maintenance. Stream tapping is a true reactive VOD system.

2. Related work

Over the past five years, several groups of researchers have presented systems for improving the bandwidth utilization and response times for VOD servers. In the following sections, we will briefly describe some of these systems. The key points of these systems are summarized in Table 1.

Table 1
A summary of key points for several efficiency-improving systems

System	True versus near VOD	Proactive versus reactive	Client bandwidth (streams)	Client buffer size (minutes)	Saves on network bandwidth
Conventional	True	Reactive	1	0	No
Batching	True	Reactive	1	0	Yes
Delayed batching	Near	Reactive	1	0	Yes
Staggered broadcasting	Near	Proactive	1	0	Yes
Pyramid broadcasting	Near	Proactive	2–3	30	Yes
Skyscraper broadcasting	Near	Proactive	3	5–40	Yes
Harmonic broadcasting	Near	Proactive	4–6	40	Yes
Piggybacking	True	Reactive	1	0	Yes
Interval caching	True	Reactive	1	0	No
Asynchronous multicasting	Near	Reactive	3 +	10–30	Yes
Stream tapping	True	Reactive	2–4	10–30	Yes

2.1. Batching

With *batching* [6], when the server has multiple requests for the same video in its request queue, it can service them all (i.e. batch them together) by multicasting the video to all of the requesting clients. If there are M requests in the queue, the VOD server will save $M-1$ times the bandwidth of the video.

Note that if there is only one request for a video in the server's request queue, the server will not save any bandwidth. The only way for batching to gain any savings is for the VOD server to allow its request queue to grow, and that only happens when the server has allocated all of its bandwidth. The rest of the time batching will perform exactly the same as conventional systems.

2.2. Delayed batching / staggered broadcasting

Delayed batching [14] and *staggered broadcasting* [2,4] are similar strategies that attempt to fix the problem with batching. Both allow the VOD server to delay a certain amount of time after receiving a request in order to increase the size of the multicast group. With delayed batching, the server starts delaying after a request is received, and the length of time it delays is usually based on the arrival rate of the requests. With staggered broadcasting, the server simply starts streams for a video after fixed intervals of time, and the delay is dependent on when the request arrives during the current interval.

The problem with delayed batching and staggered broadcasting is that, while they improve the efficiency of the server's bandwidth, they force a latency penalty on the client in order to do so. In fact, the average client latency is guaranteed to be non-zero with these strategies. Delayed batching and staggered broadcasting are also only used with popular videos since unpopular videos are unlikely to receive multiple requests during the (short) delay intervals.

2.3. Segmentized broadcasting

Segmentized broadcasting systems include *Pyramid broadcasting* [15], *Skyscraper broadcasting* [9], and *Harmonic broadcasting* [10]. These systems do not broadcast an entire video on a stream; they break up each video into segments and then show each segment repeatedly on a stream. The differences in the systems relate to the sizes of the segments, the number of segments clients must receive at any one time, and the data transfer rates used by the VOD server.

While the segmentized broadcasting systems can significantly reduce client latencies over staggered broadcasting systems, they require a more complicated client STB. The STB must, for example, be able to switch from stream to stream in order to receive all of the segments, be able to receive data at two or more times the consumption rate for the video, and contain a local buffer able to hold up to half of the video data. Moreover, these systems still guarantee a non-zero client latency and are only suitable for popular videos.

2.4. Piggybacking

In *piggybacking* [1,8], the display rates of videos are changed by $\pm 5\%$ (an amount supposedly undetectable to human observers) so that two existing data streams can be ''merged'' into one. Once the two streams reach the same point in the video, one of the streams can be released and its clients switched to the surviving stream.

There are two main drawbacks to piggybacking. The first is that it takes a long time for two streams to merge. If the streams start Δ minutes apart, it will take piggybacking 10Δ minutes to merge them. The second drawback is that, unless the VOD server is powerful enough to make the video rate changes on the fly, it will have to store at least two versions of each video, doubling the storage requirements.

2.5. Interval caching

With *interval caching* [5], the VOD server uses a local cache for its storage system. Inside the cache, as much as possible, the server stores intervals of data between pairs of streams for the same video. This strategy allows the second stream of the pair to get almost all of its data from the cache (which is being fed by the first stream of the pair), saving the server almost a stream of bandwidth.

The main issue with interval caching is its cost effectiveness. The VOD provider has a choice of using a cache and saving bandwidth, or adding more storage devices and adding bandwidth. Dan and Sitaram did a cost analysis and found for the VOD

servers of the size discussed here (300 streams), the cache should only be large enough to hold 20–40 minutes of video data [5]. That will only provide savings for approximately 5–10% of the requests. Note also that interval caching does nothing to reduce the bandwidth of the network. Cached streams on the VOD server will still need to be transported across the network just like normal streams.

2.6. Asynchronous multicasting

Asynchronous multicasting [16,11] 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 display rate of the video so the transfer only takes S/N minutes. That 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 it missed.

Due to the high transfer rate of the segments, clients can only receive one segment of the video at a time. That means a client must arrive before the Nth segment of a video is shown—or within $(N-1)S$ minutes of the start of the video—if it wants to join the multicast group. Using an example of $N=3$ and $S=6$ [11], the client's buffer must be able to hold 18 minutes of video data, but the client can only catch up to videos that started less than 12 minutes in the past. Also, if the client arrives while another segment is being shown, it must wait for a free slot, adding latency.

Asynchronous multicasting shares some similarities with stream tapping (see Section 3.2), but there are also some key 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, never delays requests, and requires a lower data rate at the client STB.

3. Stream tapping

The key idea behind stream tapping is that clients are not restricted to their assigned stream. If other 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 streams as much as possible, clients minimize the amount of new bandwidth they require.

3.1. Notation

We use the following notation when describing the parameters and workload characteristics of the system:

β = the size of the STB buffer, measured in minutes of video data,

B_c = the maximum bandwidth of the client STB, measured in streams,

B_s = the maximum bandwidth of the VOD server, measured in streams,

N = the number of videos offered by the VOD server,

L_i = the length of video i, in minutes, for $1 \le i \le N$,

λ = the arrival rate of requests at the VOD server, measured in requests per hour,

Δ_i = the amount of time that has passed, in minutes, since video i required a stream of data for the entire length of the video.

3.2. Stream classifications

Stream tapping classifies streams into three different types:

- *Original streams* read the entire video from the VOD server's storage system. They can be used at any time, but since they do not use data from other streams, they do not provide any savings for the VOD server. Stream tapping avoids using them when possible.
  Since they read out the entire video, original streams have a service time of
  $$S_o(i) = L_i \quad (1)$$
  minutes, where i is the index of the video being read.
- *Full tap streams* can only be used within β minutes of the start of an original stream for the

same video (i.e. when $\Delta_i \le \beta$). This proximity allows full tap streams to work in conjunction with original streams so they can be released after a very short period of time.

In particular, the requesting STB will receive both the full tap and original streams for Δ_i minutes. During that time the full tap stream will read the first Δ_i 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 Δ_i 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 Δ_i-minute window of the video.

Fig. 1 illustrates the two stages an STB goes through when it is assigned a full tap stream. Part (A) represents the STB when it is receiving the full tap and original streams, and Part (B) represents the STB after the full tap stream has been released. The buffer here is simply a block of storage on the STB that can store β minutes of video data. The shaded part of the buffer indicates where there is data the STB still needs.

Since full tap streams can be dropped after Δ_i minutes, they have a service time of

$$S_f(i) = \Delta_i \tag{2}$$

minutes.

- *Partial tap streams* can be used when an original stream for the same video is active but started over β minutes in the past (i.e. when $\beta < \Delta_i \le L_i$). Like full tap streams, partial tap streams can work in conjunction with original streams, but they provide less savings and that savings quickly diminishes the farther away the partial tap stream gets from the start of the original stream.

  In particular, the STB will receive the partial tap stream and the original stream for β minutes. The partial tap stream will read the first β minutes of the video, and the STB will display it live. Meanwhile, the original stream will read minutes Δ_i to $\Delta_i + \beta$ of the video, and the STB will place that data in its buffer.

  From that point on, the STB will alternate between the following two steps until the video is complete.

  1. The STB will stop receiving the original stream and reacquire (if necessary) the partial tap stream. The STB's buffer will be full at this point, but the data in it will be $\Delta_i - \beta$ minutes away from the STB's current position in the video, so the STB will use the partial tap stream to receive the data it needs.
  2. The STB will release the partial tap stream and begin displaying the data from its buffer. At the same time it will begin receiving the original stream again, simultaneously filling its buffer while emptying it.

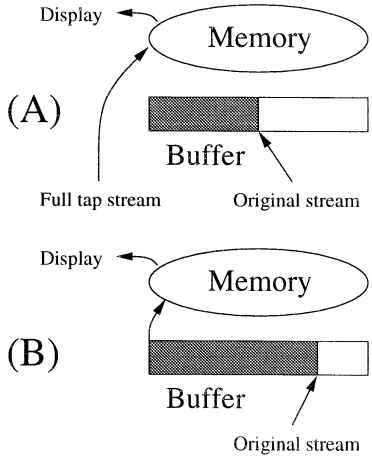

Fig. 1. The two stages an STB goes through when it is assigned a full tap stream.

Fig. 2 illustrates the three stages an STB assigned a partial tap stream must go through. Part (A) represents the first β minutes when it is receiving the partial tap and original streams, and Parts (B) and (C) represent, respectively, Steps 1 and 2 from above. In total, partial tap stream will exist for the first β minutes of the display and then for the first $\Delta_i - \beta$ minutes of every succeeding Δ_i-minute interval. That gives partial tap streams a service time of

$$S_p(i) = \beta + \left\lfloor \frac{L_i - \beta}{\Delta_i} \right\rfloor (\Delta_i - \beta) + \min(\Delta_i - \beta, (L_i - \beta) \bmod \Delta_i) \tag{3}$$

minutes.

Fig. 3 shows the three stream types from the VOD server's perspective. The two tap streams are tapping data from the original stream, and the shaded areas indicate when the streams are active.

One last definition relating to stream types is that for *video groups*. A video group is defined as an

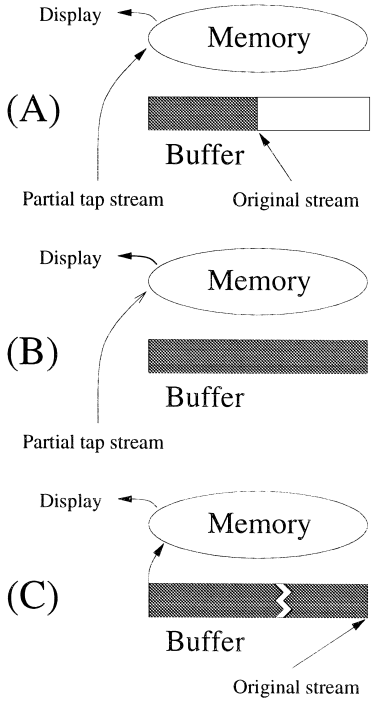

Fig. 2. The three stages an STB goes through when it is assigned a partial tap stream.

original stream for a video plus all of the streams tapping data from it.

3.3. The stream tapping algorithm

The actual algorithm for stream tapping is very straightforward. Every time the VOD server recognizes the potential to service a request, it must first assign each of the requests in its request queue one of the three stream types. Once the requests have been assigned, the server will know deterministically the service time and scheduling requirements of each. It can then use that information to check which requests can be serviced.

The stream assignment process works as follows. If no original stream is active for the requested video, then the request requires an original stream. Otherwise, if the request can use a tap stream, the server must decide between assigning the request a tap stream or assigning it an original stream. The choice for the current request will affect later requests, so the local (greedy) answer is not always best.

The server makes the decision between tap and original streams by examining the current video group of the request. With a minimal amount of extra storage (three fields for each video), the server can keep track of the minimum average service time (AST) for the streams in the group. It can also calculate (using Eq. (2) and Eq. (3)) the exact service time for the request should it be assigned a tap stream. The server can then compare the minimum AST of the group to the AST of the group with the tap stream included. If the new AST is worse than the minimum AST, then the server will select an original stream for the request; otherwise it will select the appropriate tap stream.

The key problem is deciding what ''worse'' means when comparing the service times. The AST of the group does not always decrease monotonically to its minimum point, and so using a strict comparison could cause the server to start a new group when the existing group could still be improved. Allowing the AST of the group to grow could cause the server to stay with a video group too long.

We solved this problem by adopting a ''3% rule'' where an original stream is only assigned if the new AST is at least 3% higher than the minimum AST. We chose the value 3% by examining simulated traces of client requests and by checking how often and by what amount different percentages would stop a group too early or stay with a group too long. Values of 0–6% performed well overall, with the smaller percentages in that range doing better with small buffers and small arrival rates and the larger percentages doing better on the opposite side of the spectrum. The value 3% was a nice middle ground, and it was never far from the optimal percentage for any trace.

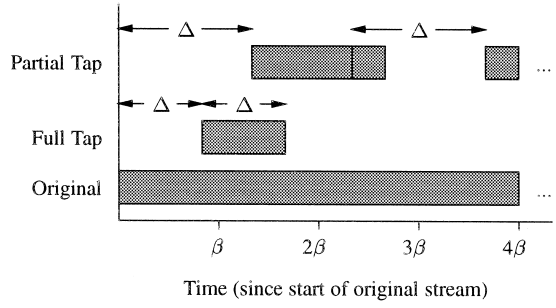

Fig. 3. The three stream types from the VOD server's perspective.

The stream decision process is summarized in Fig. 4.

3.4. Options

With the stream tapping algorithm as described in Section 3.3, the client STB only needs to receive at most two streams at any one time. If it can receive more, two options become available to the system.

The first of these options is called *extra tapping*. It allows an STB to tap data from any stream on the VOD server, not just from the original stream in its video group. By tapping extra data, the STB's request will require a shorter service time and the VOD server will save bandwidth.

There are three restrictions for extra tapping: the STB should not tap from more streams than its maximum bandwidth allows, the tapped data should not displace any data in the STB's buffer that the STB still needs, and the tapped data should still be in the STB's buffer when it is needed. These restrictions mean that extra tapping can only be used during the first β minutes of full and partial tap streams.

As an example, consider a situation where $\beta = 10$ and there are two full tap streams to an original stream, the first with $\Delta = 5$ and the second with $\Delta = 6$. Without extra tapping, the STB receiving the second full tap stream would require a stream of data on the VOD server for the first 6 minutes of the video. But since the STB can also use the data from the first full tap stream, extra tapping allows the STB to get minutes 2–5 of the video from there, reducing its service time from 6 to 2 minutes.

The second option is called *stream stacking*. If the VOD server currently has bandwidth available, an STB can acquire additional streams from the server to more quickly load in the data it needs. However, the STB must give up those streams if the server later needs them back, and stream stacking is faced with the same restrictions—and therefore can only be used in the same situations—as extra tapping.

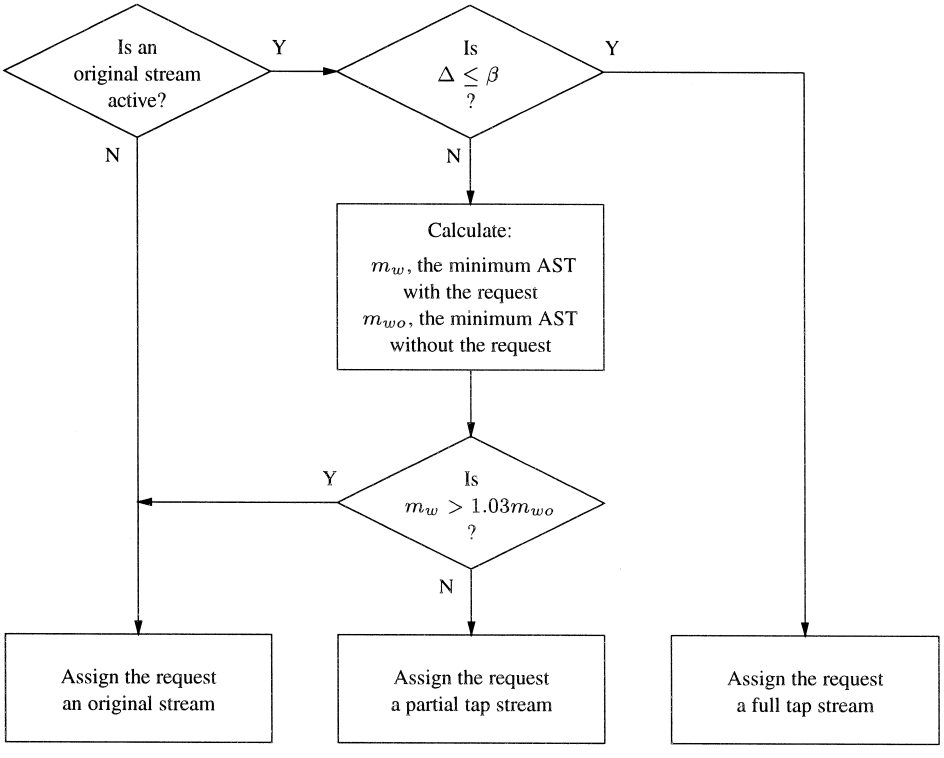

Fig. 4. The stream decision process.

Note that stream stacking does not change the service time of the request like extra tapping does. It simply rearranges when storage accesses take place on the server to try and avoid future bandwidth contention.

For example, suppose $\beta = 10$ and a full tap stream starts 4 minutes after an original stream. If the VOD server has an extra stream available, then the STB receiving the full tap stream can acquire the extra stream from the server and effectively double its bandwidth for receiving the first 4 minutes of the video. It can then release both streams after 2 minutes, thus freeing up bandwidth more quickly than it would have otherwise.

If stream stacking and extra tapping are used simultaneously, stream stacking can cause data to be read redundantly on the server. This happens when one request stacks data that later-arriving requests could have tapped, causing the VOD server to access the data again.

This redundancy does not hurt the VOD server, though, since the redundant data only uses bandwidth that otherwise would not have been used, and so at worst stream stacking can have no effect.

4. Simulation

We used simulation to analyze the stream tapping system. Several of the assumptions we made for this simulation are described below.

4.1. Video library

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 80 minutes and a maximum of 180 minutes to keep the values realistic.

The popularity of each video was modeled using a Zipf-like distribution with a parameter of 0.271. The Zipf distribution is the distribution recommended by Drapeau et al. (1994) and used by many others [6,14]. The parameter value was selected by Dan et al. (1996) to tune the distribution to match empirical video store rental patterns for the most popular 92 videos of the time.

4.2. 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. Interactivity is planned for the future.

We also made the simplifying assumption that all client STB's were exactly the same in terms of buffer size and bandwidth. In general, this does not need to be the case.

4.3. VOD server and network

We made three assumptions about the VOD server and the network: that the start-up latency for a stream was zero, that the network always had bandwidth available for the VOD server, and that the network latency was zero. These assumptions simplified the simulation while not significantly changing the results and are consistent with the assumptions made by other researchers in the field [1,2,6,8–11,14–16].

5. Results

We will now present several results for the stream tapping system. These results were obtained through the use of the simulation described in Section 4. Each run of the simulation consisted of a three-hour warm-up period followed by a 12-hour interval during which statistics were kept, and each data point is the mean of five such runs. This kept the variance of the values to less than 5%.

5.1. Configurations

We drew upon two standard configurations—one for each of our metrics—when running the simulation. Unless specified otherwise, these configurations define the parameters we used when simulating results.

The *latency configuration* is defined as follows:

- The VOD server has enough bandwidth for 300 streams of data and has a library of 92 videos.
- The client STB has a ten-minute buffer and has enough bandwidth to accept four streams of data.

The STB is also allowed to use extra tapping and stream stacking.

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

For the *bandwidth configuration*, we wanted to know how much bandwidth a VOD server would use if there was no contention for resources. Therefore this configuration has the following parameters:

- The VOD server has an unconstrained amount of bandwidth, but it only shows a single video. Information for multiple videos can then be extrapolated from the data for the single video.
- The client STB is the same as in the latency configuration, except it is not allowed to perform stream stacking. Given the unconstrained nature of the VOD server, stream stacking would only cause the server to read a great deal of redundant data, skewing the results.

5.2. Stream decision process

The stream decision process is the strategy by which the VOD server assigns a request one of the three stream types (see Section 3.3). Comparing our strategy to the optimal set of choices turned out to be slightly difficult.

Since almost every request can be assigned an original stream or a tap stream, exhaustively checking every possible set of choices would take $O(2^{\lambda})$ time. Thus, instead of determining the average service time (AST) of the optimal set of choices, we decided to find upper and lower bounds for it. We accomplished this through the use of an $O(\lambda^2)$ algorithm that made several passes through the requests, each pass i starting a video group at request i and determining the minimum AST of that group. The smallest of the minimum AST's over all of the requests is then a lower bound on what any set of choices could accomplish. By also keeping track of the index of the video where the minimum AST was achieved for each group, the same algorithm was also able to determine the AST of a policy that would always start a new video group when the current group had reached its minimum AST. The AST of this ''good'' policy is then an upper bound on the AST of the optimal set of choices.

Fig. 5 shows that our 3% rule fared very well in comparison to the bounds for the optimal set of choices. In fact, it is almost indistinguishable from the upper bound, which we expect is much closer to the optimal value than the lower bound. The greedy algorithm (always taking a tap stream when the opportunity is there) is shown as a point of interest.

5.3. Client STB

The more complex the client STB is, the more it will cost, and the less popular it will be with the consumer. Stream tapping was designed to work under a variety conditions, and in this section we will show how the complexity of the STB affects the efficiency of the VOD server.

Figs. 6 and 7 show how the size of the client STB's buffer affects the efficiency of the VOD server. While the average server bandwidth levels off fairly quickly, the drop in average client latency is dramatic. Even with a buffer as small as 10 minutes (or 115 megabytes for MPEG-1 encoding) clients wait on average well less than ten minutes, and when the buffer size is 30 minutes, clients almost do not have to wait at all.

An STB that can only receive two streams is essentially the same as one that does not use either of the options to the system, and increasing the number of streams the STB can receive only increases the degree to which the STB can use the options. Thus, Figs. 8 and 9 show not only how

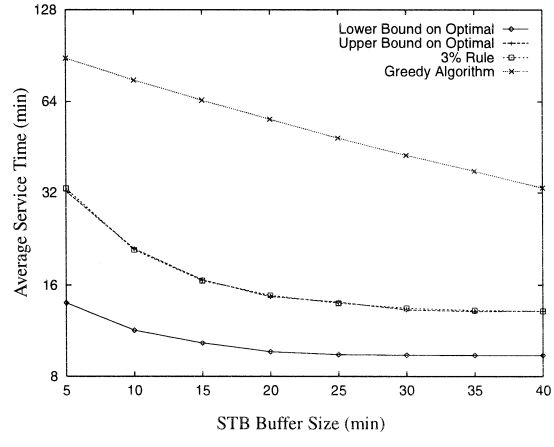

Fig. 5. Rating the stream decision process ($N = 1$, $\lambda = 30$, B_s unconstrained).

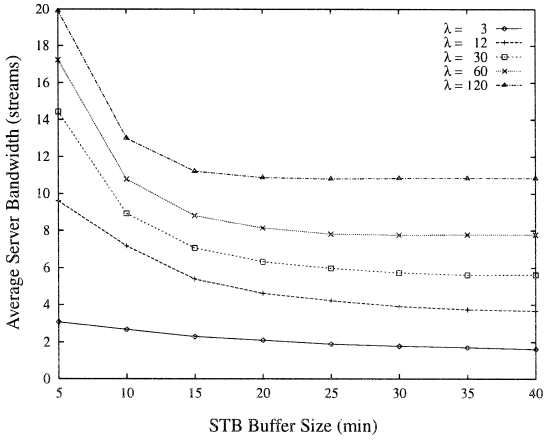

Fig. 6. β versus average server bandwidth ($N = 1$, B_s unconstrained).

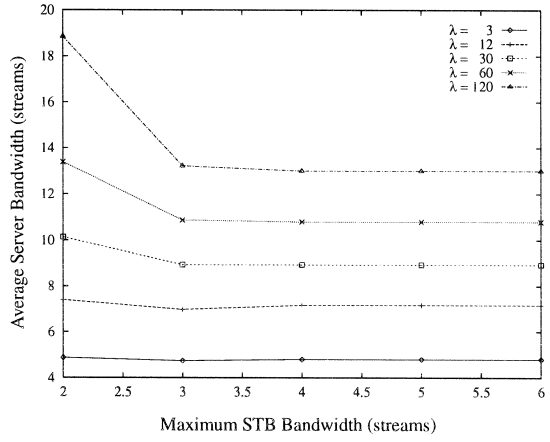

Fig. 8. B_c versus average server bandwidth ($N = 1$, B_s unconstrained).

much the STB bandwidth affects VOD server efficiency, they also indicate how useful the options are to the system.

Clearly, the savings gained from using the options is minimal. Although Fig. 8 shows a promising drop in server bandwidth when the STB increases its bandwidth from two to three streams, the client latencies are almost constant. In fact, the savings are due almost entirely to the extra tapping option, which only uses about one stream of bandwidth. Stream stacking has the problem that the times when it could be helpful are the same times when there are no available streams on the server for it to use. However, stream stacking might be much more useful in an interactive version of the system (see Section 6).

5.4. Comparisons

In this section we will compare stream tapping to several other VOD systems. In cases where there are multiple choices for the competing system, we have chosen a representative version for the comparison.

Fig. 10 illustrates how much server bandwidth is saved by using stream tapping instead of a conventional system. Note that stream tapping saves over 80% when the interarrival time is small and even saves 5–30% when the interarrival time is as much

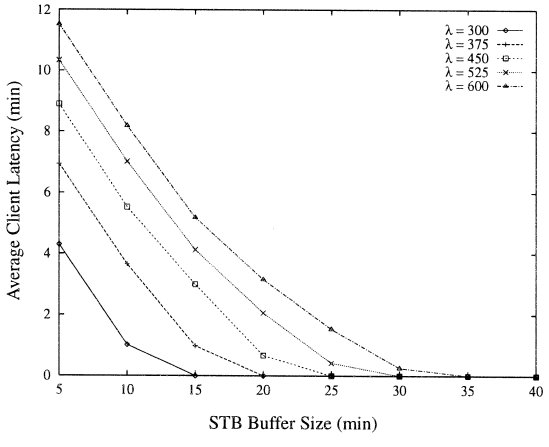

Fig. 7. β versus average client latency ($N = 92$, $B_s = 300$).

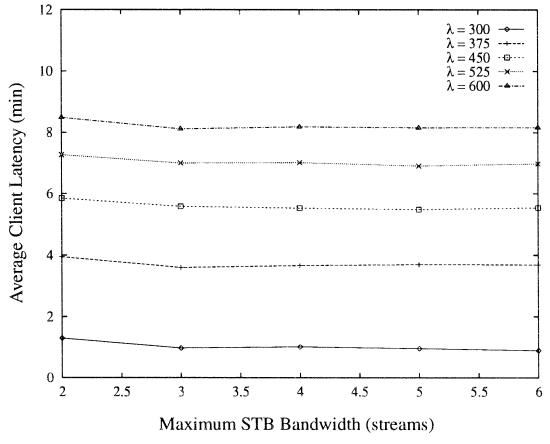

Fig. 9. B_c versus average client latency ($N = 92$, $B_s = 300$).

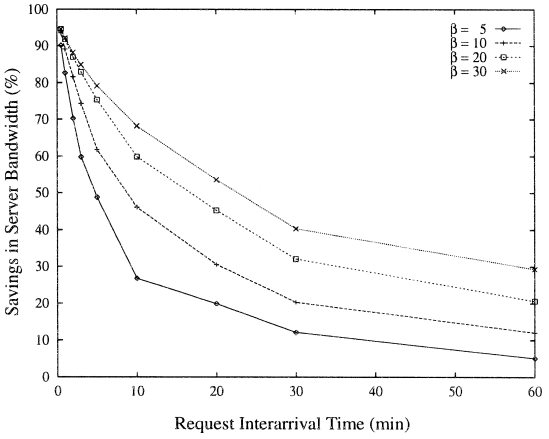

Fig. 10. Comparing stream tapping to conventional systems ($N = 1$, B_s unconstrained).

as 60 minutes. We could also compare latencies between stream tapping and conventional systems, but for any arrival rate over 163 per hour, a conventional system will begin to generate an infinite queue, and its latency will grow with the simulation length. For rates of 163 per hour and less, stream tapping will always generate a zero latency for clients unless the STB buffer is only one minute in size.

Next we compared stream tapping to the broadcasting systems. Since the quality of the broadcasting systems do not depend on the arrival rates of client requests, we were able to model them analytically, calculating the average client latency as a function of the bandwidth reserved by the systems. Approximately five videos are released each week, and so we chose that as the number of videos with which to evaluate the systems. Also, since the broadcasting systems tend to use at least a half hour of buffer space, we allowed stream tapping to use that much space as well.

The results are shown in Fig. 11, and even with the extremely high request rate—a situation that should hurt stream tapping in comparison to the broadcasting systems—stream tapping performs very well. In fact, from seven stream per video on, stream tapping beats all of the other systems except for Harmonic Broadcasting, and Harmonic Broadcasting may be of more theoretical than practical importance since it can require the client STB to receive hundreds of low bandwidth streams from the VOD server.

Finally, we compared stream tapping to the following reactive systems: batching, piggybacking using the ''optimal simple merging policy'' [1], interval caching with an 8-hour cache, and asynchronous multicasting modeled as stream tapping with only full tap streams and no options. Reducing asynchronous multicasting to a subset of stream tapping should help the system since it allows the client STB to use the entire buffer, and it allows clients to join existing multicast groups without waiting.

Fig. 12 shows the bandwidth comparison and Fig. 13 the latency comparison. In both cases, stream

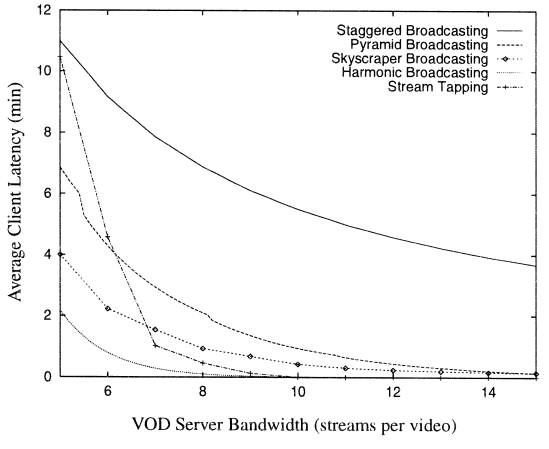

Fig. 11. Stream tapping versus the broadcasting systems ($N = 5$, $\beta = 30$, $\lambda = 360$).

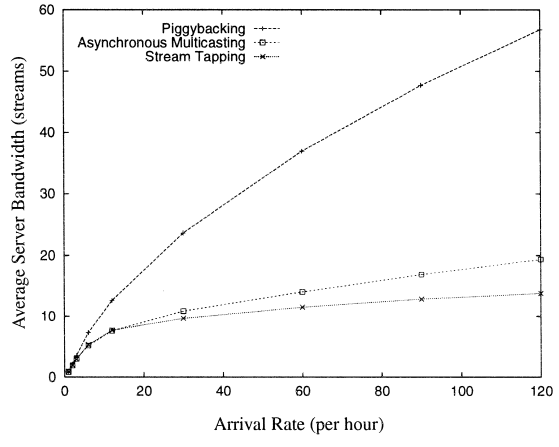

Fig. 12. Stream tapping versus other reactive systems ($N = 1$, B_s unconstrained).

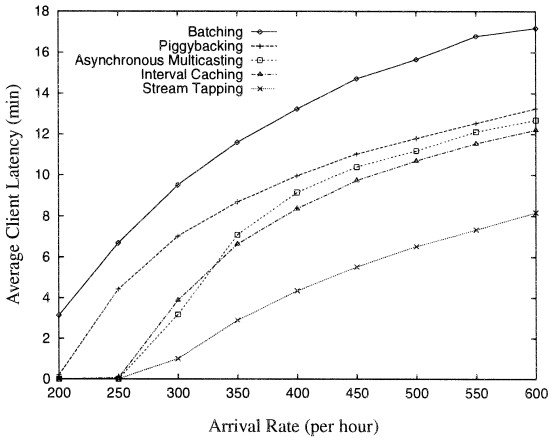

Fig. 13. Stream tapping versus other reactive systems ($N = 92$, $S = 300$).

tapping handily beats the other systems. Note that neither batching nor interval caching are considered in Fig. 12; batching would simply perform like a conventional system under the bandwidth configuration (since requests are never queued) and interval caching would be able to store the entire video in its cache, never having to use disk bandwidth.

6. Future work

The most important topic we plan to pursue in the future is interactivity. An interactive VOD server allows clients to use VCR controls such as pause, fast forward and rewind when viewing videos. It also makes the stream of data required by the client far more unique, reducing the ability of client STB's to share data.

A VOD server that shows videos at regular intervals (such as one employing staggered broadcasting) can support interactivity very efficiently in a *discontinuous* manner [2]. That is, the VCR controls can be mimicked by allowing the client STB to switch to an earlier-starting stream for fast forward, switch to a later-starting stream for rewind, and wait for a later-starting stream to catch up to the same position in the video for pause. This strategy does not require any extra bandwidth on the VOD server, but it does not give clients much precision with regard to the controls, and it does not allow for *cuing* (viewing the video while using fast forward or rewind).

Systems such as stream tapping, piggybacking, and batching can perform *continuous* VCR controls —but at a reduced rate of performance. Each VCR control used by the client can potentially move the client STB out of its current video group and into a new video group where it is the only member. This move can break down much of the work done by the system and cost the server extra bandwidth.

However, stream tapping should have an advantage:

- Stream tapping already uses a buffer on the client STB, and that buffer can keep the STB from requiring a new stream of data on the VOD server. For example, a client receiving an original stream can pause for up to β minutes before it has to switch to a new stream.
- If the client STB has to switch from the current stream (or streams) it is receiving, it is possible the STB will end up closer to an original stream than it was before, actually saving bandwidth.
- Interactive VOD servers often have to use *contingency channels* [4], which are streams reserved for clients who use VCR controls and consequently require new bandwidth on the server. These streams can go to waste in other systems, but stream tapping will be able to use them at all times with the stream stacking option.

7. Conclusions

We have presented a system called stream tapping that can improve the bandwidth efficiency of a video-on-demand (VOD) server. Through the use of a small buffer on the set-top box (STB), it allows clients to ''tap'' into existing streams of data on the VOD server, thus reducing the amount of new bandwidth clients require for their requests. This reduction allows more clients to use the server at once, and that in turn lowers the amount of time clients must wait before their requests can be serviced.

Stream tapping does not make any assumptions about its environment; it can be scaled to meet most requirements, including the size of the VOD server and the sophistication of the client set-top box. Stream tapping never delays requests when the VOD server has resources available, making it truly an on-demand system. It also does not require any a

priori knowledge by the VOD provider. Stream tapping will divide up the available bandwidth as the request patterns of the videos dictate.

We analyzed stream tapping through the use of simulation. Even when the STB buffer was only large enough to hold ten minutes of video data (115 megabytes for MPEG-1 encoding), stream tapping supported 600 requests per hour on a 300-stream VOD server and gave less than ten minute latencies for the clients. With a 30-minute buffer, it supported 600 requests per hour on the same system with almost no waiting time at all for the clients.

We also compared stream tapping to several other VOD systems. Against conventional systems, which simply dedicate a unique stream of data to each request, stream tapping saved over 80% on bandwidth for popular videos. Against more sophisticated systems, including broadcasting systems that can guarantee performance results regardless of the client request rate, stream tapping again performed better, providing significant savings in bandwidth and client latency.

Acknowledgements

We would like to thank the members of the Systems Group for their insights and helpful comments, J.-F. Pâris for his guidance and enthusiasm, and the Office of Naval Research for their support under Grant N00014–92–J–1807.

References

[1] C.C. Aggarwal, J.L. Wolf, P.S. Yu, On optimal piggyback merging policies for video-on-demand systems, in: Proc. Int. Conf. on Multimedia Systems, Hiroshima, Japan, June 1996, IEEE Computer Society Press, Silverspring, MD, pp. 253–258.

[2] K.C. Almeroth, M.H. Ammar, The use of multicast delivery to provide a scalable and interactive video-on-demand service, IEEE Journal on Selected Areas in Communications 14 (5) (August 1996) 1110–1122.

[3] S.W. Carter, D.D.E. Long, Improving video-on-demand server efficiency through stream tapping, in: Proc. ICCCN 97, Las Vegas, NV, September 1997, IEEE Computer Society Press, Silverspring, MD, pp. 200–207.

[4] A. Dan, P. Shahabuddin, D. Sitaram, D. Towsley, Channel allocation under batching and VCR control in video-on-demand systems, Journal of Parallel and Distributed Computing 30 (2) (November 1995) 168–179.

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

[6] A. Dan, D. Sitaram, P. Shahabuddin, Dynamic batching policies for an on-demand video server, Multimedia Systems 4 (3) (June 1996) 112–121.

[7] S. Goldstein, Time Warner proves it's RIP for VOD Billboard 109 (20) (May 1997) 58.

[8] L. Golubchik, J.C.S. Lui, R.R. Muntz, Adaptive piggybacking: a novel technique for data sharing in video-on-demand storage servers, Multimedia Systems 4 (30) (June 1996) 140–155.

[9] K.A. Hua, S. Sheu, Skyscraper broadcasting: a new broadcasting scheme for metropolitan video-on-demand systems, in: Proc. SIGCOMM 97, Cannes, France, September 1997, ACM, pp. 89–100.

[10] L. Juhn, L. Tseng, Harmonic broadcasting for video-on-demand service, IEEE Transactions on Broadcasting 43 (3) (September 1997) 268–271.

[11] H. Kalva, B. Furht, Techniques for improving the capacity of video-on-demand systems, in: Proc. 29th Annual Hawaii Int. Conf. on System Sciences, Wailea, HI, January 1996, IEEE Computer Society Press, Silverspring, MD, pp. 308–315.

[12] S. McCarthy, Bell Atlantic claims victory for true VOD, Telephony 230 (13) (March 1997) 16.

[13] T.S. Perry, The trials and trevails of interactive TV, IEEE Spectrum 33 (4) (April 1996) 22–28.

[14] H. Shachnai, P.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.

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

[16] H. Woo, C. Kim, Multicast scheduling for VOD services, Multimedia Tools and Applications 2 (2) (March 1996) 157–171.