Title

An efficient implementation of interactive video-on-demand

Authors

Carter, SW
Long, DDE
Paris, J-F

Publication Date

2000

DOI

10.1109/mascot.2000.876442

Peer reviewed

An Efficient Implementation of Interactive Video-on-Demand

Steven W. Carter Darrell D. E. Long*
Department of Computer Science
Jack Baskin School of Engineering
University of California, Santa Cruz
Santa Cruz, CA 95064
{*carter, darrell*}*@cse.ucsc.edu*

Jehan-François Pâris†
Department of Computer Science
University of Houston
Houston, TX 77204-3475
paris@acm.org

Abstract

The key performance bottleneck for a video-on-demand (VOD) server is bandwidth, which controls the number of clients the server can simultaneously support. Previous work has shown that a strategy called stream tapping can make efficient use of bandwidth when clients are not allowed to interact (through VCR-like controls) with the video they are viewing. Here we present an interactive version of stream tapping and analyze its performance through the use of discrete event simulation. In particular, we show that stream tapping can use as little as 10% of the bandwidth required by dedicating a unique stream of data to each client request.

1 Introduction

Video-on-demand (VOD) is a service that allows clients to communicate with a VOD server to select and then view the video of their choice at the time of their choice. A few companies, such as IBM [17] and the DIVA Systems Corporation [8], are successfully running VOD servers for relatively small numbers (less than 500) of clients. In order for VOD to become a wider commercial success and challenge the multi-billion dollar video rental industry, it must be able to handle many more clients, allow those clients to interact with the video through standard VCR controls, and do so with as much efficiency as possible to keep costs down.

The main bottleneck for a VOD service is bandwidth—either the disk bandwidth of the VOD server or the network bandwidth connecting the VOD server to the client *set-top box* (STB), the piece of hardware at the client premises responsible for receiving and decoding video data. These two bandwidths are related—using one efficiently almost always means using the other efficiently—and together they control how many concurrent streams of data the VOD server can send to clients.

A *conventional VOD system* does not use bandwidth efficiently. It simply dedicates a unique stream of data to each client request. For *interactive VOD*, which allows clients to interact with their selected video through VCR controls such as pause and rewind, conventional systems have a certain amount of appeal. They are straightforward to implement, and, since clients potentially will have unique streams of data due to their interactions, it is not clear to what degree the VOD server would be able to share data between clients anyway. Still, conventional systems do not scale well, and better solutions are necessary.

Systems such as *stream tapping* [3, 4] and *patching* [9] present one such solution. Stream tapping allows clients to "tap" into streams of data created for other clients who have requested the same video. By using existing streams as much as possible, clients minimize the amount of new bandwidth they require and allow more clients to simultaneously use the VOD server.

With more clients able to use the server, the VOD provider's cost per client is reduced and clients do not have to wait as long for their request to be serviced. Since the most important criteria for how a client will judge a VOD service are likely to be cost and waiting time, stream tapping has the potential to be an extremely effective solution for VOD.

Indeed, previous work [3, 4] has shown that stream tapping is effective for non-interactive VOD, performing better than all non-broadcasting schemes and performing competitively with broadcasting schemes, even at high workloads. Here we present an interactive version of stream tapping and explore its performance under a variety of conditions.

The subsequent sections are organized as follows. In Section 2 we describe stream tapping in some detail. Section 3 introduces interactive stream tapping and section 4 presents information about the simulation we used to analyze stream tapping, and then in Section 5 we show results from that simulation. Finally, Section 6 contains our con-

*This research was supported by the Office of Naval Research under Grant N00014–92–J–1807 and by the National Science Foundation under Grant PO–10152754.

†This research was supported by the Texas Advanced Research Program under grant 003652-0124-1999.

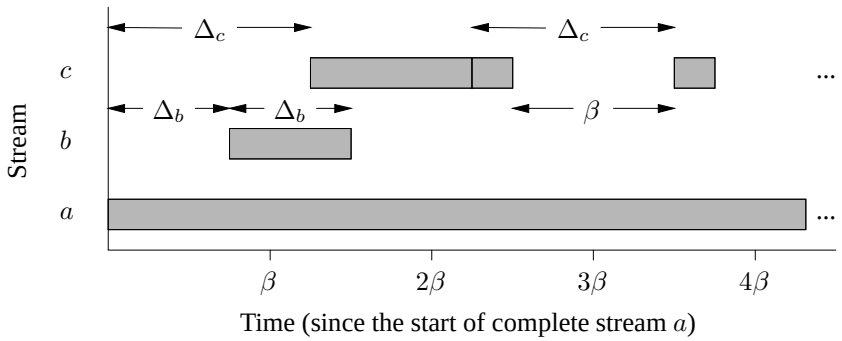

Figure 1: The three stream types from the VOD server's perspective. Stream a is a complete stream, b is a full tap stream, and c is a partial tap stream.

cluding remarks.

2 Stream Tapping

Stream tapping [3, 4] allows clients to "tap" into data streams created for other clients, which reduces the amount of new bandwidth required for their own requests. The strategy is very effective, using less than half the bandwidth of a conventional system for any video requested more than 10 times an hour [4]. Stream tapping also outperforms both *piggybacking* [7], which alters video display rates to merge video streams together, and *staggered broadcasting* [2,5], which starts video instances at regular intervals.

2.1 The Basic Algorithm

Stream tapping deals with overlapping videos. If one client begins viewing a 120-minute video, and 10 minutes later another client begins viewing the same video, then their videos will overlap for 110 minutes. If the second (later-arriving) client can use the data from the video stream feeding the first client, then the second client will only need its own stream for 10 minutes, and the VOD server will save over 90% on the bandwidth cost. Stream tapping defines a way in which clients can often "tap" into streams of data originally intended for other clients.

Because stream tapping allows clients to receive data far in advance of when it is needed, clients will need some sort of local buffer space. The capacity of this buffer, designated as β and measured in minutes of video data, is one of the factors determining when and in what way a client can tap data.

Clients will tap most of their data from *complete streams*. These are "normal" streams. They start at a particular position in a video and transmit the remainder of the video. For non-interactive stream tapping, this starting position is always the beginning of the video.

There are two mutually exclusive ways a client can tap data. These two ways are determined by Δ, the difference in video position (in minutes) between the client and a complete stream. If the client's request has not been serviced yet, we assume it is at position zero.

The first way a client can tap data is through the use of a *full tap stream* (Figure 1, stream b). This method can be used if there exists a complete stream for the same video as the request, and if $\Delta \leq \beta$. Then the client can receive the complete stream and put it in its buffer while it simultaneously receives the first Δ minutes of the video from the full tap stream and displays it live. Once the client reaches the Δ-minute point of the video, it can receive the rest of the video from its buffer, which will continue to be fed by the complete stream, and which will therefore always contain a moving Δ-minute window of data.

Clients can also use *partial tap streams* (Figure 1, stream c). These streams can be used if there is a complete stream for the same video as the request, but $\Delta > \beta$. In this case, the client must go through phases of filling up and emptying its buffer since the buffer will not be large enough to account for the difference in video positions. In particular, the client will receive the complete stream for β minutes to fill up its buffer and simultaneously receive the first β minutes of the video from the partial tap stream. The client will then receive the next $\Delta - \beta$ minutes of the video via the partial tap stream to catch itself up to the video data in its buffer. At that point it can simultaneously empty its buffer and re-fill it using the complete stream, and then receive the next $\Delta - \beta$ minutes of the video from the partial tap stream again to catch itself up to the buffer, and then repeat until the video is complete.

Note that these definitions describe when a client *can* use a tap stream but not when it *should*. Stream tapping makes this decision based on the *service times* of the three types of streams (see Table 1) and the current *video group* of the request. The service time is defined as the amount of time the client requires its own stream of data, and the video group is defined as a complete stream and all of the streams tapping data from it. With a minor amount of bookkeeping, the VOD server can keep track of the current and minimum average service times for each video

Table 1: Service times for the three stream types when the length of the video is L and the difference in video position (for tap streams) is Δ.

Stream type	Service time
Complete	L
Full tap	Δ
Partial tap	$\beta + \lfloor \frac{L-\beta}{\Delta} \rfloor (\Delta - \beta) + \min(\Delta - \beta, (L - \beta) \bmod \Delta)$

group.

Stream tapping makes the stream decision in the following manner. If no complete stream for the requested video is active, the request must use a complete stream. If a tap stream can be used, stream tapping compares the service time of the potential tap stream to the minimum average service time of the video's current group. If the former is less than a constant factor α of the latter, then the request will be assigned a tap stream. Otherwise, it must use a complete stream.

The idea behind the decision process outlined above is that a video group will start with a large average service time because of the initial complete stream, but then tap streams will advance the service time almost monotonically to its minimum point. Stream tapping attempts to determine this minimum point and then start a new video group when it is reached. Since the progression to the minimum service time is almost monotonic, an α value near 1.0 makes the most sense, and we have found $\alpha = 1.03$ to work well in practice.

2.2 Further Improvements

For a client to use a tap stream, it must have the capability to receive two distinct streams at once—the tap stream and a complete stream. If the client can handle a bandwidth higher than two streams, then two more options become available to it.

The first of these options is called *extra tapping*. Extra tapping allows a client to tap data from any stream on the VOD server providing data that it can use, and not just from the complete stream in its video group.

For example, suppose a complete stream starts at time t_0, and a full tap stream starts at time $t_0 + 5$ minutes. Then, if a second full tap stream starts at time $t_0 + 6$ minutes, extra tapping will allow it to tap normally from the complete stream but also tap four minutes of data from the first full tap stream. In all, the service time of the second full tap stream would drop from six minutes to two minutes, saving the VOD server some bandwidth for other requests.

There are two limitations to extra tapping. The first, as alluded to earlier, is bandwidth-related. The client cannot receive more streams than its maximum bandwidth allows. The second limitation is buffer-related. The client is not allowed to tap data that it does not need or displace data that it does need. Since the buffer of a partial tap stream is completely spoken for after β minutes, and since a full tap stream exists for at most β minutes, extra tapping can only be used for the first β minutes of tap streams.

Stream tapping's second option is called *stream stacking*. If a client, under the same limitations as for extra tapping, can receive its tap stream at a higher rate than normal, and if the VOD server has streams available for use, then the client can stack some of these streams together and use them to more quickly receive the tap stream. By rearranging bandwidth in this way, stream stacking allows the VOD server to finish servicing existing streams more quickly, allowing new streams to be scheduled.

3 Interactive Stream Tapping

Interactive VOD is much more difficult to implement that non-interactive VOD. Not only do clients have to be "merged" together initially so they can share data, they also potentially have to be "re-merged" every time they make an interaction. The initial merging problem has been solved in numerous ways, but the interactive merging problem has caused new problems, and that is why many broadcasting protocols either do not support interactive VOD [10, 11, 15, 16, 18] or only support limited or discontinuous VCR functions [2]. It is also why straight batching is not a good idea (since it cannot re-merge clients) and why piggybacking [7] is not effective (since it re-merges too slowly). Interactive VOD adds two new areas for stream tapping to manage. One is the interaction itself, which can include removing a client from its video group and deallocating bandwidth, and the other is the completion of the interaction, which can include adding a client to an existing video group or creating a new video group altogether. We will discuss these areas in order.

When a client initiates an interaction, stream tapping first attempts to release bandwidth associated with that client. It examines each stream the client is scheduled to receive, and if the client is the only one to receive them, then the streams will be deallocated.

Stream tapping then determines the amount of bandwidth required by the interaction, if any. If the bandwidth is available, the client will begin the interaction immedi-

ately using an *interaction stream*. Otherwise the client will be forced to wait. Note that we could allow stream tapping to use the client's buffer at this point, but we wanted to ensure that once a client starts an interaction, it will be able to continue the interaction for as long as desired, and not be forced either to terminate the interaction early or wait for bandwidth in the middle.

Once the client finishes its interaction, it will find itself in a similar position to when it first made its request: there will either be a leading complete stream some Δ away that it can tap from, or there will be no such stream. Once again the server will have to decide on a stream type for the request, but this time the process will be different.

Whereas stream tapping used service times to make the decision, interactive stream tapping uses the positional difference, or *distance*, between streams. There are two reasons for this. First, distance correlates well with service time, but distance is easier to calculate. Second, it is important to space complete streams sufficiently apart to allow room for stream tapping to assign tap streams, and while it is possible to calculate the distance between a client and leading and trailing complete streams, it is not possible to do so with service times.

Therefore, given some distance δ for each video, any client finishing an interaction with a video position within δ of a complete stream is assigned the appropriate tap stream. Otherwise it is assigned a complete stream.

Note that δ is the maximum distance away from a complete stream that a tap stream should be assigned. Stream tapping can approximate this distance easily. Every time a new group for a video is created, the amount of time the previous group was active is an approximation of δ. Stream tapping simply averages the three most recent of these times together and uses that value for δ.

Once the client has been assigned a stream type, the VOD server will know the initial stream requirements for the client. At this point the client can use its buffer to perhaps reduce those requirements, and then the server will be able to attempt to allocate bandwidth for the client. If the bandwidth is available, the client will begin receiving data immediately. Otherwise it will have to wait.

3.1 Contingency Streams

When clients make an interaction, they often require a new stream of data, either for the interaction itself (*e.g.* when rewinding with picture) or for the period of normal playback following the interaction. In order to prevent the client from being blocked in such a request for new bandwidth, stream tapping attempts to keep a number of streams available on the VOD server. These streams are called *contingency streams*.

Normally, the more contingency streams a VOD server has, the less time clients will have to wait to resume playback after an interaction, but then more bandwidth will be wasted. With stream tapping, however, clients employing the stream stacking option can use all available streams, including contingency streams, and so very little bandwidth will be wasted.

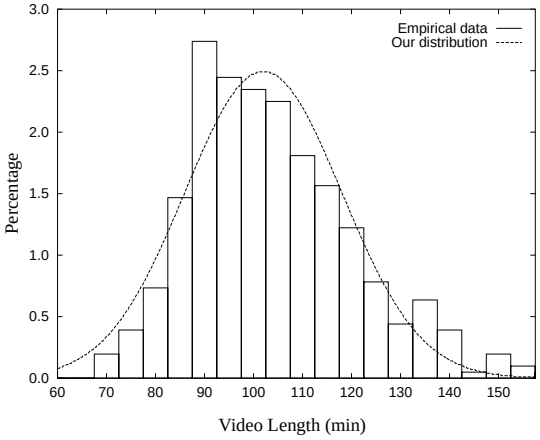

Figure 2: A comparison of our video length distribution to empirical data.

4 Our Simulation Model

Stream tapping is complex enough that an analytical model was not an option, and so we used discrete event simulation to study the system. In this section we will detail some of the assumptions and workload characteristics we used in the simulation.

4.1 Video Library

We had to decide on two factors for each video: its length and its popularity. For the lengths, we used empirical data from 409 videos released during 1997 and 1998. The Internet Movie Database[1] was our primary source for video lengths and Video Hits Spotlight[2] for video release dates. From this data we found that a normal distribution with a mean of 102 minutes and a standard deviation of 16 minutes provided the smallest sum-squared error while retaining integer values for the parameters. A comparison of our distribution to the empirical data is shown in Figure 2. We also truncated the video lengths to a minimum of 70 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 parameter 0.271. This is the distribution used by most VOD studies [1, 2, 5, 6, 12–14]. The 0.271 parameter was determined using empirical rental patterns from 92 videos, and so we modeled the VOD server to have 92 videos as well ($n = 92$).

[1] http://www.imdb.com
[2] http://www.myvideostore.com

4.2 VCR Controls

We modeled four VCR controls in our simulation:

- Pause,
- Rewind with *cueing*, where cueing means the server provides picture but not sound,
- Fast forward with cueing, and
- Jump, which allows the client to jump to any point in the video.

We believe these are the four controls that any interactive VOD service should provide.

The usage distributions that we used for the VCR controls are summarized in Table 2. These parameters represent our best estimates in the absence of empirical evidence. We assume that clients will simply watch the videos they select, and that they will mostly use interactions when they are interrupted or if they need to review something they missed the first time. That is why, for example, pause and rewind have the highest two frequencies, and why we modeled the initiation of interactions using a Poisson process with an interarrival time of 30 minutes (for each client).

We chose a cueing rate of 20 times the display rate. This is a rate that should be similar to what one might find on a standard VCR. We also assumed that each cued operation could be handled using a single stream, rather than requiring 20 times the bandwidth. If this cued stream cannot be formed by sampling frames from a normal stream (which is likely the case for MPEG encodings), then the VOD provider can always keep a second (cue-friendly) version of each video and only incur a small storage penalty. Since cued data cannot be used for normal playback (or tapping), it is immediately flushed from the client buffer after it is displayed.

4.3 Clients

Clients were generated using a Poisson arrival process with an interarrival time of $1/\lambda$, for varying values of λ. between 250 and 350 arrivals per hour. Once generated, clients simply selected a video, waited for their request to be serviced, and then interacted and watched the video until it was completed.

4.4 Server, Network, and STB

These are the components for which we made the most simplifying assumptions—partly because detailing them would not greatly enhance the simulation and also because we wanted the results and parameters to reflect only what is caused and needed by stream tapping alone. So when we restrict a client to a specific buffer size, this is only the buffer that stream tapping uses and may not be the only buffer on the STB. When we report client latency, this is only the latency caused by stream tapping and does not include transportation delay, stream start-up delay, or other possible delays. We also assume that the network never refuses bandwidth to the VOD server, and that all of the client STB's have identically sized buffers and can receive four streams of data at once.

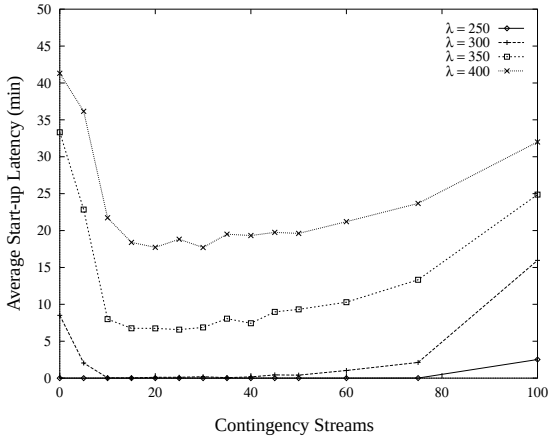

Figure 3: How the number of contingency streams affects start-up latency ($n = 92$, $s_t = 300$, $\beta = 30$).

5 Results

For the results, we chose to model a relatively small VOD server, one that only has enough bandwidth for 300 streams of data ($s_t = 300$). Smaller VOD servers are more difficult to manage than larger servers because there is less bandwidth available per video and because with fewer clients able to use the server, the proximity of requests is farther apart and merging streams becomes more expensive. Thus, the results presented in this section can be seen as a lower bound on stream tapping's capabilities.

One of the most important aspects of the performance of a VOD server is its *start-up latency*, that is the amount of time clients must wait to watch their video. Figure 3 displays the system average start-up latency for request arrival rates between 250 and 400 requests per hour, and shows how these latencies are affected by the number of contingency streams s_c. As one can see, start-up latencies remain very small as long as the request arrival rate λ remains less than or equal to 300 requests per hour. On the other hand, higher request arrival rates, say 350 to 400 requests per hour, result in unacceptable latencies. As we will see later, the poor performance of the server in this range of request arrival rates is due to the relatively small size of the client buffer.

We also found out that allocating too few contingency streams is much worse for start-up latency than allocat-

Table 2: The distribution parameters for each of the four interactions. The actual duration is selected in a uniform manner between the minimum and maximum values. C_R is the cueing rate.

Interaction	Frequency of Use (%)	Minimum Duration (min)	Maximum Duration (min)
Pause	60.0	1.0	15.0
Fast Forward	10.0	$1.0/C_R$	$5.0/C_R$
Rewind	25.0	$1.0/C_R$	$5.0/C_R$
Jump	5.0	–	–

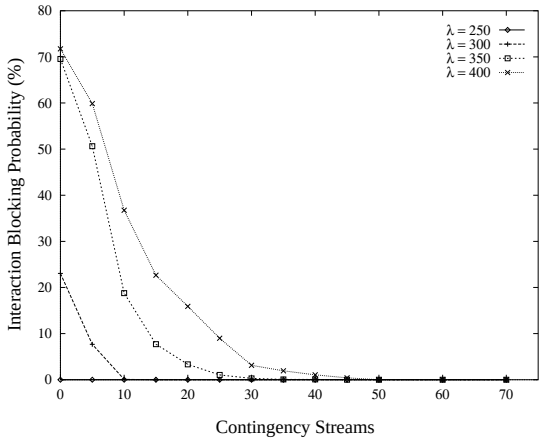

Figure 4: How the number of contingency streams affects resume blocking ($n = 92$, $s_t = 300$, $\beta = 30$).

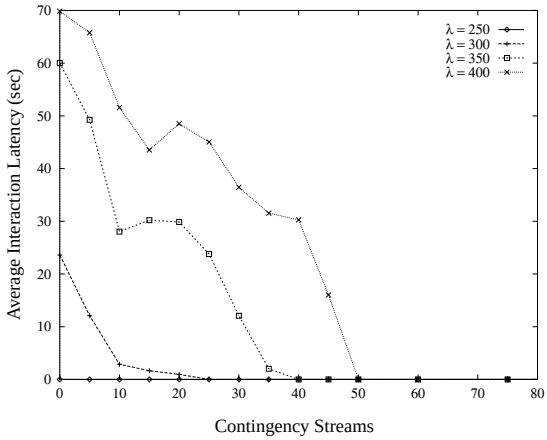

Figure 5: How the number of contingency streams affects resume latencies ($n = 92$, $s_t = 300$, $\beta = 30$).

ing too many streams. This fact is largely dependent on the stream stacking option, which can use available bandwidth to improve performance. When the number of contingency streams is low, bandwidth will almost always be in use and stream stacking will be able to do little work.

Figures 4 and 5 show how the number of contingency streams affects interactions. The *interaction blocking probability* measures the likelihood that a client must wait for bandwidth either during or after an i nteraction, and the *interaction latency* is the total amount of time the client must wait for bandwidth (only) when it is blocked. Assuming the VOD provider is anticipating fewer than 350 requests per hour (which is likely considering Figure 3), 35 contingency streams are enough to guarantee a blocking probability of less than 0.1% and an average latency of 2 seconds. Those bounds should be more than enough to ensure client satisfaction.

Conventional stream tapping can provide good performance even when the client buffer is as small as 5–10 minutes in size [4]. We found this was not the case with interactive stream tapping, As Figure 6 demonstrates, clients need at least a 25-minute buffer for adequate performance when the arrival rate λ is 300 requests per hour and a 35-minute buffer when the rate is 350 requests per hour.

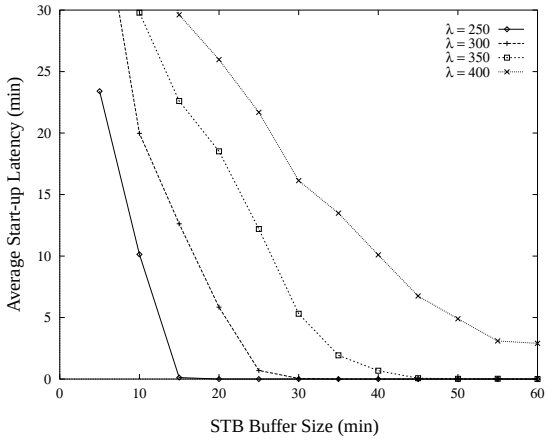

Figure 6: How the client buffer size affects performance ($n = 92$, $s_t = 300$, $s_c = 30$).

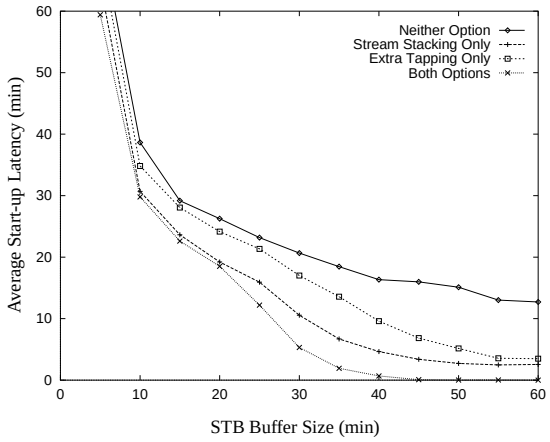

Figure 7: How the stream stacking and extra tapping options affect performance ($n = 92$, $s_t = 300$, $s_c = 30$, $\lambda = 350$).

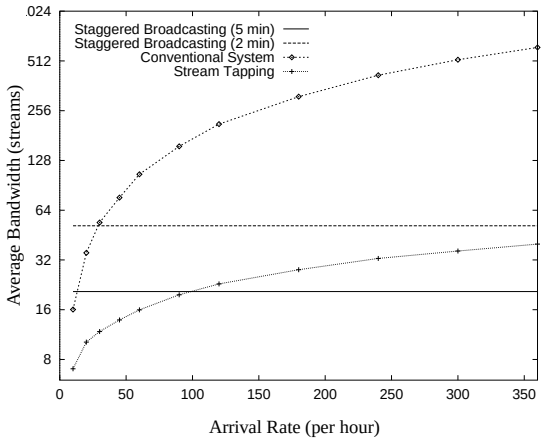

Figure 8: A comparison between conventional systems, staggered broadcasting (with 2- and 5-minute intervals between retransmissions), and stream tapping ($n = 1$, s_t unconstrained, $\beta = 30$).

There is indeed a stronger case for even larger buffers. First, a 60-minute buffer would allow the server to handle 400 requests an hour while keeping the average start-up latency under three minutes. Second, a two-hour buffer would allow the client to keep almost all previously viewed video data in local storage. As a result, rewind and pause interactions could be handled by the client STB without any server intervention. If MPEG-2 encoding (at 4 Mb/s) is used, then a two-hour buffer would require 3.6 GB of space, and a disk drive of that capacity would not add greatly to the price of the STB (indeed, it is difficult to buy a disk drive with capacity less than 4GB).

The two stream tapping options, extra tapping and stream stacking, can only be used during the first β minutes of full and partial tap streams, and we would expect that their impact would increase with β . This is indeed the case and is shown in Figure 7. With a 45-minute buffer, using both options reduces client start-up latency from almost 16 minutes to less than 5 seconds. Hence using both stream tapping and extra tapping is essential to the good performance of the server.

Figure 8 compares the bandwidth requirements of stream tapping and conventional systems. Little's Law tells us that with at most 300 active requests and an average service time of 104 minutes,[3] conventional systems can only obtain a throughput of 173 requests an hour. We have already shown that stream tapping can handle twice that amount effectively (in Figure 3, for example). Therefore it should be no surprise that stream tapping also outperforms conventional systems in terms of bandwidth, using as little as 6% as much bandwidth at the highest shown arrival rate and 44% at the lowest.

Of the remaining interactive strategies, only staggered broadcasting is likely to be competitive with stream tapping. We have previously shown, for example, that stream tapping performs better than piggybacking [7] for non-interactive VOD [4], and since stream tapping can "merge" streams ten times faster than piggybacking, and since interactive VOD only means streams will have to be merged more often, the difference between the two should only grow larger.

Figure 8 shows the bandwidth requirements for stream tapping as compared to staggered broadcasting when the intervals between successive retransmissions are two and five minutes. Those two intervals are likely to be the upper and lower bounds for staggered broadcasting since anything more than five minutes will not give enough interactive precision to the clients, and anything less than two minutes will take up too much bandwidth. Stream tapping uses less bandwidth than even the lowest staggered broadcasting bound as long as the video is not requested more than 100 times per hour, and it sits comfortably between the two bounds even if the video is requested 360 times per hour. Since stream tapping gives perfect precision for interactions and does not impose a start-up latency, it is clearly competitive with staggered broadcasting.

6 Conclusions

Interactive video-on-demand (VOD) allows a client to watch the video of its choice at the time of its choice, and then interact with the video through VCR-like controls such as pause and rewind. Most strategies designed to make efficient use of a VOD server's bandwidth either do not support interactive VOD at all, or they only provide VCR controls with limited duration or coarse precision.

We have presented a strategy for interactive VOD based

[3] We assume that clients will release streams when they pause, and so interactions only change the average service time of a video very slightly.

on stream tapping, a scheme previously shown to work well in a non-interactive environment.

Through the use of discrete event simulation, we were able to examine the performance of interactive stream tapping. We found that two factors were essential to the performance of our new strategy. First, interactive stream tapping requires a much larger client buffer than non-interactive stream tapping. While non-interactive stream tapping provided good performance with a client buffer capable of containing between 5 and 10 minutes of video data, interactive stream tapping requires a 25-minute buffer in each STB and performs much better with a 60-minute buffer. Assuming a MPEG-2 encoding at 4Mb/s, this translates into between 750 MB and 3.6 GB of disk space. Second, extra tapping and stream stacking are essential to the good performance of the server.

Under these circumstances, interactive stream tapping outperforms all other existing strategies for interactive VOD. In particular, interactive stream tapping can use less than 10% of the bandwidth required by a strategy dedicating a unique stream of data to each client request. Also, although interactive stream tapping can use a similar amount of bandwidth as staggered broadcasting when client request rates are high, it provides a better VOD service by not forcing clients to wait for their requests and by not limiting interactions.

References

[1] E. L. Abram-Profeta and K. G. Shin. Providing unrestricted VCR functions in multicast video-on-demand servers. In *IEEE International Conference on Multimedia Computing and Systems*, pages 66–75, Austin, TX, USA, June 1998. IEEE Computer Society Press.

[2] K. C. Almeroth and 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):1110–22, Aug. 1996.

[3] S. W. Carter and D. D. E. Long. Improving video-on-demand server efficiency through stream tapping. In *Proceedings of the Sixth International Conference on Computer Communications and Networks (ICCCN '97)*, pages 200–7, Las Vegas, NV, USA, Sept. 1997. IEEE Computer Society Press.

[4] S. W. Carter and D. D. E. Long. Improving bandwidth efficiency on video-on-demand servers. *Computer Networks*, 30(1–2):99–111, Jan. 1999.

[5] A. Dan, P. Shahabuddin, D. Sitaram, and D. Towsley. Channel allocation under batching and VCR control in video-on-demand systems. *Journal of Parallel and Distributed Computing*, 30(2):168–79, Nov. 1995.

[6] A. Dan and D. Sitaram. Multimedia caching strategies for heterogeneous application and server environments. Technical Report RC 20670, IBM Research Division, T.J. Watson Research Center, Dec. 1996.

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

[8] M. Gunther. Interactive TV: it's baaack! *Fortune*, 138(2):136–7, July 1998.

[9] K. A. Hua, Y. Cai, and S. Sheu. Patching: A multicast technique for true video-on-demand services. In *Proceedings of the Sixth ACM Multimedia Conference*, pages 191–200, Bristol, UK, Sept. 1998. ACM.

[10] K. A. Hua and S. Sheu. Skyscraper Broadcasting: a new broadcasting scheme for metropolitan video-on-demand systems. In *Proceedings of SIGCOMM '97*, pages 89–100, Cannes, France, Sept. 1997. ACM.

[11] L. Juhn and L. Tseng. Harmonic broadcasting for video-on-demand service. *IEEE Transactions on Broadcasting*, 43(3):268–71, Sept. 1997.

[12] H. J. Kim and Y. Zhu. Channel allocation problem in VOD system using both batching and adaptive piggybacking. *IEEE Transactions on Consumer Electronics*, 44(3):969–76, Aug. 1998.

[13] S.-E. Kim, A. Sivasubramaniam, and C. R. Das. Analyzing cache performance for video servers. In *Proceedings of the 1998 ICPP Workshop on Architectural and OS Support for Multimedia Applications Flexible Communication Systems*, pages 38–47, Minneapolis, MN, USA, Aug. 1998. IEEE Computer Society Press.

[14] W. Liao and V. O. K. Li. The split and merge protocol for interactive video-on-demand. *IEEE Multimedia*, 4(4):51–62, Dec. 1997.

[15] J.-F. Pâris, S. W. Carter, and D. D. E. Long. A hybrid broadcasting protocol for video on demand. In *Proceedings of the 1999 Multimedia Computing and Networking Conference (MMCN '99)*, pages 317–26, San Jose, CA, USA, Jan. 1999.

[16] J.-F. Pâris, S. W. Carter, and D. D. E. Long. A reactive broadcasting protocol for video on demand. In *Proceedings of the 2000 Multimedia Computing and Networking Conference (MMCN '00)*, pages 216–23, San Jose, CA, USA, Jan. 2000.

[17] T. Sanuki and Y. Asakawa. Design of a video-server complex for interactive television. *IBM Journal of Research and Development*, 42(2):199–218, Mar. 1998.

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