A Stream Tapping Protocol with Partial Preloading

Jehan-François Pâris

Department of Computer Science
University of Houston
Houston, TX 77204-3475

paris@cs.uh.edu

Abstract

Stream tapping–also known as patching–can reduce the bandwidth requirements of video-on-demand services by allowing new customer requests to "tap" the data streams of other requests for the same video. Previous studies have shown that stream tapping works best when the request arrival rate does not exceed ten to twenty requests per hour for a two-hour video. At higher arrival rates, it performs much worse than broadcasting protocols.

To overcome this limitation, we propose a stream tapping protocol that preloads in the customer set-top box the first few minutes of all popular videos. To offset the cost of the additional buffer space, our protocol never requires the set-top box to receive data from the video server at more than twice video consumption rate. Our simulations indicate that preloading the first eight minutes of a two-hour video was enough to achieve lower bandwidth requirements than the best broadcasting protocols at any request arrival rate.

1. Introduction

One of the reasons behind the slow deployment of video-on-demand (VOD) services is the high cost of providing these services. These high costs result mostly from the high demands that VOD makes upon video servers. Assuming that the videos are in MPEG-2 format, each user request will require the delivery of around 5 Megabits of data per second. Hence a video server allocating a separate stream of data to each request would need an aggregate bandwidth of 5 Gigabit/s to accommodate 1,000 concurrent users.

This situation has led to numerous proposals aiming at reducing the bandwidth requirements of VOD services. These proposals can be broadly classified into two groups. Proposals in the first group are said to be *proactive* because they distribute each video according to a fixed schedule that is not affected by the presence–or the absence–of requests for that video. They are also known as *broadcasting* protocols. Some of the best-known broadcasting protocols are *staggered broadcasting* [Alm96], *pyramid broadcasting* [20], *skyscraper broadcasting* [10], *harmonic broadcasting* [12] and its variants [14].

Other solutions are purely *reactive*: they only transmit data in response to a specific customer request. Unlike proactive protocols, reactive protocols do not consume bandwidth in the absence of customer requests. The best known reactive protocols include *piggybacking* [9], *stream tapping* [2]–also known as *patching* [11]–and *dynamic skyscrape*r [5].

The main advantage of broadcasting protocols is that they scale up extremely well. Since broadcasting protocols distribute each video according to a fixed schedule, the number of incoming user requests does not affect their bandwidth requirements. Hence they are especially suited to the distribution of videos that are in very high demand. The most efficient broadcasting protocols only require a bandwidth equal to six times the video consumption rate to guarantee that no customer will have to wait more than one minute before starting to watch a two-hour video.

Despite their low bandwidth requirements, broadcasting protocols have their own limitations. First, the most efficient broadcasting protocols require a set-top box (STB) capable of receiving data at up to six or seven times the video consumption rate. Those that have lower client bandwidth requirements also put more demands on the video server. While Hua and Sheu's *skyscraper broadcasting* [10] never requires the STB to receive more than two streams at the same time, it requires ten times the video consumption rate to guarantee a maximum customer waiting time of less than a minute for a two-hour video. Second, most broadcasting protocols require a STB capable of storing locally up to 60 percent of each video being watched. Finally, they waste a considerable amount of bandwidth whenever the request arrival rate falls below ten to twenty requests per hour for the same two-hour video.

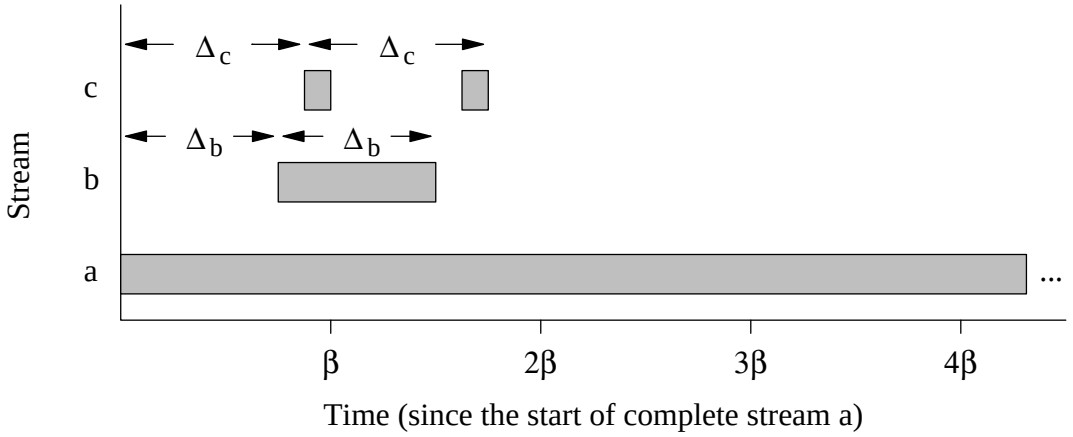

Figure 1: How stream tapping works

We propose here a different approach. First, we will use a reactive distribution protocol because these protocols perform very well at low to medium request arrival rates.

Second, we will never require the STB to receive data at more than twice the video consumption rate, which will greatly simplify the design of the STB disk controller. Finally, we will keep the server bandwidth low by preloading in the customer STB the first 8 to 15 minutes of all popular videos, that is, between 300 and 563 Megabytes of data per video in MPEG-2 format. As a result, our *stream tapping* protocol with *partial preloading* will trade excess bandwidth for an increase of the size of the STB buffer. All VOD distribution protocols that minimize the server bandwidth require a STB buffer of the order of a few Gigabytes. In the present state of memory technology, this implies the presence of a disk drive in each STB. The sole additional requirement of our protocol is to have a disk drive capable of storing twenty to thirty Gigabytes of data in each STB. Such disk drives sell now at a retail prices below one hundred dollars.

2. Previous Work

Two of the earliest reactive distribution protocols are batching and piggybacking. *Batching* [4] reduces the bandwidth requirements of individual user requests by multicasting one single data stream to all customers who request the same video at the same time. Some strategies even involve delaying customer requests for a short period of time in order to increase the number of customers sharing the same data stream. *Piggybacking* [9] can be used alone or in combination with batching. It adjusts the display rates of overlapping requests for the same video until their corresponding data streams can be merged into a single stream. Consider for instance, two requests for the same video separated by a time interval of three minutes. Increasing the display rate of the second stream by 10 percent will allow it to catch up with the first stream after 30 minutes.

Stream tapping [2, 3] or *patching* [11], assumes that each customer STB has a buffer capable of storing at least 10 minutes of video data. This buffer will allow the STB to "tap" into streams of data on the VOD server originally created for other clients, and then store these data until they are needed. In the best case, clients can get most of their data from an existing stream.

In particular, stream tapping defines three types of streams. *Complete streams* read out of a video in its entirety. These are the streams clients typically tap from. *Full tap streams* can be used if a complete stream for the same video started $\Delta \leq \beta$ minutes in the past, where β is the size of the client buffer, measured in minutes of video data. In this case, the client can begin receiving the complete stream right away, storing the data in its buffer. Simultaneously, it can receive the full tap stream and use it to display the first Δ minutes of the video. After that, the client can consume directly from its buffer, which will then always contain a moving Δ-minute window of the video. Stream tapping also defines *partial tap streams*, which can be used when $\Delta < \beta$. In this case clients must go through cycles of filling up and then emptying their buffer since the buffer is not large enough to account for the complete difference in video position.

To use tap streams, clients only have to receive at most two streams at any one time. If they can actually handle a higher bandwidth than this, they can use an option to the protocol called *extra tapping*. Extra tapping allows clients to tap data from any stream on the VOD server, and not just from complete streams. Figure 1 shows some sample streams from the VOD server's perspective. Stream *a* is a complete stream, and it must exist for the

entirety of the video. Stream *b* is a full tap stream starting Δ_b minutes after stream *a*. It only has to exist for Δ_b minutes. Stream *c* is another full tap stream, but it is able to use extra tapping to tap data from stream *b*, and so its service time is much smaller than Δ_c.

Eager and Vernon's *dynamic skyscraper broadcasting* (DSB) [5] is a reactive protocol based on Hua and Sheu's *skyscraper broadcasting* protocol [10]. Like skyscraper broadcasting, it never requires the STB to receive more than two streams at the same time. Their more recent *hierarchical multicast stream merging* (HMSM) protocol requires less server bandwidth than DSB to handle the same request arrival rate. Its bandwidth requirements are indeed very close to the upper bound of the minimum bandwidth for a reactive protocol that does not require the STB to receive more than two streams at the same time

$$\eta_2 \ln\left(1+\frac{N_i}{\eta_2}\right)$$

where $\eta_2 = (1+\sqrt{5})/2$ and N_i is the request arrival rate.

Selective catching combines both reactive and proactive approaches. It dedicates a certain number of channels for periodic broadcasts of videos while using the other channels to allow incoming requests to catch up with the current broadcast cycle. As a result, its bandwidth requirements are O(log($\lambda_t L_i$) where λ_t is the request arrival rate and L_t the duration of the video [7].

Partial preloading [15] loads in the customer STB the first few minutes of the top 10 to 20 videos in order to provide zero-delay access to these videos and reduce the server bandwidth of the broadcasting protocol distributing the remainder of the video.

3. Our Protocol

Stream tapping, dynamic skyscraper broadcasting and hierarchical multicast stream merging have two major advantages over broadcasting protocols. First, they provide true instant access to the video. Second, they require much less bandwidth than broadcasting protocols to distribute videos that are not requested more than ten times per hour for a two-hour video.

Unfortunately, the same is not true at higher request arrival rates. Since they handle all customer requests one by one, these three proactive protocols require much more bandwidth than most broadcasting protocols whenever the request arrival rate exceeds 30 to 60 requests per hour. One solution would be to batch requests together so that several incoming requests could share the same data streams. The sole problem with this approach is the delays it would introduce. As we said earlier, all three proactive protocols outperform the best broadcasting protocols when the rate remains below ten requests per hour for a two-hour video. Achieving the same performance at *any* request arrival rate would require a batching interval of six minutes, which means that customers wanting to watch a video would have to wait an average of three minutes. While smaller batching intervals are possible, they would have much less effect on the server bandwidth.

We propose another solution. Over the last few years, disk drive capacities have been doubling every eighteen months. One can now find 30 Gigabyte hard drives at retail prices below one hundred dollars. We can store on one of these hard drives more than 13 hours of video data assuming a very comfortable bandwidth of 5 Megabits per second. This would allow us to store the first 10 minutes of 80 videos or the first 15 minutes of 53 videos. Preloading the first few minutes of all videos in the server library would give us the same bandwidth reduction as batching all incoming requests and allow us to continue a zero-delay policy.

The impact of partial preloading on the protocol bandwidth is indeed so strong that it allows us to simplify the stream tapping protocol and eliminate all cases of extra tapping that require the customer STB to receive video data from more than two channels at the same time. This will greatly simplify the design of the STB disk controller and contribute to offset the cost of a larger disk drive.

Let us consider a video of duration *D* and let us assume that the first D_{pp} minutes of that video are already present in the customer STB. Four cases have to be considered:

1. When the first request for a video arrives at the server at time t_a, the server allocates a complete stream to the video. Since the customer STB already has the first D_{pp} minutes of the video on its hard drive, the complete stream can be delayed by D_{pp} minutes. As shown on Figure 2, the complete stream will start at time $t_c = t_a + D_{pp}$ and last until $t_a + D$. Thanks to this delay, all requests arriving in the time interval $[t_a, t_c]$ will be able to get all the data they need from the complete stream.

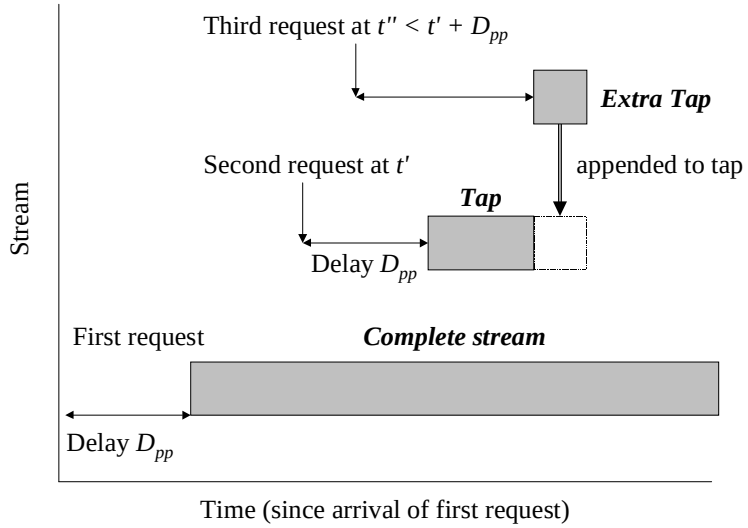

Figure 2. How the STPP protocol operates.

2. Assume now that a second request arrives at time t' such that $t_c < t' < t_a + D$. The request will still be able to "tap" the complete stream of the first request but will have missed the first $d_f = t' - t_c$ minutes of that stream. As shown on Figure 2, the server will thus schedule a full tap starting at time $t_f = t' + D_{pp}$ and ending at time $t' + D_{pp} + d_f$.
3. Consider now a third request arriving at time t'' such that $t' < t'' < t_f + D_{pp}$. That request will arrive before the beginning of the full tap that was scheduled for the second request. It will be able to find most data it needs from the complete stream and the full tap but will still need $t'' - t'$ minutes of additional data. Rather than start an extra tap stream for that request, we will extend the duration of the last full tap by $t'' - t'$ minutes.
4. A fourth request arriving after the beginning of the full tap would either require a new full tap if it arrives before the end of the complete stream or a new complete stream if it arrives after that.

This informal description implicitly assumes that the server will always allocate a tap stream to any incoming request whenever it can, that is, as long as a complete stream is still active. Carter and Long [2] found that it was much more efficient to stop the tapping earlier and then start a new complete stream. To that effect, they proposed a criterion based on the average cost of all requests sharing the same complete stream. We found that their criterion did not work well in our case due to the presence of numerous cases where no tap was necessary.

As one can see in Figure 3, the STPP protocol uses a much simpler criterion: it restarts a new complete stream whenever the duration of the new tap stream exceeds a given threshold θ, which must satisfy the condition $\theta \leq D - D_{pp}$.

We have not yet discussed how the preloaded segments of each video are distributed to the customer STB's. The task of distributing these data will be assigned to one or two dedicated channels that will continuously broadcast the first D_{pp} minutes of all videos that are currently offered for viewing. Any change in the set of videos being broadcast will require each STB to download the first D_{pp} minutes of the new videos being offered and to store them on its hard drive. Our protocol will thus need a mechanism allowing the VOD server to notify the STB's that they have new data to download but this mechanism could be as simple as agreeing upon some predefined time. It might also be more practical not to require all customers to have in their STB the first D_{pp} minutes of all videos that are currently offered for viewing. Two good candidates for this option are less frequently requested videos and customers joining the service either for the first time or after a power failure. One of the most attractive options for reducing the storage costs of our protocol is to begin each program by one or two trailers announcing future releases. This would not be very different of what is being done today in movie theaters, on videocassettes or on DVDs. The main advantage of this solution is that a small number of trailers could be shared among a much larger number of videos.

Assumptions:

D is the duration of the video
first D_{pp} minutes of video are preloaded in the customer STB
t_c is start time of last complete stream
t_f is start time of last full tap
d_f is duration of last full tap
$\theta \leq D - D_{pp}$ is policy threshold
a new request arrives at time t_a

Algorithm:

if $t_a \leq t_c$ **then**

STB will receive data from complete stream starting at t_c

else if $t_a \leq t_f$ **then**

increase duration of last full tap by $t_a - (t_c + d_f)$
STB will receive data from the complete stream starting at t_c
and the full tap starting at t_f

else if $t_a - t_c \leq \theta$ **then**

start full tap at $t_f = t_a + D_{pp}$
set full tap duration $d_f = t_a - t_c$
STB will receive data from complete stream starting at t_c
and full tap starting at t_f

else

start complete stream at $t_c = t_a + D_{pp}$
STB will receive data from complete stream starting at t_c

endif

Figure 3. The stream tapping protocol with partial preloading.

We saw in a previous example that we can store the first 10 minutes of 80 videos on a 30-Gigabyte hard drive. Requiring customers to watch a 3-minute trailer before each video would allow us to store on the same hard drive the first 7 minutes of 112 videos while leaving enough space for 5 different trailers.

4. Performance Analysis

To evaluate the performance of our protocol we wrote a simple simulation program assuming that request arrivals for a particular video were distributed according to a Poisson law. The program was written in CSIM and simulated requests for a single two-hour video. Since no data are shared among customers watching different videos, the total bandwidth of a server distributing several videos will always be equal to the sum of the bandwidths it dedicates to each video.

We considered six possible durations for the preloaded part of the video, namely 3 minutes, 5 minutes, 8 minutes, 10 minutes, 15 minutes, and 20 minutes. We measured the average bandwidth of the STPP protocols at request arrival rates varying between one and one thousand requests per hour. We did not consider higher arrival rates as we found the protocol bandwidth requirements to stabilize around 60 requests per hour. Each simulation run involved 200,000 arrivals over a simulated time period of at least 200 hours. We assumed for all runs the same threshold value of 35 minutes for θ because it provided the best average results over all arrival rates and all durations of the preloaded part.

Our results are summarized in Figure 4. Request arrival rates are expressed in arrivals per hour and bandwidths are expressed in multiples of the video consumption rate. We assumed a video duration of two hours and an unlimited buffer size for stream tapping.

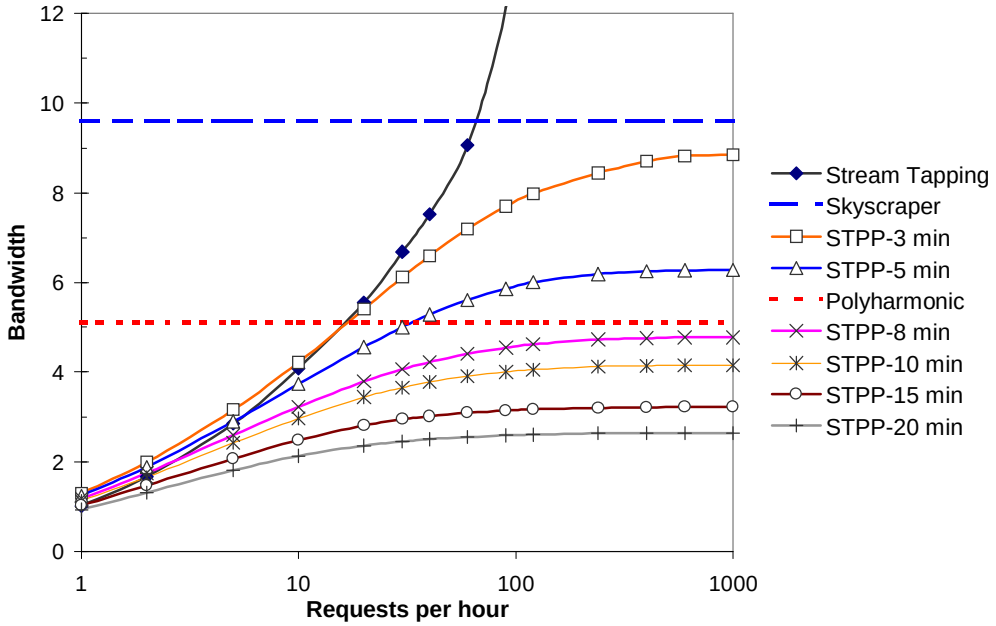

Figure 3. Compared average bandwidth requirements of stream tapping, skyscraper broadcasting, new pagoda broadcasting and stream tapping with partial preloading for a two-hour video and various durations of the preloaded portion of the video.

The two horizontal lines represent the bandwidth requirements of the *skyscraper broadcasting* and the *polyharmonic broadcasting* protocols for a maximum waiting time of one minute. Both protocols provide interesting benchmarks against which to compare the bandwidth requirements of our STPP protocol. As we saw earlier, *skyscraper broadcasting* [10] is the sole broadcasting protocol that never requires the STB to receive more than two streams at the same time. Hence it has the same client bandwidth requirements as our STPP protocol. *Polyharmonic broadcasting* [14] is the broadcasting protocol that requires the least amount of server bandwidth to guarantee a given maximum waiting time. The price to pay for this excellent performance is higher demands on the STB hardware, as the STB has to receive data broadcast over one hundred to one thousand low bandwidth channels.

As Figure 4 indicates, our STTP protocol only requires three minutes of preloaded data per video to perform better than skyscraper broadcasting at every request arrival rate. Increasing the duration of these preloaded data to eight minutes suffices to guarantee that STPP will also outperform polyharmonic broadcasting.

These results are even more impressive if we consider that the STTP protocol provides instant access to the video while skyscraper broadcasting introduces an average delay of 30 seconds and polyharmonic broadcasting imposes a fixed delay of one minute. In addition, the polyharmonic protocol requires each STB to receive data from the video server at a maximum bandwidth equal to 5.18 times the video consumption rate.

Even lower bandwidths can be achieved by increasing the duration of the preloaded segment to 10, 15 or 20 minutes. Unfortunately, these bandwidth savings obey to the law of diminishing returns because equal amounts of additional storage space result in ever decreasing bandwidth savings. For this reason, we see little motivation for preloading much more than 15 minutes of video data per video.

While the STTP protocol requires less bandwidth than stream tapping whenever the request arrival exceeds 20 requests per hour, it does not compare as favorably with stream tapping at arrival rates. The worst performance gap occurred at one request per hour where our protocol required up to 29 percent more bandwidth than stream tapping. We should note that this only occurred when the protocol preloaded three minutes of video data per video. In fact, the gap becomes insignificant when the protocol preloads at least 15 minutes of data per video.

Two factors can explain this performance gap. First, the stream tapping protocol we used as benchmark allows unlimited extra tapping and requires a STB capable of receiving video data at more than twice the video consumption rate. Second, the fixed 35-minute threshold we used in all our simulations resulted in a rather poor

threshold at low request arrival rates. Increasing the threshold to 70 minutes at one request per hour would reduce the bandwidth requirements of our protocol by 9 to 10 percent. As a result a better-tuned STPP protocol preloading at least 15 minutes of video data would have a lower bandwidth requirements than stream tapping for all request arrival rates. Achieving even lower bandwidths would require a more complex protocol integrating some features from *optimal stream merging* [5, 6].

A last issue to address is the asymptotic performance of the protocol at very high customer arrival rates. This is clearly an exceptional situation. It could only happen in a large metropolitan and would require a large percentage of the customer base to watch the same video the same night.

Under very heavy load, the STPP protocol will continuously repeat a cycle starting with the arrival of a request causing a new complete stream and ending with the arrival of the next request causing a new complete stream. Let t_c and $t_c + \Delta$ respectively designate the arrival times of these two requests.

Since the complete stream resulting from the first request of the cycle will be delayed by D_{pp} minutes, all requests arriving during the D_{pp} first minutes of the cycle will get all their data from the compete stream. All requests arriving during the next D_{pp} minutes, that is within the time interval $(t_c + D_{pp}, t + 2D_{pp}]$ will have missed at most D_{pp} minutes from the complete stream and will share a common tap stream of duration D_{pp} minutes.

Similarly all requests that arrive during the following D_{pp} minutes will have missed at most $2D_{pp}$ minutes from the complete stream and will share a common tap stream of duration $2D_{pp}$ minutes. More generally, all requests arriving within the time interval $(t_c + kD_{pp}, t_c + (k+1)D_{pp}]$ with $0 \le k < \theta/D_{pp} + 1$ will have missed at most kD_{pp} minutes from the complete stream and will share a common tap stream of duration kD_{pp} minutes.

The cycle will end when $k \ge \theta/D_{pp} + 1$ because the protocol will then restart a new complete stream rather than starting a tap of initial duration $d \ge \theta$.

The total duration of the cycle will thus be equal to $\Delta = (k_{\max} + 1)D_{pp}$ with

$$k_{\max} = \left\lceil \frac{\theta}{D_{pp}} \right\rceil$$

and the duration of all the data streams initiated in that time interval equal to

$$T = (D - D_{pp} + \sum_{k=1}^{k_{\max}} (k-1)D_{pp})b$$

where b represents the video consumption rate.

The average bandwidth B of the protocol is then obtained by dividing T by Δ, which gives

$$B = \frac{T}{\Delta} = b\frac{(D - D_{pp} + \sum_{k=1}^{k_{\max}} (k-1)D_{pp})}{(k_{\max} + 1)D_{pp}}$$

For instance, a STPP protocol preloading the first eight minutes of each video will never require more than 4.8 times the video consumption rate to distribute a two-hour video, which is well below the requirements of the polyharmonic broadcasting protocol [14].

5. Discussion

The critical assumption in our proposal is the necessity of having a disk drive in each customer STB. It will necessarily impact the cost of this STB and would result in additional outlays. One may therefore wonder whether the savings in server bandwidth will be sufficient to compensate these outlays.

The best answer to this question is mentioning the several additional advantages of having a very large buffer in each STB. First, we could keep in the STB buffer the previously viewed portion of each video. This would allow a very inexpensive implementation of interactive VOD because the STB could handle all *pause* and *rewind* requests [16]. The video server would still have to handle *fast forward* commands but we can expect this command to be infrequently used by most viewers. The STB could indeed store in its buffer the last thirty to sixty minutes of any program being watched to provide the same *pause* and *rewind* features as a ReplayTV [17], a TiVo[18], or an UltimateTV [19] STB.

A very large buffer would also allow a more efficient utilization of the channel bandwidth. Future VOD services will distribute videos in compressed form and the bandwidth requirements of these videos will depend on the rate at which the images being displayed change [1, 8]. For instance, daytime action scenes and cartoons will require more bandwidth than slower moving scenes and night scenes. The buffer sizes of conventional diskless STBs are limited by memory cost considerations. Hence buffer overflow is as a serious concern as buffer underflow. As a result, the video server will have to transmit at a lower bandwidth during slower moving scenes. A STB with a disk drive will not have that problem. Hence the video server will always transmit at the maximum bandwidth that has been allocated to the channel. Most video data would arrive sufficiently ahead of time to reduce or even eliminate bandwidth fluctuations [13].

6. CONCLUSION

Most existing distribution protocols for video-on-demand are tailored for a specific range of video request arrival rates and perform poorly beyond that range.

Reactive protocols like stream tapping, patching or dynamic skyscraper do not anticipate user requests and perform best when the request arrival rate does not exceed twenty requests per hour for a two-hour video. Above that, they require much more bandwidth than broadcasting protocols.

We have presented a stream tapping protocol that preloads in the customer set-top box the first few minutes of all popular videos. As a result, our stream tapping with partial preloading (STPP) protocol provides true instant access to the videos. In addition, it never requires the customer STB to receive video data at more than twice the video consumption rate.

We found that our STPP protocol only needed to have the first three minutes of a two-hour video preloaded in the customer STB to be able to distribute it at a lower cost than the best existing broadcasting protocols. We also found that our protocol performed much better than stream tapping with unlimited extra tapping at high request arrival rates. In addition, an STTP protocol preloading the first eight minutes of the same two-hour video would always outperform the best harmonic broadcasting protocols.

References

[1] Beran, J., R. Sherman, M. Taqqu, and W. Willinger. Long-range dependence in variable bit-rate video traffic. *IEEE Transactions on Communications*, 43:1566–1579, 1995.

[2] Carter, S. W. and D. D. E. Long. Improving video-on-demand server efficiency through stream tapping. *Proceedings of the 5th International Conference on Computer Communications and Networks*, pages 200-207, Sep. 1997.

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

[4] Dan, A., 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):168–179, Nov. 1994.

[5] Eager, D. L. and M. K. Vernon. Dynamic skyscraper broadcast for video-on-demand. *Proceedings of the 4th International Workshop on Advances in Multimedia Information Systems*, pages 18–32, Sep. 1998.

[6] Eager, D. L., M. K. Vernon and J. Zahorjan. Minimizing bandwidth requirements for on-demand data delivery. *Proceedings of the 5th International Workshop on Advances in Multimedia Information Systems*, , Oct. 1999.

[7] Gao, L., Z.-L Zhang and D. Towsley. Catching and selective catching: efficient latency reduction techniques for delivering continuous multimedia streams. *Proceedings of the 1999 ACM Multimedia Conference*, pages 203–206, Nov. 1999.

[8] Garrett, M. and W. Willinger. Analysis, modeling and generation of self-similar VBR video traffic. *Proceedings of the ACM SIGCOMM '94 Conference*, pages 269–280, Aug. 1994.

[9] Golubchik, L., J. Lui, and R. Muntz. Adaptive piggybacking: a novel technique for data sharing in video-on-demand storage servers. *ACM Multimedia Systems Journal*, 4(3): 140–155, 1996.

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

[11] Hua, K. A., Y. Cai, and S. Sheu. Patching: a multicast technique for true video-on-demand services. *Proceedings of the 6th ACM Multimedia Conference*, pages 191–200, Sep. 1998.

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

[13] McManus, J. M. and K. W. Ross, "Video-on-demand over ATM: constant rate transmission and transport," *IEEE Journal on Selected Areas in Communication*, 14(6):1087–1098, 1996.

[14] Pâris, J.-F., S. W. Carter and D. D. E. Long. A low bandwidth broadcasting protocol for video on demand. *Proceedings of the 7th International Conference on Computer Communications and Networks* (ICCCN '98), pages 690–697, Oct. 1998.

[15] Pâris, J.-F., D. D. E. Long and P. E. Mantey. A zero-delay broadcasting protocol for video on demand. *Proceedings of the 1999 ACM Multimedia Conference*, pages 189–197, Nov. 1999.

[16] Pâris, J.-F. An interactive broadcasting protocol for video-on-demand, *Proceedings of the 20th International Performance of Computers and Communication Conference*, pages 657–664, April 2001.

[17] ReplayTV. http://www.replay.com/.

[18] TiVo Technologies. http://www.tivo.com/.

[19] UltimateTV. http://www.ultimatetv.com/.

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