INVOLVING CLIENTS IN THE DISTRIBUTION OF VIDEOS ON DEMAND

Santosh Kulkarni *Jehan-François Pâris*

Department of Computer Science,
University of Houston, Houston, TX 77204-3010

ABSTRACT

We present a stream tapping protocol that involves clients in the video distribution process. As in conventional stream tapping, our protocol lets new clients tap the most recent broadcast of the video they are watching. While conventional stream tapping required the server to send to these clients the part of the video they missed, our protocol delegates this task to the clients that are already watching the video, thus greatly reducing the workload of the server. Unlike previous solutions involving clients in the video distribution process, our protocol works with clients that can only upload video data at a fraction of the video consumption rate and includes a mechanism to control its network bandwidth consumption.

1. INTRODUCTION

Distributing videos on demand is a costly proposition, mostly because of the high bandwidth requirements of the service. Assuming that the videos are in MPEG-2 format, each user request will require the delivery of approximately six megabits of data per second. Hence, a video server allocating a separate stream of data to each request would need an aggregate bandwidth of six gigabits per second to accommodate one thousand overlapping requests.

This situation has led to numerous proposals aimed 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. 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.

Nearly all these proposals assume a clear separation of functions between the server, which distributes the video, and the customers, who watch it on their personal computers or on their television sets. They do not take advantage of the upstream bandwidth of the clients to lower thee server's workload.

The stream tapping protocol we present here is the first protocol that can harness the collective bandwidth of clients with limited individual upstream bandwidths. As in conventional stream tapping, our protocol requires the server to start a new video broadcast whenever a client cannot get enough video data by "tapping" a previous broadcast of the same video. Unlike conventional stream tapping, our protocol uses the available upstream bandwidth of previous clients to reduce the amount of video data that the server will still have to send to the clients that "tap" a previous broadcast of the video. As we will see, delegating these tasks to the clients results in a dramatic reduction of the server workload at medium to high request arrival rates.

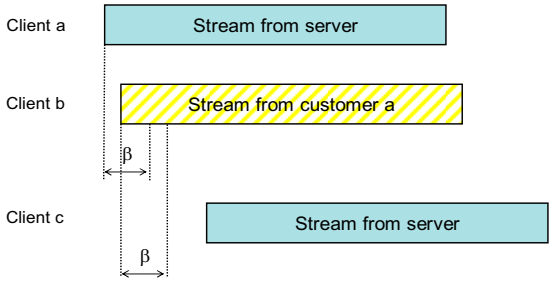

Fig. 1. How chaining works.

2. PREVIOUS WORK

Chaining [4] was the first video distribution protocol to take advantage of the upstream bandwidth of its clients. It constructs chains of clients such that (a) the first client in the chain receives its data from the server and (b) subsequent clients receive their data from their immediate predecessor. As a result, video data are "pipelined" through the clients belonging to the same chain. Since chaining only requires clients to have very small data buffers, a new chain has to be restarted every time the time interval between two successive clients exceeds the capacity β of the buffer of the first client. Fig. 1 shows three sample client requests. Since client *a* is the first customer, it will get all its data from the server. As client *b* arrives less than β minutes after customer *a*, it can receive all its data from client *a*. Finally client *c* arrives more than β minutes after client *a* and must be serviced directly by the server.

The cooperative video distribution protocol [2] extends the chaining protocol by taking advantage of the larger buffer sizes of modern clients. If all clients have buffers large enough to store the entire video, the server will never have to transmit video data at more than the video consumption rate.

Stream tapping [1] requires each client set-top box to have a buffer capable of storing at least 10 to 15 minutes of

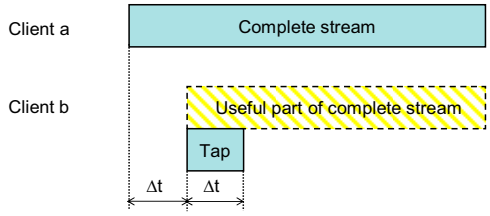

Fig. 2. How stream tapping works.

video data and to be able to receive data at at least twice the video consumption rate. This buffer will allow the set-top box to "tap" into data streams that were originally created for previous clients, and then store these data until they are needed. In the best case, clients obtain most of their data from an existing stream.

In particular, stream tapping defines two types of streams. *Complete streams* broadcast a video in its entirety. *Full tap streams* can be used if a complete stream for the same video started $\Delta \leq b$ minutes in the past, where b is the size of the client buffer, measured in minutes of video data. In this case, the client begins receiving the complete stream right away, storing the data in its buffer. Simultaneously, it receives a full tap stream and uses it to display the first Δ minutes of the video. After that, the client will consume directly from its buffer.

Clients that can receive data at three times the video consumption rate can use an option of 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. Fig. 2 shows some sample client requests. As client a is the first client, it is serviced by a complete stream, whose duration is equal to the duration D of the video. Since client b arrives Δt minutes after client a, it can share $D - \Delta t$ minutes of the complete stream and only requires a full tap of duration Δt minutes.

3. OUR PROTOCOL

Both chaining and the cooperative protocol require clients capable of sending video data at the video consumption rate. As a result, they exclude most home-based clients because these clients typically have upstream bandwidths that are one eighth to one fourth of their downstream bandwidths. While these clients might be able to download video data at twice their video consumption rate, they might only be able to forward video data at one fourth to one half of that rate.

We wanted to develop a video distribution protocol that allowed clients to participate in the video distribution process even if they could only retransmit data at a fraction the video consumption rate. We thus assumed that:

1. Clients would be able to receive video data at twice their video consumption rate;
2. Clients would only be able to forward video data at a rate equal to a fraction α of the same video consumption rate;
3. Clients would not have to forward video data after they have finished watching that video;

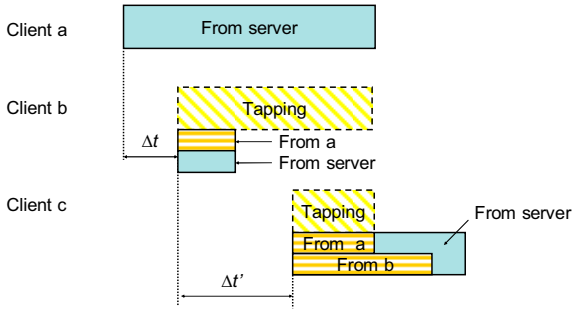

Fig. 3. How the full tap streams are distributed by the server and the previous clients.

4. Clients should have enough buffer space to store the previously viewed portion of the video they are watching until they have finished watching it.

As we can see, our protocol makes few demands on the transmission capabilities of the client hardware. In contrast, it requires client buffers capable of storing an entire video, that is, several gigabytes of compressed video data. Two factors motivated this choice. First, the diminishing cost of every kind of storage makes this requirement less onerous today than it would have been a few years ago. Second, we expected many clients to keep the previously viewed portion of the video they are watching in their buffer in order to provide the equivalent of a VCR rewind feature.

Our protocol is a fairly straightforward implementation of stream tapping without extra tapping as extra tapping would have required clients to be able to receive videos at three times the video consumption rate. It only differs from the original stream tapping protocol in the way it handles tap streams. While tap streams originally were the sole responsibility of the server, this task is now shared by the server and the previous client. Consider two consecutive requests for a video of duration D. Let T_c denote the time elapsed since the start of the last complete stream and Δt the time interval between the two requests:

1. If $T_c \geq D$, the two requests do not overlap and the second client cannot tap any data from the last complete stream. As in the original stream tapping protocol, the server will then start a new complete stream.
2. If $T_c < D$, there is an overlap between the current request and the last complete stream. As in the original stream tapping protocol, the server will then evaluate whether it would be more advantageous to keep tapping from the last complete stream or to start a new one. If the server decides to keep tapping from the last complete stream, it will have to provide the second client with a full tap stream of duration T_c. Two alternatives must now be considered:
   a. If $T_c \leq D - \Delta t$, the previous client will provide a fraction α of the full tap stream and the server the remaining $1 - \alpha$ fraction.
   b. If $T_c > D - \Delta t$, the previous client will finish watching the video before being able to transmit all its share of the full tap stream and the previous

client will only be able to transmit a fraction $\alpha (D - \Delta t) / T_c$ of the full tap stream with the server transmitting the remainder of the stream.

If the video is long enough, the new request is likely to overlap with more than one previous request. We propose to harness the available bandwidth of the clients that issued these requests in order to further reduce the workload of the server. The contributions of these clients will be subject to two restrictions. First, upstream bandwidth restrictions prevent any client to upload data for two different clients at the same time. Second, we will never require a client to transmit video data after the client has finished watching the video.

Consider for instance how the protocol would handle the three requests displayed in Fig. 3. The first request to the video will be entirely serviced by a complete stream coming from the server. The second request will get the last $D - \Delta t$ minutes of the video by tapping client *a*'s complete stream and the first Δt minutes from a full tap stream of duration Δt. A fraction α of this stream will be sent by customer *a* and the remaining $1 - \alpha$ fraction will come from the server. Assuming that the server decides not to start a new complete stream for customer *c,* that customer would get:

1. The last $D - (\Delta t + \Delta t')$ minutes of the video by tapping client *a*'s complete stream;
2. A fraction α of the first $D - (\Delta t + \Delta t')$ minutes of the video from a tap stream sent by customer *a*; this tap stream will end when customer *a* will finish watching the video $D - (\Delta t + \Delta t')$ minutes after the arrival of customer *c*;
3. A fraction α of the first $D - \Delta t'$ minutes of the video from a tap stream sent by customer *b*; this tap stream will end when customer *b* will finish watching the video $D - \Delta t'$ minutes after the arrival of customer *c*;
4. The remaining portion of the first $\Delta t + \Delta t'$ minutes of the video from the server.

One last issue to consider is when to halt tapping from the current complete stream and start a new one. To achieve this goal, our protocol keeps track of the minimum average request service time of all requests sharing the same complete stream. Before adding a new request to a group, it computes what would be the new average request service time of the group if the new request was added to the group. Should this new average request service time be lesser than or equal to the minimum average request service time of the group, our protocol adds the new request to the group; otherwise, it starts a new group. This criterion is similar but not identical to that used by Carter and Long [1, 2].

3.1 Handling Client Failures

To operate correctly, our protocol requires all clients to forward video data to the next customers for the same video. Any client failure will deprive all subsequent customers from their video data.

There is a simple solution to the problem. Let us return to the scenario of Fig. 3 where client *c* receives most of its tap stream from clients *a* and *b* while client *b* receives almost half of its tap stream from client *a*. Any failure of either client *a* or client *b* would immediately affect the correct flow of data to client *c*. A failure of client *a* will require the server to take over the role of client *a* and send the missing video data to clients *b* and *c*. A failure of client *b* would have less impact on the server workload as it would also free client *a* from its obligation to send client *b* a fraction of its tap stream, thus freeing enough upstream bandwidth to let client *a* take over the role of client *b* and send most of the missing video data to client *c*. Making the protocol fault-tolerant will thus require the server to keep track of which client is sending video data to each client.

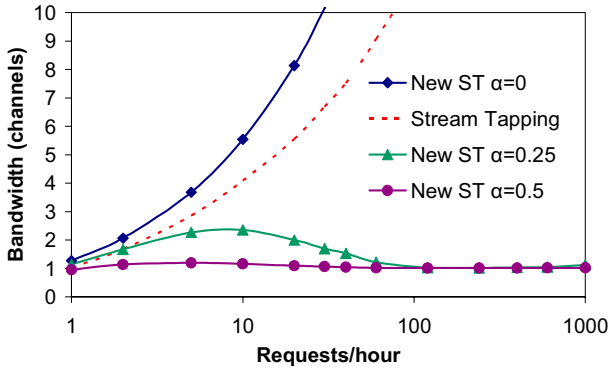

Fig. 4. Server bandwidth requirements of the new stream tapping protocol.

4. PERFORMANCE EVALUATION

Fig. 4 displays the server bandwidth requirements of our new stream tapping protocol for selected values of α and request arrival rates varying between one and one thousand requests per hour. All bandwidths are expressed in multiples of the video consumption rates. We assumed that the server was broadcasting a two-hour video and that request arrivals could be modeled by a Poisson process.

In addition, the dotted line represents the server bandwidth requirements of the original stream tapping protocol with extra tapping. Let us observe that the comparison between the two protocols is not totally fair since extra tapping requires clients capable of receiving video data at three times the video consumption rate, while our protocol only requires clients capable of receiving video data at two times that rate.

As we can see, our new stream tapping protocol outperforms conventional stream tapping even when clients can only forward data at one fourth of the video consumption rate, that is, when $\alpha = 0.25$. These results are much better than those of an earlier version of the protocol that would not allow clients to receive video data from more than one client [3].

This excellent performance comes however at a stiff price. As seen on Fig. 5, the network bandwidth requirements of our stream tapping protocol increase much

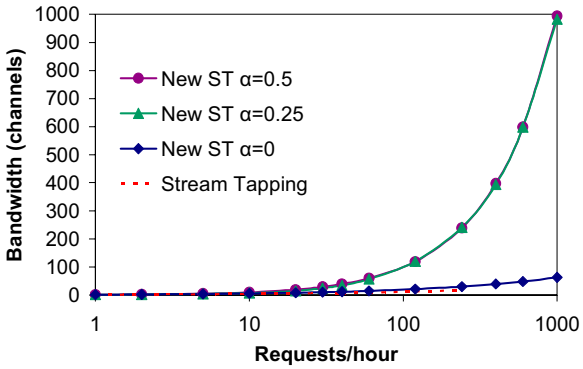

Fig. 5. Network bandwidth requirements of the new stream tapping protocol.

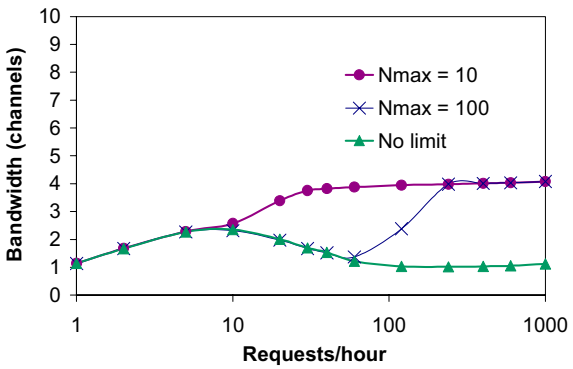

Fig. 6. Server bandwidth requirements of the protocol for $\alpha = 0.25$ and $\beta_{max} = 0.25$.

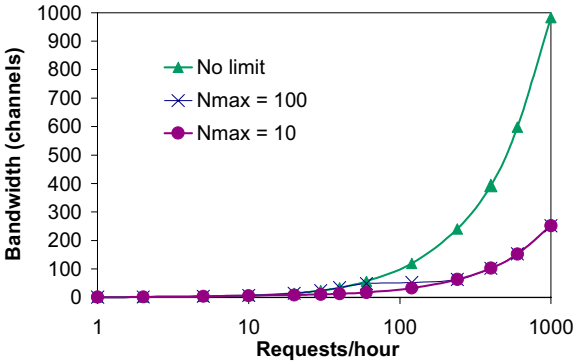

Fig. 7. Network bandwidth requirements of the protocol for $\alpha = 0.25$ and $\beta_{max} = 0.25$.

more rapidly than those of the original stream tapping protocol when the client request arrival rate exceeds ten requests per hour. This phenomenon can be explained in part by the fact that our protocol does not allow extra tapping. A more important factor is the way the server decides when to start a new complete stream. Since the clients handle most of the tap streams, adding extra requests to any existing group has a negligible impact on the server workload. As a result, the server will not start a new complete stream before the end of the previous one. Thus the average duration of a tap stream is equal to half the duration of the video and the average network bandwidth is roughly equal to one half the bandwidth required by a unicast scheme.

A simplistic solution to this problem would be to limit the size of the tap streams to a fraction β_{max} of the duration of the video. This would reduce the average duration of these streams and proportionally reduce the network bandwidth. This solution would however affect the performance of the protocol at low arrival rates, where long tap streams are the norm. Having investigated several other options, we found out that the best way to limit the growth of the network bandwidth was to limit the size of the tap streams at high arrival rates. We did not want to complicate the design of the server by requiring it to maintain some moving average of the request arrival rates for each video. We decided instead to use as threshold the number of clients sharing the same complete stream and force the server to start a new complete stream whenever (a) the size of the tap stream would otherwise exceed a fraction β_{max} of the duration of the video and (b) more than N_{max} requests were already sharing the current complete stream.

Fig. 6 and 7 display the impact of this modification to the server and network bandwidth of our protocol. We considered clients capable of uploading data at one-fourth the video consumption rate and set our β_{max} to 0.25. Each individual curve corresponds to a different value of N_{max}. We see that limiting the tap stream length to one fourth of the video duration reduces by a factor of four the network bandwidth of the protocol while increasing the server bandwidth at the highest arrival rates by the same factor. Even under these conditions the server bandwidth remains well below that of the original stream tapping protocol.

5. CONCLUSIONS

We have presented a stream tapping protocol that involves clients in the video distribution process. Our protocol is tailored to environments where client machines are able to download video data at twice the video consumption rate but can only forward video data at a fourth to a half of that rate. We observed that our technique achieved a dramatic reduction of the server workload at medium to high request arrival rates but also resulted in much higher network bandwidth consumptions. These increases can however be controlled by requiring the server to restart complete streams at some specific intervals.

References

[1] Carter, S. W. and D. D. E. Long. Improving video-on-demand server efficiency through stream tapping. *Proc. 5th ICCCN Conf.*, pp. 200–207, Sep. 1997.

[2] Pâris, J.-F. A Cooperative Distribution Protocol for Video-on-Demand. *Proc. 6th Mexican Int'l Conf. on Computer Science*, pp. 240–246., Sep. 2005.

[3] Pâris, J.-F. Using Available Client Bandwidth to Reduce the Distribution Costs of Video-on-Demand Services. *Proc. 7th WDAS Workshop*, Jan. 2006.

[4] Sheu, S., K. A. Hua, and W. Tavanapong. Chaining: A Generalized Batching Technique for Video-on-Demand Systems. *Proc. ICMS Conf.*, pp. 110-117, June 1997.