Hindawi Publishing Corporation
Advances in Multimedia
Volume 2008, Article ID 265309, 9 pages
doi:10.1155/2008/265309

Research Article

A Stream Tapping Protocol Involving Clients in the Distribution of Videos on Demand

Santosh Kulkarni, Jehan-François Pâris, and Purvi Shah

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

Correspondence should be addressed to Jehan-François Pâris, paris@cs.uh.edu

Received 1 May 2007; Revised 7 November 2007; Accepted 11 January 2008

Recommended by Qian Zhang

We present a stream tapping protocol that involves clients in the video distribution process. As in conventional stream tapping, our protocol allows new clients to 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. As a result, they cannot take advantage of the upstream bandwidth of the clients to lower the 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.

The remainder of this paper is organized as follows Section 2 reviews previous work Section 3 introduces our stream tapping protocol Section 4 discusses its performance and shows how we can limit the network bandwidth consumption of the protocol at high arrival rates Section 5 presents a simple probabilistic model of our protocol and Section 6 discusses its applicability to actual networks Finally, Section 7 has our conclusions

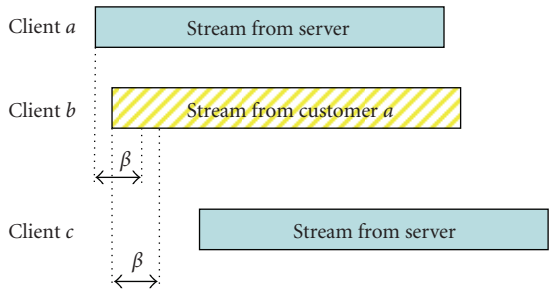

Figure 1: How chaining works.

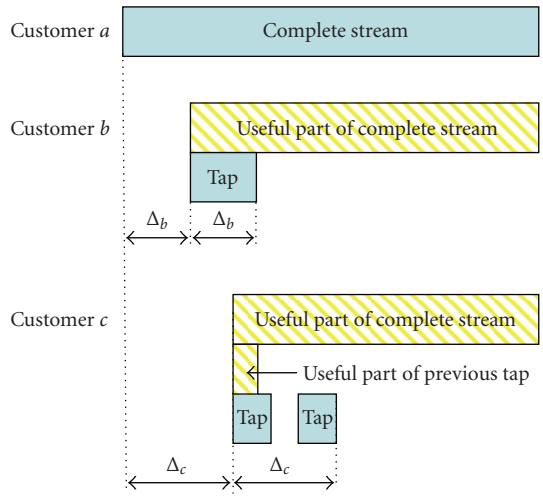

Figure 2: How stream tapping works.

2. PREVIOUS WORK

Two of the earliest reactive distribution protocols are batching and piggybacking. *Batching* [1] 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. *Piggybacking* [2] 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.

Chaining [3] 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 Figure 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. *Optimized chaining* [4] exploits the buffers of other clients in order to construct longer chains and reduce the server workload.

The cooperative video distribution protocol [5] extends the chaining protocol by taking advantage of the larger buffer sizes of modern clients Hence, it should be better named *extended chaining* 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 [6, 7], also known as *patching* [8], requires each client set-top box to have a buffer capable of storing at least 10 to 15 minutes of 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 $\beta \leq b$ 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 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 need only receive at most two streams at any one time. 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. Figure 2 shows some samples of 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 Finally, customer c can use extra tapping to tap data from both the complete stream and the previous full tap, and so its service time is smaller than Δ_c.

Eager and Vernon's *dynamic skyscraper broadcasting* (DSB) [9] is another reactive protocol based on Hua and Sheu's *skyscraper broadcasting* protocol [10, 11] 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 [12] 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, that is,

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

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

Selective catching [13] 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 $O(\log(\lambda_i L_i))$, where λ_i is the request arrival rate and L_i the duration of the video.

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 clients that have limited upstream bandwidths, say, no more than 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 of the video consumption rate [14] We thus assumed the following:

(1) clients would be able to receive video data at twic 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 be able to multicast video data,

(4) clients would not have to forward video data after they have finished watching that video,

(5) 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 buffers 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 its tap streams While tap streams originally were the sole responsibility of the server, this task is now shared by the server and client b. Let us consider the scenario described in Figure 3 and focus on the request issued by client c. Let $\Delta t'$ denote the time interval between that request and a request issued by client b, and let T_c denote the time elapsed since the start of the last request that was serviced by a complete stream.

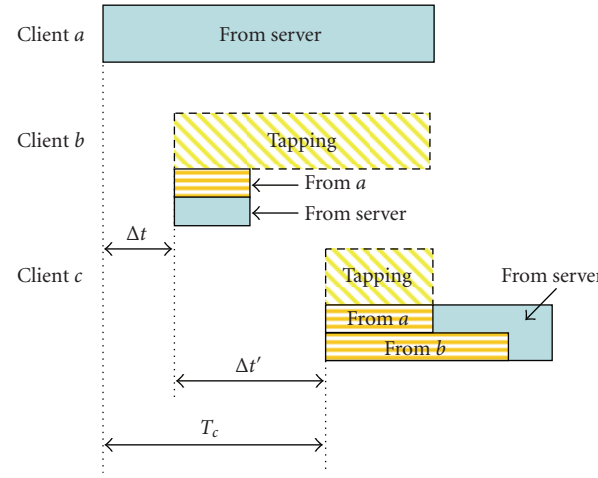

FIGURE 3: How the full tap streams are distributed among the server and the previous clients.

(1) If $T_c \geq D$, the two requests do not overlap and client c 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 client c with a full tap stream of duration T_c Two alternatives must now be considered:

(a) if $T_c \leq D - \Delta t'$, client b will provide a fraction α of the full tap stream and the server the remaining $1 - \alpha$ fraction of the stream;

(b) if $T_c > D - \Delta t'$, client c will finish watching the video before being able to transmit all its share of the full tap stream, and 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.

In our example, the request form client a is entirely serviced by a complete stream coming from the server The request from client b gets 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 the following.

(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 [6, 7].

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 Figure 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.

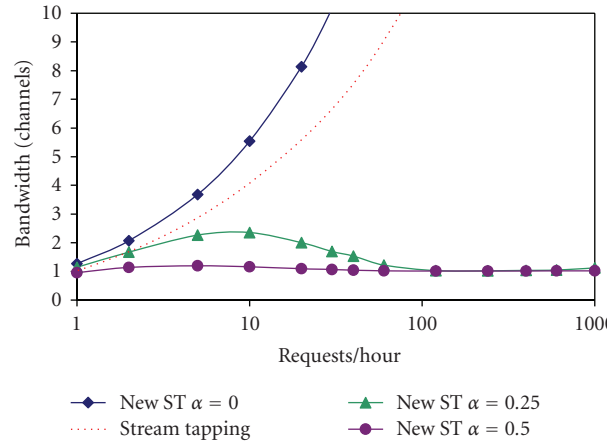

Figure 4: Server bandwidth requirements of the new stream tapping protocol.

4. PERFORMANCE EVALUATION

To evaluate the performance of our new protocol, we designed a simulator modeling the distribution of a two-hour video assuming that request arrivals could be modeled by a Poisson process [14]. It is based on similar simulators used for previous papers on stream tapping and was modified to model clients that could not forward video data at a rate equal to the video consumption rate [15, 16].

Figure 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 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 [16].

This excellent performance comes however at a stiff price As seen in Figure 5, the network bandwidth requirements of our stream tapping protocol increase much 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 never starts a new complete stream before the end

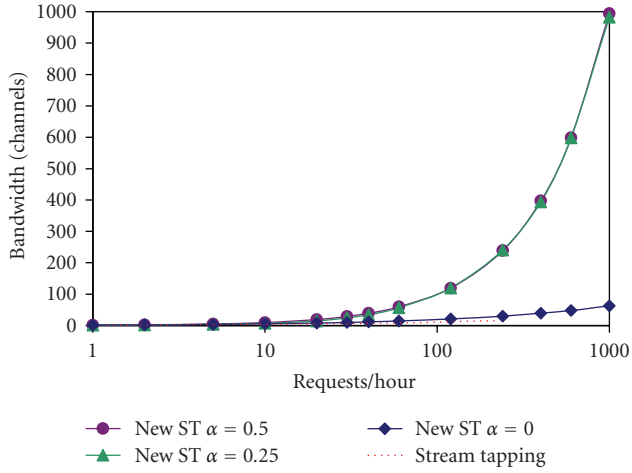

FIGURE 5: Network bandwidth requirements of the new stream tapping protocol.

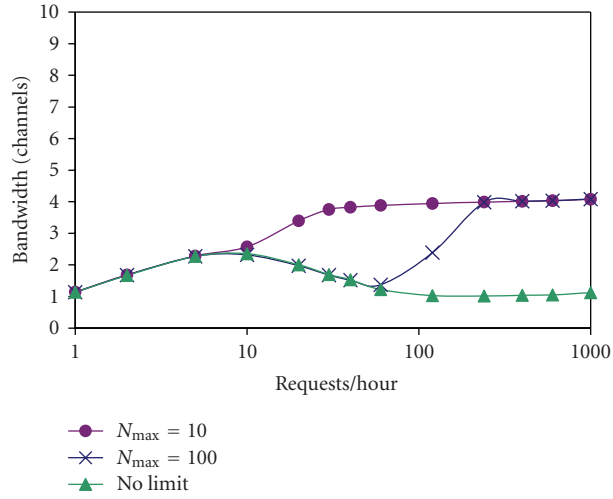

FIGURE 6: Server bandwidth requirements of the protocol for $\alpha = 0.25$, and $\beta_{max} = 0.25$.

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. At the same time, 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} o the duration of the video, and (b) more than N_{max} requests were already sharing the current complete stream.

Figures 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 quarter 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.

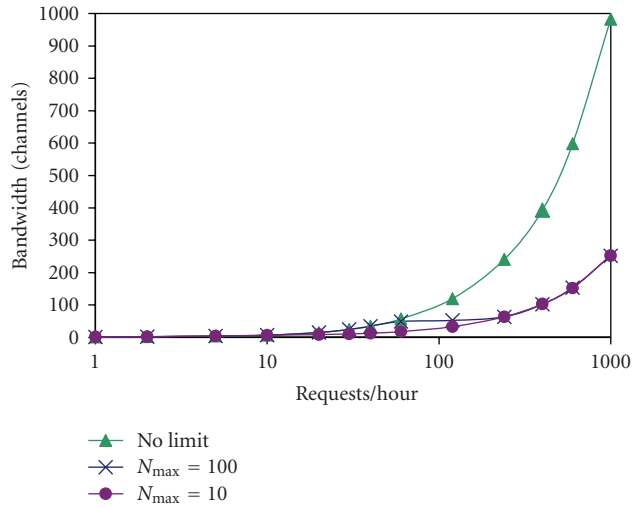

FIGURE 7: Network bandwidth requirements of the protocol for $\alpha = 0.25$, and $\beta_{max} = 0.25$.

5. AN ANALYTICAL MODEL

As in a previous paper [17], we consider stream tapping without extra tapping for a video of duration D that is being accessed at a rate λ. We assume that we will restart a new complete stream whenever the length of the next full tap exceeds βD, where $0 < \beta \leq 1$ is a parameter to be determined.

In other words, we will wait for an incoming request, start a complete stream of duration D, tap this stream over a time interval of duration βD, then restart the process.

During this time interval, the server will process an average of $\lambda\beta D$ requests in addition to the request that prompted the complete stream Hence the average number of requests sharing the same complete stream is

$$n_{avg} = 1 + \lambda\beta D. \quad (2)$$

Since the lengths of the $\lambda\beta D$ full tap streams in the group will be uniformly distributed over the interval $(0, \beta D]$, the average duration of each of these streams will be

equal to $\beta D/2$ The total duration of the streams required for processing these $1 + \lambda\beta D$ requests will thus be

$$W = D + (\lambda\beta D)\left(\frac{\beta D}{2}\right) = D + \frac{\lambda\beta^2 D^2}{2}. \quad (3)$$

We define the average service time T_{avg} for the requests in a group as the total duration W of the streams required for processing these requests divided by the number n_{avg} of requests in the group We will then have

$$T_{\text{avg}} = \frac{W}{n_{\text{avg}}} = \frac{D + \lambda\beta^2 D^2/2}{1 + \lambda\beta D}. \quad (4)$$

Our protocol assumes that previous clients will share with the server the task of sending the full tap streams Let $\gamma \leq 1$ denote the fraction of the tap streams that the clients will be able to send. This fraction γ will depend both on the factor α characterizing the upload bandwidth of the clients and the existence of previous clients whose requests overlap with the current requests. As the request arrival increases, the number of these clients will also increase, which will allow them to send a larger fraction γ of the tap streams.

For a given value of γ, the contribution of the server to the average request service time will be

$$T_{s,\text{avg}} = \frac{D + (1-\gamma)\lambda\beta^2 D^2/2}{1 + \lambda\beta D}, \quad (5)$$

and that of the previous clients will be

$$T_{c,\text{avg}} = \frac{\gamma\lambda\beta^2 D^2/2}{1 + \lambda\beta D}. \quad (6)$$

To find the value β_{opt} of the coefficient β that minimizes the server workload, we differentiate (5) with respect to β, obtaining

$$\frac{(1-\gamma)\lambda\beta D^2(1+\lambda\beta D) - \lambda D(D + (1-\gamma)\lambda\beta^2 D^2/2)}{(1+\lambda\beta D)^2}, \quad (7)$$

the only positive root of which

$$\beta_{\text{opt}} = \frac{\sqrt{2\lambda D(1-\gamma) + (1-\gamma)^2} - (1-\gamma)}{\lambda D(1-\gamma)}. \quad (8)$$

When γ equals zero, clients do not participate in the sending of full tap streams and (5) reverts to

$$\beta_{\text{opt}} = \frac{\sqrt{2\lambda D + 1} - 1}{\lambda D}. \quad (9)$$

When γ equals 1, clients handle all full tap streams, the server only handles the complete streams, and β_{opt} goes to infinity As a result, the server will never start a new complete stream before the end of the previous one, and the server bandwidth $B_{s,\text{ST}}$ will never exceed one channel.

To estimate the total bandwidth requirements of the clients when $\gamma = 1$, let us consider, let us consider an interval of duration $2D$ starting with the beginning of a new complete stream During this time interval, the system will receive

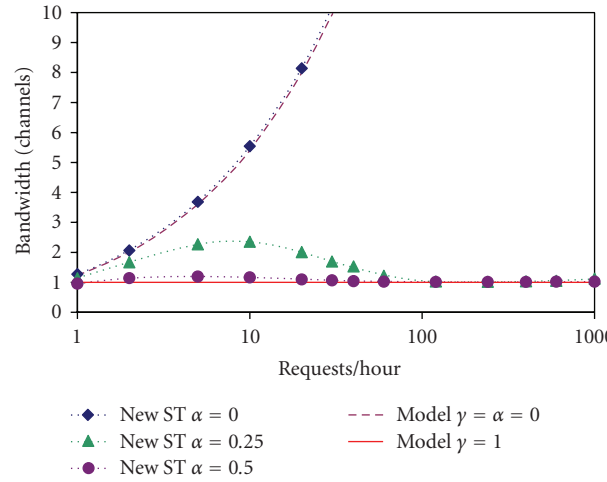

Figure 8: Server bandwidth requirements of the new stream tapping protocol.

an average of $2\lambda D$ requests in addition to the request that prompted the complete stream and the clients will send an average of $2\lambda D$ full tap streams. Since the lengths of these full tap streams will be uniformly distributed over the interval $(0, D)$, the average duration of each of these streams will be equal to $D/2$ The total duration of these streams will thus be

$$W_c = (2\lambda D)\left(\frac{D}{2}\right) = \lambda D^2. \quad (10)$$

The average bandwidth $B_{c,\text{ST}}$ used by the clients to send full taps streams is then given by dividing this expression by the duration of the considered interval,

$$B_{c,\text{ST}} = \frac{W_c}{2D} = \frac{\lambda D}{2}. \quad (11)$$

The total network bandwidth requirements B_{ST} of our protocol are obtained by adding the average bandwidth $B_{s,\text{ST}}$ used by the server to that expression As we can see in Figure 4, $B_{s,\text{ST}}$ is very close to on for large values of λ. We can thus approximate B_{ST} as

$$B_{\text{ST}} = B_{c,\text{ST}} + B_{s,\text{ST}} \approx \frac{\lambda D}{2} + 1. \quad (12)$$

Compare these requirements with those of a video distribution scheme that does not use any form of multicast Over an interval of duration T, it will handle an average of λT requests. Since each request will be serviced by a separate stream of duration D, the total duration of these streams will be λDT, and the average bandwidth requirements of the scheme would be λD, that is, almost twice the average bandwidth B_{ST} requirements of our cooperative stream tapping protocol.

Figures 8 and 9 compare our analytical results with our simulation results We considered the two limit cases when $\gamma = 0$ and $\gamma = 1$ As we can see, both models predict identical network bandwidths for all values of α and all arrival rates.

This is not the case for server bandwidths Both models are in substantial agreement when either $\alpha = 0$ or $\alpha \geq 0.5$ and disagree when $0 < \alpha < 0.5$ Let us consider these three cases.

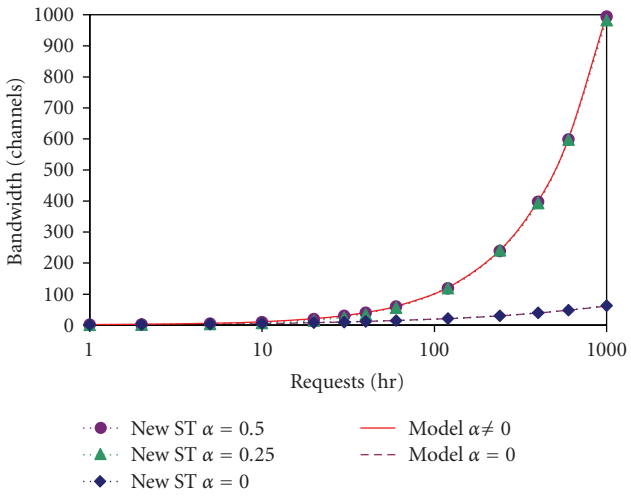

Figure 9: Network bandwidth requirements of the new stream tapping protocol.

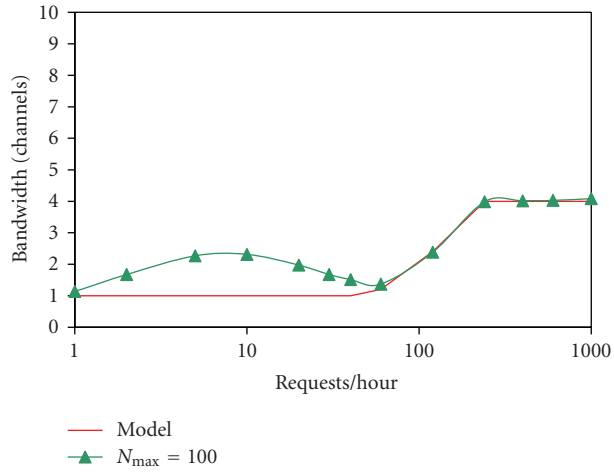

Figure 10: Server bandwidth requirements of the protocol for $\alpha = 0.25$, $\beta_{max} = 0.25$, and $N_{max} = 100$.

(1) $\alpha = 0$

When $\alpha = 0$, the clients cannot upload video data Hence, $\gamma = 0$ and both models predict identical network bandwidths.

(2) $\alpha \geq 0.5$

When $\alpha \geq 0.5$, the clients can upload video data at at least half the video consumption rate. As a result, they can handle most, if not all, of the tap streams, and their share γ of the stream tapping data is close to one. This is less true for request arrival rates below 20 requests per hour, where the server must still handle a small fraction of the tap streams.

(3) $0 < \alpha < 0.5$

When $0 < \alpha < 0.5$, the clients have very limited upload bandwidths They participate in the distribution of tap streams but cannot fully substitute for the server as long as the request arrival rate remains below 60 requests per hour As a result, $0 < \gamma < 1$ and the actual server bandwidth stands somewhere between the values predicted for $\gamma = 0$ and $\gamma = 1$.

Let us turn now our attention to the performance of the scheme reducing the network bandwidth consumption of our protocol at high arrival rates Recall that this scheme consisted of starting 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.

On the average, this scheme will take effect whenever the request arrival rate λ will exceed a threshold,

$$\lambda^* = \frac{N_{max}}{\beta_{max} D}. \tag{13}$$

At higher arrival rates, the server will restart a new complete stream of duration each $\beta_{max} D$ time units Its bandwidth requirements will thus be equal to $1/\beta_{max}$ As a result, the average bandwidth requirements of the server will be

$$B_{s,ST} = \begin{cases} 1 & \text{for } \lambda < \lambda^*, \\ \dfrac{1}{\beta_{max}} & \text{for } \lambda \geq \lambda^*. \end{cases} \tag{14}$$

Let us now focus on the behavior of clients at arrival rates greater than or equal to λ_{max} Assuming again that the clients will handle all full tap streams, they will send an average of $\lambda \beta_{max} D$ full tap streams each $\beta_{max} D$ time units The average length of these tap streams will be equal to $\beta_{max} D/2$, and the average bandwidth $B_{c,ST}$ used by the client will be

$$B_{c,ST} = \begin{cases} \dfrac{\lambda D}{2} & \text{for } \lambda < \lambda^*, \\ \dfrac{\lambda \beta_{max} D}{2} & \text{for } \lambda \geq \lambda^*. \end{cases} \tag{15}$$

Figures 10 and 11 compare our analytical results with our simulation results for $\alpha = 0.25$, $\beta_{max} = 0.25$, and $N_{max} = 100$ As before, both models predict identical or near identical network bandwidths for all values of α and all arrival rates This is not the case for the server bandwidth Here again our analytical model underestimates the server bandwidth when the request arrival rate remains below 60 requests per hour because it incorrectly assumes that the clients can handle all tap streams.

We can thus say that the results of our probabilistic analysis confirm those obtained through our simulation study and conclude that our cooperative stream tapping protocol can effectively harness the collective upload bandwidth of its clients even when each individual client cannot upload video data at more than one quarter of the video consumption In addition, requiring the server to restart complete streams at fixed intervals provides an effective tool for limiting the network bandwidth consumption of the protocol.

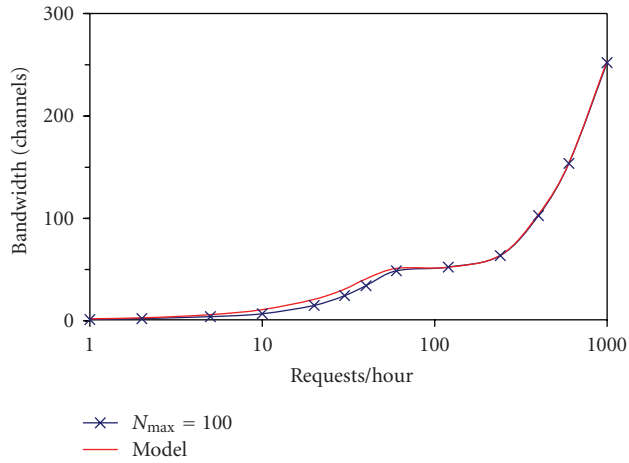

Figure 11: Network bandwidth requirements of the protocol for $\alpha = 0.25$, $\beta_{max} = 0.25$, and $N_{max} = 100$.

6. PROTOCOL APPLICABILITY

Many organizations such as universities and companies run their own multicast-capable networks Consider the example of a local telephone company that decides to increase its service offerings to include on-demand video entertainment Many of their existing clients are already subscribed to their high-speed internet access by means of DSL With our solution, the telephone company can make use of their multicast-enabled infrastructure to deliver videos on demand to a large number of clients at very low cost.

7. CONCLUSIONS

We have presented a stream tapping protocol that involves clients in the video distribution process Our protocol is tailored to multicast-capable environments where client machines are able to download video data at twice the video consumption rate but cannot necessarily forward video data at 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 Even then, the server bandwidth requirements of the protocol remain well below those of pure client-server solutions.

Our proposal differs from conventional peer-to-peer (P2P) solutions in two different ways First, it assumes the existence of one or more servers capable of multicasting video data at twice to four times the video consumption rate This allows the protocol to make much more efficient use of the network bandwidth as a single video stream can be broadcast to an arbitrary number of clients Second, it does not require peers to be able to upload video data at more than one quarter to one half of their video consumption rates, which would not be possible in a pure P2P solution such as [18] In pure P2P systems, the download rate is positively correlated to the upload rate Legout et al. [19] illustrated through experiments that the amount of data uploaded is very close to the data downloaded in P2P systems such as BitTorrent [20]. Thus, to sustain a video streaming service, a peer should have an upload rate at least equal the video consumption rate.

Stream tapping protocol involving clients is a very good solution, especially over local networks as it provides a scalable way to distribute on-demand high-bandwidth videos streams without overtaxing the network bandwidth.

REFERENCES

[1] 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*, vol. 30, no. 2, pp. 168179, 1995.

[2] 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*, vol. 4, no. 3, pp. 140155, 1996.

[3] S. Sheu, K. A. Hua, and W. Tavanapong, Chaining: a generalized batching technique for video-on-demand systems, in *Proceedings of the IEEE International Conference on Multimedia Computing and Systems (ICMCS '97)*, pp. 110117, Ottawa, Ontario, Canada, June 1997.

[4] T.-C. Su, S.-Y. Huang, C.-L. Chan, and J.-S. Wang, Optimal chaining scheme for video-on-demand applications on collaborative networks, *IEEE Transactions on Multimedia*, vol. 7, no. 5, pp. 972980, 2005.

[5] J.-F. Pâris, A cooperative distribution protocol for video-on-demand, in *Proceedings of the 6th Mexican International Conference on Computer Science (ENC '05)*, pp. 240246, Puebla, Mexico, September 2005.

[6] S. W. Carter and D. D. E. Long, Improving video-on-demand server efficiency through stream tapping, in *Proceedings of the 6th International Conference on Computer Communications and Networks (ICCCN '97)*, pp. 200207, Las Vegas, Nev, USA, September 1997.

[7] S. W. Carter and D. D. E. Long, Improving bandwidth efficiency of video-on-demand servers, *Computer Networks*, vol. 31, no. 1-2, pp. 111123, 1999.

[8] K. A. Hua, Y. Cai, and S. Sheu, Patching: a multicast technique for true video-on-demand services, in *Proceedings of the 6th ACM International Conference on Multimedia*, pp. 191200, Bristol, UK, September 1998.

[9] D. L. Eager and M. K. Vernon, Dynamic skyscraper broadcast for video-on-demand, in *Proceedings of the 4th International Workshop on Advances in Multimedia Information Systems*, pp. 1832, Istanbul, Turkey, September 1998.

[10] K. A. Hua and S. Sheu, Skyscraper broadcasting: a new broadcasting scheme for metropolitan video-on-demand systems, in *Proceedings of the ACM SIGCOMM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM '97)*, pp. 89100, Cannes, France, September 1997.

[11] K. A. Hua and S. Sheu, An efficient periodic broadcast technique for digital video libraries, *Multimedia Tools and Applications*, vol. 10, no. 2-3, pp. 157177, 2000.

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

[13] L. Gao, Z.-L. Zhang, and D. Towsley, Catching and selective catching: efficient latency reduction techniques for delivering continuous multimedia streams, in *Proceedings of the 7th ACM International Conference on Multimedia*, pp. 203206, Orlando, Fla, USA, October 1999.

[14] S. Kulkarni and J.-F. Pâris, Involving clients in the distribution of videos on demand, in *Proceedings of the IEEE International Conference on Multimedia and Expo (ICME '06)*, pp. 16771680, Toronto, Ontario, Canada, July 2006.

[15] J.-F. Pâris, A stream tapping protocol with partial preloading, in *Proceedings of the 9th IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS '01)*, pp. 423430, Cincinnati, Ohio, USA, August 2001.

[16] J.-F Pâris, Using available client bandwidth to reduce the distribution costs of video-on-demand services, in *Proceedings of the 7th Workshop on Distributed Data and Structures (WDAS '06)*, Santa Clara, Calif, USA, January 2006.

[17] J.-F. Pâris and D. D. E. Long, An analytic study of stream tapping protocols, in *Proceedings of the IEEE International Conference on Multimedia and Expo (ICME '06)*, pp. 12371240, Toronto, Ontario, Canada, July 2006.

[18] P. Shah and J.-F. Pâris, Peer-to-peer multimedia streaming using BitTorrent, in *Proceedings of the 26th IEEE International Performance, Computing, and Communications Conference (IPCCC '07)*, pp. 340347, New Orleans, La, USA, April 2007.

[19] A. Legout, G. Urvoy-Keller, and P. Michiardi, Understanding BitTorrent: an experimental perspective, *Tech. Rep. inria-00000156*, INRIA, Sophia Antipolis, France, 2005.

[20] B. Cohen, Incentive-build robustness in BitTorrent, in *Proceedings of the 1st Workshop on Economics of Peer-to-Peer Systems*, Berkeley, Calif, USA, June 2003.