

Practical Efficiencies of Slotted Stream Tapping in IP Networks

Achraf Gazdar and Abdelfettah Belghith

Ecole Nationale des Sciences de l'Informatique,
CRISTAL-RIM Laboratory, University of Manouba, 2010, TUNISIA
achraf.gazda@cristal.rnu.tn, abdefattah.belghith@ensi.rnu.tn

Abstract: Many broadcasting protocols have been proposed for video-on-demand systems. The overwhelming majority of these protocols are, however, analytically evaluated or studied through simulations but virtually with no consideration to their feasibility and or ease of implementation in a real network environment. Efficient VoD protocols are usually the most complex in terms of implementation at both the client and the server sides. In this paper, we detail the design and the implementation of our proposed VoD protocol called Slotted Stream Tapping (SST), in an IP environment using IETF standards. We compare SST results to the theoretical results of the most known hybrid protocols and show that SST presents indeed an adequate compromise between ease of implementation and server/network performance. Extensive experiments are conducted on our test bed to evaluate the server/client bandwidth consumption and the client buffer size. We finally compare SST theoretical and experimental results which we prove to be in very close resemblance.

Keywords: *Streaming Applications, Merging Protocols, Video-on-Demand.*

Received: May 1, 2006 | **Revised:** June 30, 2006 | **Accepted:** September 30, 2006

1. Introduction

Video on Demand (VoD) is a multimedia service allowing a remote user to connect and then view a video of his choice. To broadcast video, many solutions have been proposed. The simplest way is to open one stream for each client requesting the movie. This solution scales worst with the client arrival rate in terms of server and network bandwidth consumption. However, the user can perform VCR-like operations (such as, Fast Forward, Pause, etc.) with the server without any functional restrictions. Many efficient solutions have been proposed and are basically based on batching and stream-sharing among the different users requesting the same movie, also known as stream-merging. While these solutions minimize indeed the bandwidth consumption, they complicate the interactivity handling. Basically, we distinguish three VoD protocol categories namely Proactive Protocols, Reactive Protocols and Hybrid Protocols. While proactive protocol servers continuously broadcast video regardless of the number of clients, reactive protocol servers broadcast a requested video only upon the first client demand. Generally, the former are recommended to broadcast hot videos and the later are recommended to broadcast less frequently requested movies. VoD service providers are confronted with the issue of when and how to switch from one protocol category to another depending on

the requests variation as a function of time. To this end, hybrid protocols are designed to provide an easy adaptation to such user requests variation. They operate like proactive protocols at high workloads and like reactive ones at low workloads.

Several hybrid protocols are proposed in the literature [2, 3, 5, 11, 12, 16, 22]. Among these, the Slotted Stream Tapping (SST) stands out in terms of better efficiency and lower complexity [11]. Surprisingly, to our knowledge, there exists no attempt to discuss the implementation requirements and complexity of these hybrid protocols. Efforts are instead oriented toward theoretical evaluations of performance metrics such as the minimization of the server and network bandwidth consumed. In this paper, we detail the implementation of our stream-merging protocol, SST, in an IP network environment using IETF Standards. We have chosen to implement SST not only for its good performance in terms of server/network bandwidth but also and more importantly for its much lower implementation complexity with respect to other merging stream protocols. We compare SST analytical performance evaluation with experimental results provided by our test bed implementation. We also discuss our implementation decisions adopted to fulfill the SST requirements.

The paper is organized as follows: in the second section, we present the state of the art on VoD

protocols. In the third section, we detail the STT operation and functions. We discuss SST performances with respect to hybrid protocols in the fourth section. In the fifth section, we detail the implementation of SST using the IETF streaming standards. The sixth section will be devoted to implementation tests. We present some Quality of Service measurements while running SST on top of a 100MB Ethernet network. Finally, we conclude and discuss further works in this area in Section 7.

2. State of the Art on VoD Protocols

Proactive protocols are subdivided into three sub-categories. They are all based on dividing the video into segments to be streamed through several channels. The size of each segment, the bandwidths assigned to the different channels and the scheduling of the different segments among the various channels differ from one sub-category to another. In Staggered Broadcasting (SB) [8, 10, 14, 17, 27], a movie of duration D is divided into n equally sized segments. SB repeatedly broadcasts different instances of the movie on different channels (as many channels as the number of segments). Each instance of the movie is shifted in time from the previous one by just the segment duration D/n in such a way that all the movie segments are broadcast simultaneously on the various channels. In Pyramid Broadcasting (PB) [1, 8, 14, 25], each movie is divided into n ordered segments with sizes growing geometrically with parameter $\alpha > 1$. PB allows, however, multiplexing m movies through the n channels. The first channel broadcasts in turn and periodically the first segment of each of the m movies, the second channel broadcasts in the same manner the second segment of each of the m movies and so on for all the remaining channels. The client must download data simultaneously from all channels to get a continuous fluid video displaying. Data must be downloaded at a high bit rate. In fact, the client have to download data with a rate equal to m times the consumption rate, denoted hereafter by b , even if it requires just one of the m multiplexed segments within this channel. A more efficient protocol, within the same sub-category, is the Pagoda Broadcasting Protocol (PaB). PaB [8, 21] uses a more complex segment-to-channel scheduling algorithm. It distributes segments over the channels using the series $\{1, 3, 5, 12, 25, \dots\}$ as follows: the first channel broadcasts one segment, the second channel broadcasts three segments, the third channel broadcasts five segments, and so on. The segment identities to be broadcast on each channel (i.e., the segment-to-channel scheduling) are given by an elaborate scheduling algorithm [8, 21]. The client has to download data from all channels at once. PaB is better than all protocols of this sub-category in terms of the required server bandwidth while maintaining the same waiting time. Unfortunately, it uses a very complex segment-to-channel scheduling algorithm. A more efficient

algorithm was proposed in [4]. Like PaB, it does not schedule segments to channel in a sequential manner. To outperform PaB, it is based on a more elaborate heuristic to schedule segments to channels. The third sub-category is named Harmonic Broadcasting (HB) [8, 15]. It is based on the Harmonic broadcasting protocol judged as the best proactive protocol [14] in terms of required server bandwidth. HB divides each movie into equally sized segments. Each one is broadcast through a separate channel. The channel bit rates are in decreasing order: the first channel bit rate is equal to the consumption (displaying) rate b , the second channel bit rate is equal to $b/2$, the third is equal to $b/3$ and so on. The client must download data from all channels at the same time to get a smooth video display.

The most known reactive protocol is Stream Tapping, ST [7]. ST assumes that each customer Set Top Box (STB) includes a buffer capable of storing few minutes of video. This buffer allows the client to tap into streams originally created for previous clients having requested the same movie. The first client gets the requested movie on a special stream called the Complete Stream (CS). Further late clients requesting the same movie are satisfied through temporary streams called the Full Tap Streams (FTS) to download the already missed part of the movie. While displaying this first part of the movie directly from the FTS, video is also tapped from the corresponding Complete Stream and stored in the buffer. Upon finishing displaying the missed part the client terminates his FTS and he starts displaying from its buffer which continues until the end of the movie to be filled from the corresponding CS. Unfortunately, this protocol performs much worst than proactive protocols at high workload [11] as the bandwidth increases very rapidly with the number of clients demanding the same movie. A more recent reactive protocol is the Split and Merge Protocol (SAM) [20]. Upon the arrival of the first client, SAM initiates a predefined batch time interval. All clients arriving within this batch will be served by the same video stream, called the S stream, which is opened at the end of the batch time interval. In addition, SAM is an interactive protocol. It handles interactive operations through special streams called the Interactive Streams (I stream) and the use of the client (synchronized) buffer. When a client performs an interactive operation, he will be served through a new I stream until he can be merged to an ongoing S stream through his synchronized buffer.

While proactive protocols are recommended to broadcast frequently requested movies, reactive protocols should be used when the client arrival rate is low. Hybrid protocols operate like proactive protocols at high workloads and like reactive ones at low workloads. They are subdivided into three sub-categories: merge-once protocols, several-merge protocols with static client download plan and

several-merge protocols with dynamical client download plan. All hybrid protocols consider a slotted time axis where all clients arriving within one slot are handled in the same manner. Considering the first sub-category, the first client requesting a movie gets a full stream. A later user requesting the same movie shares the same stream with the first client and gets a second stream for broadcasting the missed part of the movie. Data downloaded from the full stream are stored in a local buffer until the client finishes with the missed part. Slotted Stream Tapping, SST [11] [12], Dynamic Heuristic Broadcasting Protocol, DHB [6] and Universal Distribution Protocol, UD [22], are examples of this sub-category. While SST client requires two downloading channels, DHB and UD [12] clients can use more than two downloading channels to download the entire movie. Recall that DHB is based on Fast Broadcasting [16].

Using several-merge protocols, a later client can share, in addition to the full stream, one or more "tap" streams initially opened for previous clients. Dyadic Tree [5] and Fibonacci Tree [2] are the most known protocols of this sub-category. Both protocols propose a static downloading plan to the client while using only two downloading channels. However, in the third sub-category the server can change the downloading plan of the client in order to keep the average consumed bandwidth closer to optimum. The Event-Driven Stream Merging (ERMT) [3] protocol is the only protocol in this sub-category. The main drawback of the two last sub-categories is the enormous complexity of their implementation [3] especially in an IP network. In the first sub-category the network must guaranty the quality of service needed to keep all clients synchronized to run the downloading plan without a video display interruption. Moreover, the server must keep trace of all client states to recalculate the "best" downloading plan for all clients upon each new arrival.

The majority of these protocols have been evaluated analytically without considering the practical side and the feasibility of their implementations on a given network architecture. Some papers, however, have discussed design and practical aspects of VoD systems. In [24], authors implement a VoD server combining two broadcasting schemes namely Skyscraper broadcasting [13] (a variant of PB protocol) and HB protocol. Another VoD system design has been proposed in [19]. This VoD system combines a reactive protocol and a proactive protocol for video broadcasting. The same issue has been discussed in [28] and [23]. In [28] authors focused on VoD server components and their interaction using a unicast broadcast scheme. However in [23], authors implement an SB-like broadcasting server. These efforts focused on server performance evaluations in terms of disk bandwidth, CPU load and memory consumption. In this paper, we rather focus on the SST bandwidth consumed at both server and client sides and the SST client buffer size.

In the next section, we will present in detail all these aspects. We start by detailing the SST principle, then positioning SST with respect to other stream-merging protocols, detailing the SST implementation and comparing practical results to analytical results. We will finish by giving some experimental results collected by running the SST protocol on different network infrastructures.

3. Slotted Stream Tapping

SST is based on the same streaming technique as ST. However, SST uses a slotted time axis. Any client arriving within a slot will be served at the beginning of the next slot: new requests are only served at the beginning of the next slot. All slots are of the same size which represents the maximum client waiting time. Initially ST was proposed with no client waiting time. This is very hard to be dealt with in practice, especially in IP networks where the client needs to compensate for the network jitter by means of buffering. The buffering time can reach up few minutes. Hence the waiting time introduced by SST protocol is inherent to any practical implementation and therefore will not burden the system performance.

As showed in Figure 1, the first client requesting the movie will wait for the start of the next slot to be served. It (and all arrivals in this same slot) will be assigned a Complete Stream. Later requests for the same movie arriving in the subsequent slots will wait to be served at the beginning of the next coming slot. All clients arriving in the same time slot are served through the same video stream. They are assigned a Full Tap Stream, FTS, to download the missed part of the movie while at the same time they tap into the CS that was initially created for the first client(s). While displaying the missed video, they save data from the

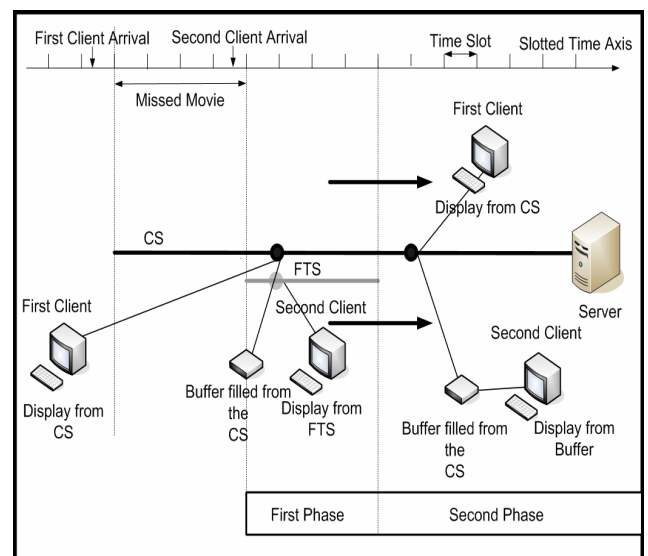


Figure 1. SST Streaming Scheme.

CS in their buffers (phase 1 in Figure 1). Clients leave the FTS and continue displaying the video through the buffer (phase 2 in Figure 1). SST

assumes that the buffer size is large enough to store any missed video data.

At a given stage, the server has to decide whether to continue the same process of opening FTS streams or to open an additional (new) CS for the same movie in order to minimize the server/network bandwidth. In [11], we determined the value of the client delay after which the server decides to open a new CS instead of opening an FTS. This delay value denoted xD is given by the following equation

$$xD = \sqrt{2sD} \quad (1)$$

where s and D are respectively the slot time duration and the movie duration.

The average consumed bandwidth is given by the following equation [11].

$$B = \sqrt{\frac{2D}{s}} - 0.5 \quad (2)$$

These results are obtained when the arrival rate goes to infinity. In all cases, the client doesn't need to download data from more than two channels. This makes SST very easy to implement in both connection-oriented and connectionless networks.

In the next section, we will compare the performance of SST to stream-merge protocols in terms of server/network bandwidth.

4. Comparing SST to Stream-Merging Protocols

We consider DHB and UD as representatives of “merge-once” protocols, Dyadic and the optimal protocol threshold as representatives of “several-merge” protocols. We choose Dyadic because it is close to ERMT and Fibonacci in terms of performance [3].

4.1 Comparing SST to UD and DHB

We consider a movie of two hours and a slot duration of 72 seconds (this is indeed the values used for UD in [22]). Figure 2 below portrays the average consumed bandwidth as a function of the total client arrivals during the movie for SST, UD and DHB.

As shown in this figure, DHB and UD consumed bandwidth attains its limiting value very rapidly, while that for SST increases more slowly. It is clear in the figure that SST performs much better than UD and DHB for low workloads. In fact, SST needs less bandwidth than DHB for an arrival rate less than 34 arrivals in two hours and less bandwidth than UD for an arrival rate less than 70 arrivals in two hours. We call this phase the reactive phase. For arrival rate values greater than 70, which lie within the proactive phase, the additional bandwidth needed by SST does not exceed 3b and 6b compared respectively to UD and DHB. We think, however, that this is a reasonable

price to pay to guaranty that the client downloading channel number does not exceed two channels, yet the SST broadcasting scheme is very much simpler.

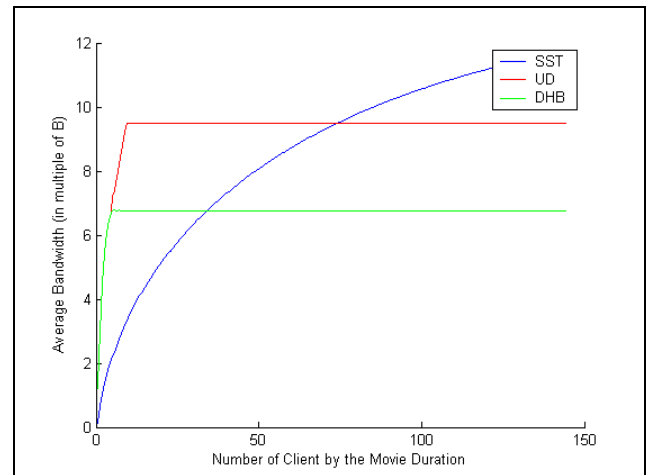


Figure 2. Comparing SST to DHB and UD for two hours movie duration and 76 sec time slot.

4.2 Comparing SST to Dyadic and the optimal limit

Figure 3 is obtained for two hours movie duration and a client arrival rate equals to 0.1 per seconds (proactive phase 720 client arrival per movie duration). As we can see in this figure, the average bandwidth needed by SST becomes very close to Dyadic and Optimal limits when the slot duration is greater than 300 sec.

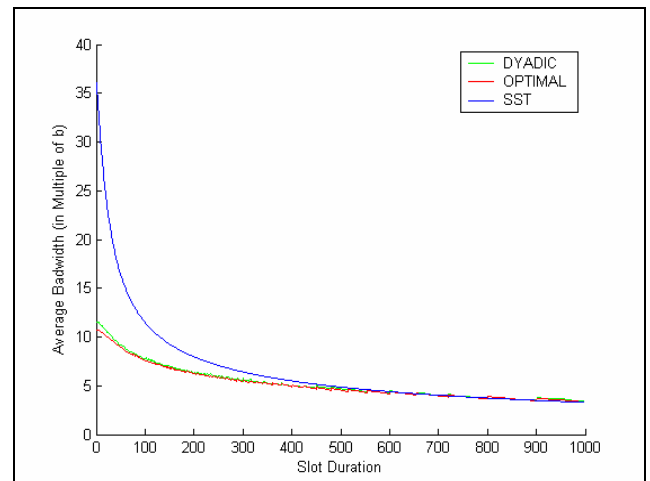


Figure 3. Comparing SST to Dyadic and the Optimal limits for two hours movie duration and a client arrival rate equal to 0.1.

Furthermore, Figure 4 below shows that SST requires less than 1.5 times the optimal bandwidth needed. Forcing the DHB client to use no more than two downloading channels makes the bandwidth consumed by DHB 3.8 times greater than the optimal bandwidth. We are here considering a slot duration equal to 2% of the entire movie duration and for a 0.1 client arrival rate value.

All previous results show that SST stands very well among all stream merging protocols in terms of bandwidth consumption, number of client downloading channels and implementation requirements.

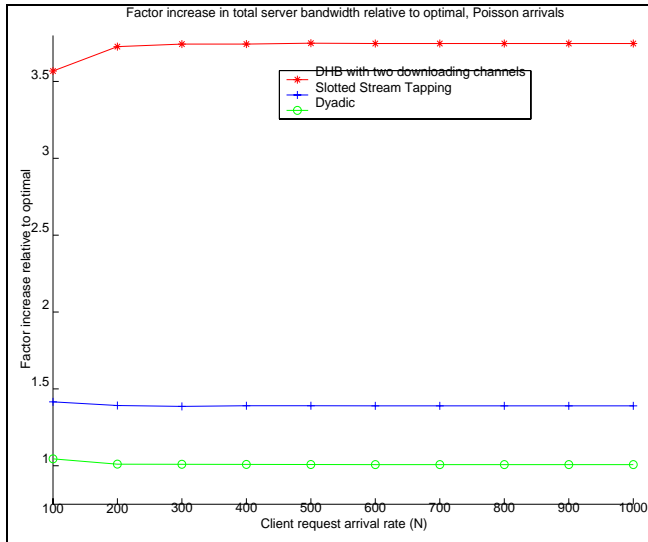


Figure 4. Increasing bandwidth factor for SST and Dyadic relatively to the Optimal limits for 0.1 arrival rate and 2% slot duration of the whole movie duration.

5. Implementing SST in an IP Network

In order to implement SST, we have reused the open source VideoLan Client media player, VLC [26], for the client side and the open source Live555.com library [18] for the server side. This library implements the RTP/RTCP and RTSP/SDP IETF standard protocols for streaming applications. Both VLC and Live555.com can be compiled for Windows and UNIX based operating systems. To meet SST requirements, we have added new modules in both VLC and Live555.com sides. All our implementations are done on a Pentium 4 machine with 1 Go of memory, 3 GHz processor and 100 Go hard disk. So far, our SST implementation only supports the MPEG2 Transport Stream video format. For experimental tests, we have installed the client in 8 machines (Pentium 3) with 512 Mo memory, 733 MHz processor and 13 Go hard disk. Our experiments have focused on the bandwidth consumption for the server side and both the consumed bandwidth and the buffer size for the client side. The implementation design is given in Figure 5. Gray parts on the figure illustrate our added modules. The default reused parts are presented in dotted boxes.

5.1 Server Side

Live555.com allows the client to get on demand streams using only unicast connections. Hence, we have added a new module to the library allowing multicast on-demand streams (Multicast on Demand Media Server on Figure 5). Through this new feature, the server becomes able to open CS streams when needed. For FTS streams, we continue to use the unicast default library implementation to open an FTS

for later clients (the unicast on Demand Media Server on Figure 5). All streams are established through RTP sessions. We have also added a new feature to the RTSP server to calculate the delay value at each new arrival (see the xD control module on Figure 5), compare it with the optimal xD value and then execute the right decision (as explained above in section 3). The server will either open an FTS stream while sharing the last CS with previous clients or just open a new CS for this client. In our implementation, we have decided to bring all elaborate features within the client side to make the server as scalable as possible. In fact, the client is made responsible to know whether it needs a new CS or just an FTS. Those decisions are based on the server parameter values sent in the SDP file upon the client connection. The whole connection/playing process in the server side are the following (see Figure 5):

- S1-** RTSP DESCRIBE commands reception on both streams (FTS+CS)
- S2-** Responding the client by two SDP descriptions on both streams
- S3-** RTSP PLAY commands reception on one or two streams depending on the client delay
- S4-** Streaming of streams on which the server has received a PLAY command
- S5-** Closing the FTS stream when receiving an RTSP TEARDOWN for it
- S6-** Ignoring all TEARDOWN commands on a given CS stream until the number of TEARDOWNS on this stream becomes equal to the all the clients sharing it.

To allow a client to know whether it needs an FTS (its delay < xD), we have introduced a new parameter in the SDP file to be returned to the client. This parameter is named 'a=x-firstSeqNum' (in the SDP standard the letter 'a' indicates a further information on the session). When the client needs an FTS this parameter value is set to 0 ('a=x-firstSeqNum:0'), else this value is set to the first sequence number of the last CS stream. The next section explains how a client knows if it needs opening an FTS or a CS. Before starting the streaming process, the server administrator must indicate, through the server GUI, the multicast IP address to use for CS streams and the RTP port number of the first CS. This port will be incremented by 2 for each new CS.

5.2 Client Side

In the client side, we have introduced a new GUI component to the VLC media player to allow the user to use the SST protocol. Figure 6 below illustrates the SST VLC GUI support. Through the SST text field, the user can indicate the SST URL to get the requested movie.

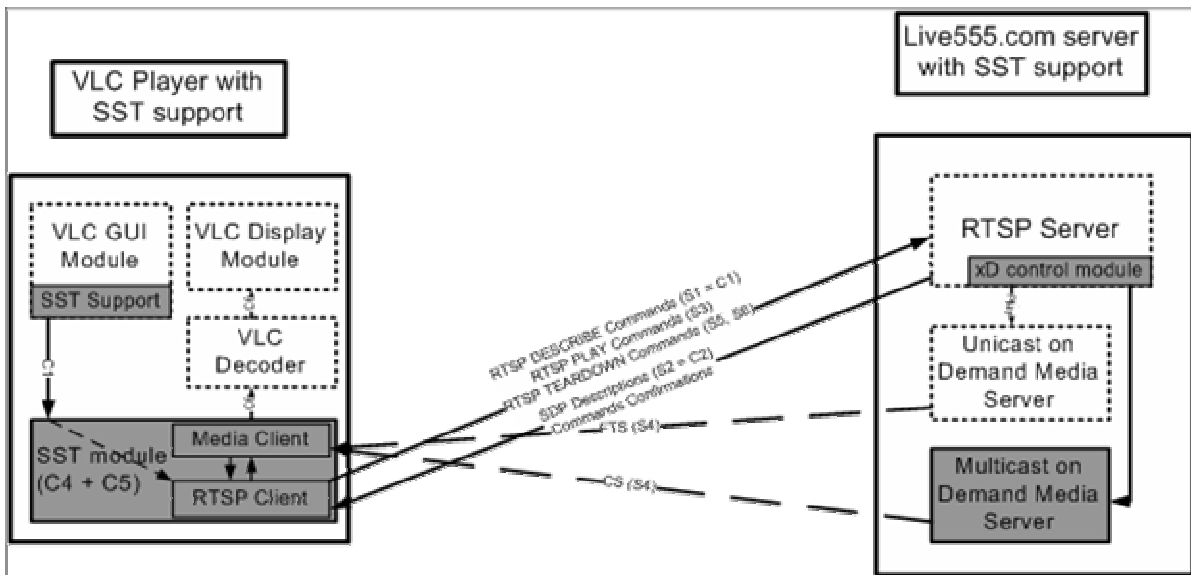


Figure 5. SST implementation design.

In the VLC code, we integrated a new module called 'sstmodule'. This module handles the SST URL as follows (see Figure 5):

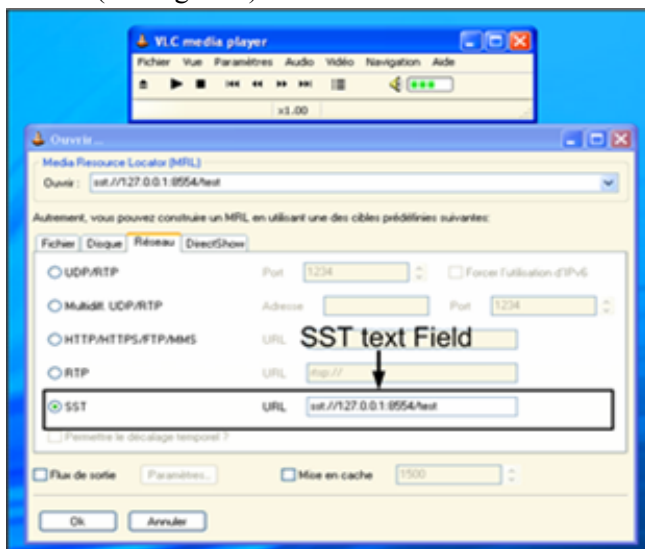


Figure 6. VLC GUI with SST support.

C1- Decomposing the SST URL introduced in the VLC GUI module into two RTSP URLs. For example, if the user indicates the SST URL:

'sst://server:portnumber/starwars',

sstmodule will generate these RTSP URLs :

'rtsp://server:portnumber/starwars' and
'rtsp://server:portnumber/starwarsMissed'

Then, the two URLs are sent to the server using RTSP DESCRIBE commands (step S1 in Figure 5). The first URL concerns the CS and the second one concerns the FTS.

C2- Receiving two SDP descriptions (one description for the CS RTP session and one description for the FTS RTP session). Figure 7 below

shows an example of descriptions received by the client.

C3- Parsing the SDP descriptions:

If the client receives 'x-firstSeqNum:0', he knows that only a CS and consequently he sends an RTSP PLAY command for this CS RTP session otherwise he sends two PLAY commands, one for each stream. He then begins downloading data from the two RTP sessions.

C4- Get the current RTP sequence number from the CS RTP session. The number of missed packets is given by $\text{currentCS RTPSeqNum} - \text{firstCS RTPSeqNum}$

C5- Filling a circular buffer with video frames downloaded from the CS stream while displaying data from the FTS stream. When the number of downloaded FTS RTP frames reaches the missed packet number, the client sends an RTSP TEARDOWN command to the server to terminate this FTS.

C6- Displaying data from the buffer while the client keeps filling it from the CS until the end of the movie

Experimental results in terms of server bandwidth variation, client bandwidth variation and client buffer size variation are given in the next section.

6. Experimental Results

In this section, we present experimental results for both the client and the server side. These results are obtained for parameter values given in table 1 below. All statistical values are collected every 2 seconds.

6.1 Server Bandwidth

In Figure 8, we plot the server bandwidth variation together with the analytical average bandwidth given by equation 2.

As we can see in this figure, the practical server bandwidth variation is close to the analytical average bandwidth. The small difference is due to the system starting period. This result shows that our analytical model gives a good estimation of an SST real VoD system at the server side.

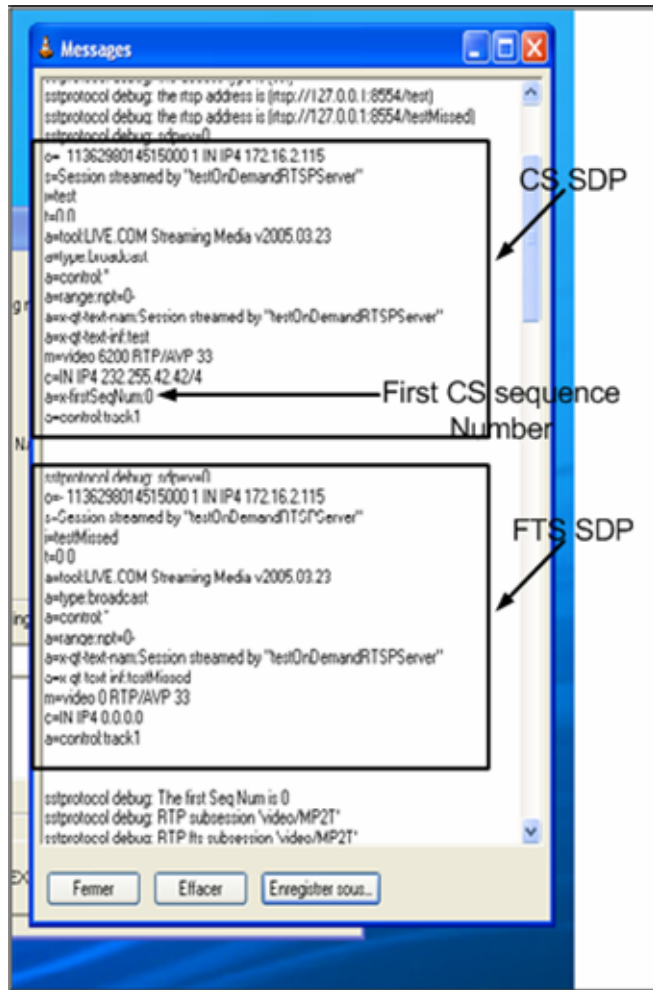


Figure 7. SDP descriptions received by VLC player.

Parameter	Value
Video File Format	MPEG2 Transport Stream
Video Duration (D)	≈ 13 mn
Consumption Rate (b)	≈ 110 kbps
xD	216 sec
Time Slot Duration	30 sec
Arrival Rate	1
Network Type	100 Mbps LAN

Table 1. Experimental parameter values

6.2 Client Bandwidth

Figure 9 portrays the client downloading bandwidth as a function of the playing time. We have considered two late client arrivals, the first with 30 seconds delay and the second with 60 seconds delay. Both clients start the video download process using about 220 kbps, which is slightly equal to two times the consumption rate. Then, this bandwidth decreases for both clients to reach 110 Kbps after about 30 seconds and 60 seconds respectively. Indeed, the server has closed the FTS streams for both clients after receiving the TEARDOWN commands.

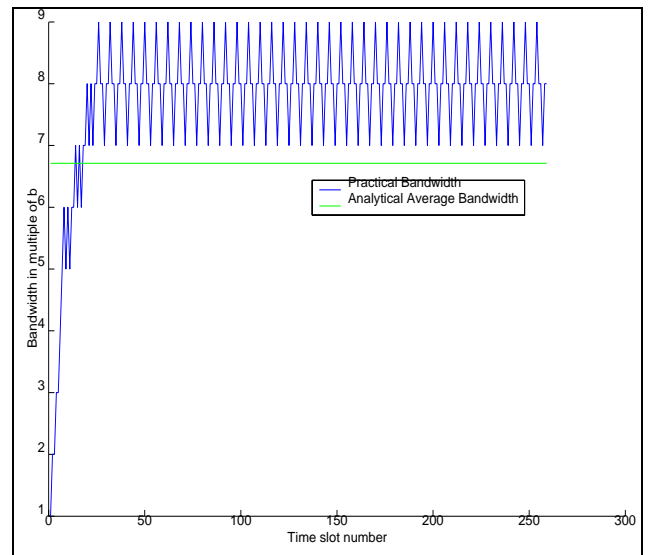


Figure 8. SST server bandwidth compared to the analytical average bandwidth.

6.3 Client Buffer

As explained in section 5, we have implemented a circular buffer to store video frames coming from the CS while playing missed frames from the FTS. After closing the FTS, the buffer continues to be filled from the head. The reader and the writer processes continue their job in a circular manner from head to tail. Hence we keep the same buffer size for the remaining part of the movie. In all cases, the maximum client buffer size doesn't exceed xD seconds of video which can vary from one video format to another.

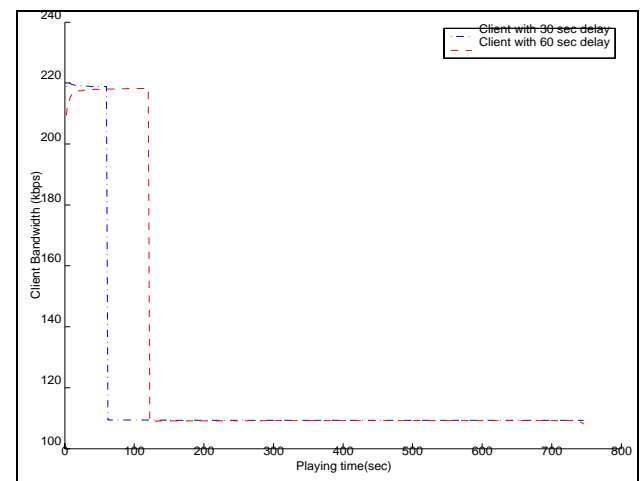


Figure 9. Client downloading bandwidth.

In our case, a later client with a delay xD equal to 216 seconds MPEG2 TS video format doesn't need more than 23.39 Mo. This size includes both video frames and extra control information such as the frame size indicator, the playing time value, etc. These results are plotted in Figure 10 which portrays the buffer size as a function of the playing time for the two late clients having 30 seconds and 60 seconds delay respectively. The figure clearly shows that the buffer size becomes constant after closing the FTS

streams. The maximum buffer size corresponding to an xD delay time is also presented in this figure

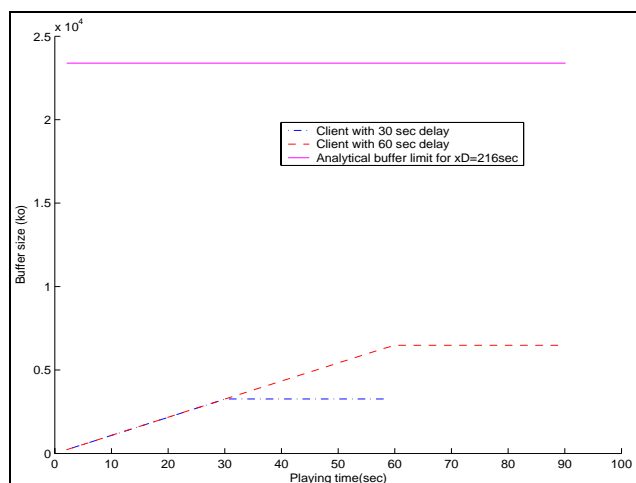


Figure 10. Client buffer size.

7. Conclusion

In this paper, we have described our SST protocol implementation in an IP environment. SST presents an adequate compromise between simplicity of implementation and performance especially in terms of consumed bandwidth with respect to other stream-merging protocols. We have also shown that experimental results are close to the analytical results found in [11]. The server scalability is an important issue in VoD systems. To this end, we integrated complex tasks such as late delay calculation and CS/FTS stream management within the client side. We have used IETF standards RTP/RTCP for streaming and RTSP/SDP for signaling.

Presently, we are investigating different ways to implement interactive VCR-like operations on top of the SST protocol. We are also studying different QoS techniques to adapt these stream-merging protocols to be used in Wireless Local Area networks.

References

- [1] C. C. Aggarwal, J. L. Wolf, and P. S. Yu, A permutation-based pyramid broadcasting scheme for video-on-demand systems, in *Proceedings of the IEEE International Conference on Multimedia and Computing Systems*, June 1996, pp. 118-126.
- [2] A. Bar-Noy, and R. E. Ladner, Competitive on-line Stream Merging Algorithm for Media-on-Demand, in *Proceedings of the Twelfth Annual ACM-SIAM Symposium on Discrete Algorithms*, 2001, pp.364-373.
- [3] A. Bar-Noy, J. Goshi, R. E. Ladner, and K. Tam, Comparison of Stream Merging Algorithms for Media-on-Demand, *Multimedia Systems*, vol. 9, no. 5, March 2004.
- [4] Amotz Bar-Noy, and Richard E. Ladner, Windows Scheduling Problems for Broadcast Systems, in *Proceedings of the 13th ACM-SIAM Symposium on Discrete Algorithms, SODA*, March 2002.
- [5] E. G. Coffman, Jr., Predrag Jelenković, and Petar Momčilović, The Dyadic Stream Merging Algorithm, *Journal of Algorithms*, vol. 43, pp. 120-137, 2002.
- [6] S. R. Carter, J.-F. Pâris, S. Mohan, and D. D. E. Long, A Dynamic Heuristic Broadcasting Protocol for Video-on-Demand, in *Proceedings of the International Conference on Distributed Computing Systems (ICDCS 2001)*, Mesa, AZ, April 2001, pp. 657-664.
- [7] S. W. Carter, and D. D. D. Long, Improving Video-on-Demand Server Efficiency through Stream Tapping, In *Proceedings Of the 6th International Conference on Computer Communications and Networks (ICCCN'97)*, Las Vegas, NV, USA, September 1997, pp. 200-207.
- [8] S. W. Carter, D. D. E. Long, and J.-F. Pâris, Video-on-Demand Broadcasting Protocols, in *Proceedings of Multimedia Communication: Directions and Innovations*, 2000, pp. 179-189.
- [9] D. Eager, M. Vernon, and J. Zahorjan, Minimizing Bandwidth Requirements for On-Demand Data Delivery, in *Proceedings Of the 5th International Workshop On Multimedia Information Systems (MIS' 99)*, Indian Wells, CA, October 1999.
- [10] A. Gazdar, and A. Belghith, Discrete Interactive Staggered Broadcasting, in *Proceedings of the first IEEE Consumer Communications and Networking Conference (CCNC'04)*, Las Vegas, Nevada USA, January 2004.
- [11] A. Gazdar, and A. Belghith, Slotted Stream Tapping," in *Proceedings of the ACM Multimedia/NRBC'04 workshop*, New York, USA, October 2004.
- [12] A. Gazdar, and A. Belghith, Hybrid Broadcasting Protocols: a Comparative Study, in *Proceedings of the 4th IEEE International Symposium on Signal Processing and Information Technology, ISSPIT'04*, Rome, Italy, December 2004.
- [13] K. A. Hua, and S. Shen, Skyscraper Broadcasting: a new Broadcasting Scheme for

- Metropolitan Video-on-Demand Systems, in *Proceedings of the SIGCOMM 97*, Cannes, France, September 1997, pp. 89-100.
- [14] A. Hu, Video-on-Demand Broadcasting Protocols: A Comprehensive Study, in *Proceedings of Infocom'01*, Anchorage, AK, April 2001.
- [15] L-S. Juhn, and L-M. Tseng, Harmonic Broadcasting for Video-on-Demand Service, *IEEE Transactions on Broadcasting*, vol. 24, no. 17, pp. 66-77, 1997.
- [16] L-S. Juhn, and L-M. Tseng, Fast Broadcasting for Hot Video Access," in *Proceedings of the International Workshop on real-Time Computing Systems and Application*, Taipei, Taiwan, October 1997, pp. 237-243.
- [17] F. B. Kwon, and H. Y. Yeom, Providing VCR Functionality in Staggered Video Broadcasting, in *Proceedings of International Conference on Protocol of Multimedia Systems (PROMS 2001)*, Enschede, the Netherlands, October 2001.
- [18] Live555.com, <http://www.live555.com>.
- [19] J. Y. B. Lee, and C. H. Lee, Design, Performance Analysis, and Implementation of a Super-Scalar Video-on-Demand System, *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 12, no. 11, November 2002.
- [20] W. Liao, and V. O. K. Li, The Split and Merge Protocol for Interactive Video-on-Demand systems, *IEEE Multimedia*, vol. 4, No. 4, pages 51-62, 1997.
- [21] J-F. Pâris, S. W. Carter, and D. E. Long, A hybrid broadcasting protocol for video-on-demand, in *Proceedings of the 1999 Multimedia Computing and Networking Conference*, San Jose, CA, January 1999, pp. 317-326.
- [22] J-F. Pâris, S. W. Carter, and D. D. E. Long, A Universal Distribution Protocol for Video-On-Demand, in *Proceedings IEEE International Conference on Multimedia and Expo 2000*, New York, NY, July-August 2000, vol. 1, pp. 49-52.
- [23] A. Papagiannis, S. Egglezos, and D. Lioupis, A scalable, low-cost VoD server with multicast support, in *Proceedings of the 2nd International Conference on Information Research and Education, ITRE 2004*, 2004.
- [24] M. Tran, W. Putthividhya, W. Tavanapong, and J. Wong, A Case for a Generalized Periodic Broadcast Server: Design, Analysis, and Implementation, in *Proceedings of the International Symposium on Applications and the Internet (SAINT'04)*, 2004.
- [25] S. Viswanathan, and T. Imielinski, Pyramid Broadcasting for Video-on-Demand Service, *The International Society for Optical Engineering SPIE*, vol. 43, no. 3, Sep. 1997.
- [26] VideoLan Client, <http://www.videolan.org>.
- [27] E. M. Yan, and T. Kameda, An Efficient VOD Broadcasting Scheme with User Bandwidth Limit, in *Proceedings of the SPIE/ACM MMCN 2003*, Santa Carla, CA, January 2003.
- [28] S. Yang, H. Yang, and Y. Yang, Architecture of High Capacity VOD Server and the Implementation of its Prototype, *IEEE Transactions on Consumer Electronics*, vol. 49, no. 4, Nov 2003.

Achraf Gazdar



received his Master degree in Computer Sciences from the National School of Computer Sciences (ENSI), University of Mannouba, Tunisia in 2002. He is currently a Ph.D student at the same school working on VoD broadcasting interactive protocols. His research interest

VoD and multimedia systems, networking and QoS management of multimedia streaming over wireless networks

Dr. Abdelfettah Belghith



received his Master of Science and his PhD degrees in computer Science from the University of California at Los Angeles (UCLA) respectively in 1982 and 1987. He is since 1992 a full Professor at the National School of Computer Science (ENSI), University of Mannouba, Tunisia. His research interests

include computer networks, wireless networks, multimedia Internet, mobile computing, distributed algorithms, simulation and performance evaluation. He runs several projects in cooperation with other universities and research institutions. He is currently the responsible of the graduate studies department and the head of the RIM research group at ENSI.