This full text paper was peer reviewed at the direction of IEEE Communications Society subject matter experts for publication in the ICC 2007 proceedings.

CLAPS: A Cross-Layer Analysis Platform for P2P video Streaming

M. Barbera, A. G. Busà, A.Lombardo, G.Schembra
Dipartimento di Ingegneria Informatica e delle Telecomunicazioni
University of Catania
Catania, Italy
{mbarbera, abusa, lombardo, schembra}@diit.unict.it

***Abstract*— Peer-to-Peer (P2P) networks are emerging as an interesting alternative to media streaming delivery with respect to Content Delivery Networks (CDN). Although several P2P protocols have been proposed to support video streaming with an adequate degree of Quality of Service (QoS), a critical point today is how to evaluate them and optimize their parameters. The difficulty lies in the fact that the behavior of the P2P overlay network and the one of the underlying packet network are closely related and need to be analyzed simultaneously. Ad-hoc simulators for each P2P architecture have been proposed but they neglect the interactions between peers and the underlying network. This paper proposes a new approach to P2P video streaming performance analysis and system design which provides a very accurate cross-layer simulation of the real behavior of P2P clients over TCP/IP networks. A fluid-flow approach has been adopted to simulate the behavior of the network elements supporting an emulated overlay network consisting of real implementations of P2P clients. The P2P SplitStream video streaming protocol is considered as a case study to demonstrate that the tool is able to capture performance parameters at both the overlay (e.g. inter description synchronization) and packet network levels (e.g. traffic source round-trip time and router queue behavior).**

***Index Terms*— Peer-to-Peer, Simulation tool, Video Streaming, SplitStream, Multiple Description Video coding.**

I. INTRODUCTION

Media streaming over the Internet has recently gained considerable popularity due to the continuous increase in network access speed at the end-user site. Content Delivery Networks (CDN) have been proposed to distribute media streaming over the Internet. However, a CDN is based on the traditional client-server model which greatly limits the performance achievable:

- even large streaming servers are not able to feed more than a few hundred streaming sessions simultaneously;
- the streaming server is the single point of failure;
- the high network access rate is expensive;
- it is difficult to select the best streaming server in a CDN during a session.

This work has been partially supported by the Italian PRIN project FAMOUS under grant number 2005093971 and the EU FP6 IST-NEWCOM project.

Instead Peer-to-Peer (P2P) networks provide a cheaper alternative to media streaming which overcomes the problems mentioned above [19]. The P2P approach is interesting because the bandwidth available to distribute the content scales with the demand (i.e. the number of interested clients). On the other hand, media distribution has rigorous Quality of Service (QoS) requirements which are very challenging in an environment in which the transmission capacity changes dynamically and the P2P resource availability is not stable.

Many researchers are currently investigating how to implement media streaming in P2P networks in order to provide an efficient, scalable multimedia distribution service. More specifically, current research is addressing both the design of new P2P protocol mechanisms [9,11,12], and the definition and application of new scalable, robust multiple description video coding frameworks [10-16]. Despite the great effort devoted to the research areas mentioned above, a critical point today is how to evaluate them and optimize their parameters. The difficulty lies in the fact that the behavior of the P2P overlay network and the one of the underlying packet network are closely related and need to be analyzed simultaneously. *Ad-hoc* simulators for each proposed P2P architecture have been implemented in previous literature [6-14], but they are severely limited when P2P media streaming is involved. Traditional P2P simulators, in fact, focus on the interactions among peers, neglecting the behavior of the routers in the underlying network. This is because traditional P2P simulators very often model the underlying network as a constant-delay link between each pair of peers. This approximation makes it possible to analyze P2P overlay networks with a great number of peers because the computational cost of IP network simulation is negligible, but at the same time it does not provide a real estimation of the P2P overlay network performance.

Moreover, an effective simulation-based analysis of a media streaming P2P architecture has to consider the effects of IP network phenomena (network congestion, buffer queue delay, packet loss, transport layer dynamics, and so on) on P2P traffic because these phenomena influence the behavior of P2P clients themselves.

This paper proposes a Cross-Layer Analysis Platform for P2P video Streaming (CLAPS) which provides a very accurate

1-4244-0353-7/07/$25.00 ©2007 IEEE

cross-layer simulation of the actual behavior of P2P clients over TCP/IP networks. More specifically, an integrated simulation environment has been defined in which a fluid-flow simulator is used to evaluate the behavior of network elements [2] when loaded by the output traffic from ongoing real implementations of P2P clients. Fluid models have recently been applied as a scalable and accurate methodology to analyze the performance of IP networks [3-13]. In this paper the Network Analysis and Parameter Tuning Tool (NAPT) [21] has been modified and adapted to the proposed simulation platform. It is able to manage a large amount of traffic flows with a significant reduction in processing time as compared with traditional packet-by-packet simulators. As far as application layer simulation is concerned, although a number of P2P simulators have been presented in past literature, all of them are difficult to validate against real, non-trivial, P2P overlay networks. For this reason, NAPT has been linked to a real implementation of a SplitStream [23] P2P client to generate the data traffic loading network nodes; however, the use of the SplitStream client can be replaced with any P2P client implementation, so the proposed approach is general and can easily be extended to any P2P protocol. Working in this way, the maximum level of accuracy at the application level has been achieved.

The rest of the paper is organized as follows. Section II describes the architecture of the proposed cross-layer simulator. Section III presents a brief overview of SplitStream, while Section IV shows how the cross-layer approach can be used to analyze the performance of a media streaming P2P environment. More specifically, inter-frame arrival times are evaluated at the overlay network level, while buffer queue delay and round-trip time are calculated at the underlying packet network level, in order to analyze the whole P2P video streaming system when the SplitStream overlay network is used to support multiple description video streaming. Finally, Section V concludes the paper.

II. CLAPS: THE PROPOSED APPROACH

This section describes the component blocks of the proposed cross-layer platform for P2P applications. It can basically be divided into three parts (see Fig. 1):

- the simulator of the IP network, based on fluid-flow models;
- the P2P overlay network emulator;
- an interface between the above parts.

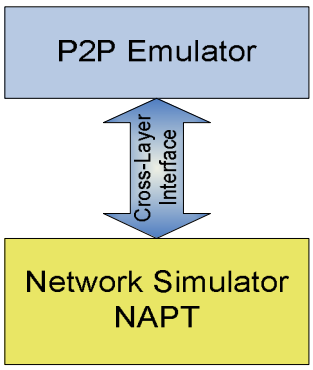

Figure 1. CLAPS Architecture

P2P data exchanges are communicated by the P2P emulator to the network simulator. The first block simulates the transmission of data and sends a notice to the P2P emulator when data are received by the destination client. The following sections illustrate this in more detail.

A. NAPT: the network-layer simulator

The need to simulate both the P2P application layer and its interactions with the IP network cannot be achieved by using a traditional packet-by-packet network simulator because it would require an excessive amount of processing resources when the number of peers in the overlay network increases.

For this reason, a network simulator, called NAPT (Network Analysis and Parameter Tuning) [2], has been developed. It provides fast, low-memory simulations by using a fluid-flow model. Fluid models describe network behavior through a set of ordinary differential equations which provide an abstract deterministic description of the average network dynamics. The fluid-flow approach neglects the detailed short-term packet-by-packet behavior which prevents the scaling of traditional approaches to the study of large IP networks. The differential equations are numerically solved to obtain estimates of the time-dependent network behavior [3].

The set of differential equations describing the behavior of each source or network element is independent of the network topology. The set of differential equations for the whole network can therefore be easily obtained by considering the equations related to each source and each network element separately. In other words, the system model as a whole is a combination of building components in which the output data resulting from the solution of the differential equations describing a given component (e.g. a TCP source) are the input for the differential equations describing the next component in the system being studied (e.g. a router).

In order to simulate heterogeneous networks, a number of components have been implemented inside NAPT, such as Drop Tail, RED and RIO Buffers, Wireless Links, TCP and UDP sources and so on. There is also the possibility of creating reusable structured components such as routers, source or destination aggregates, to design large networks. All the basic components are implemented in separate modules.

In order to make NAPT easily expandable, the differential equations of the fluid-flow model inside each generic module are written using a specific syntax similar to C++ language. This syntax is subsequently parsed to generate C++ code. Moreover, NAPT has been provided with both a user-friendly graphic interface to design topologies with simple drag-and-drop commands, and the API functionalities needed for the cross-layer interface towards the P2P emulator. The graphic interface is in Java code to make NAPT portable to different operating systems. The most important elements of the NAPT architecture are shown in Fig. 2. This figure also shows how NAPT is linked to a P2P client implementation through the Cross-Layer Interface. In order to demonstrate how fast and accurate NAPT is in both transient and steady-state analysis, two network analysis case studies have been considered.

The network topology in the first case study is made up of four Drop Tail routers, and data traffic is generated by two

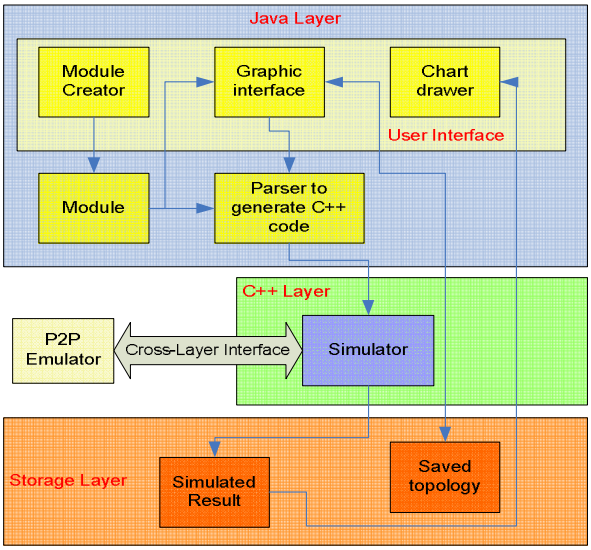

Figure 2. NAPT Architecture.

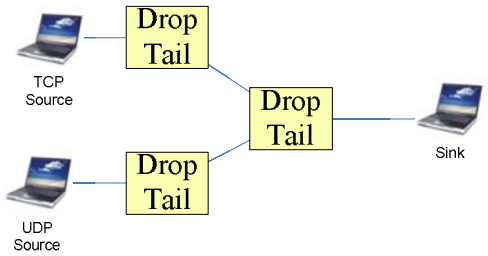

Figure 3. Topology of the first case study.

TABLE I. AVERAGE THROUGHPUT AND GOODPUT IN NAPT AND NS2

	NAPT average value (Kb/sec)	NS2 average value (Kb/sec)
UDP Throughput	128	122.07
TCP Throughput	94.51	94.28
UDP Goodput	125.08	116.7
TCP Goodput	89.85	90.10

TABLE II. EXECUTION TIME IN NAPT AND NS2

Simulation time	NAPT with trace	NS2 with trace	NAPT without trace	NS2 without trace
10 sec.	5 sec.	5 min.	0,36 sec.	2 sec.
20 sec.	10 sec.	11 min.	1,1 sec.	3 sec.
60 sec.	18 sec.	23 min.	3.06 sec	6 sec.

sources: a TCP greedy source and a UDP source with an average bit rate of 128 Kb/s (see Fig. 3). Fig. 4 shows the average temporal behavior of the TCP goodput (defined as the throughput without considering lost packets) calculated by both NAPT and the NS2 simulator [20]. Table I lists the average throughput and goodput values for TCP and UDP sources from both simulators. These results demonstrate the accuracy of NAPT in predicting not only the expected values of the considered metrics, but also their temporal evolutions.

The second case study proposed aims to evaluate NAPT performance in terms of the processing time required. For this reason we considered a more complex network (see Fig. 5) with 5 Drop Tail routers and one RED router, loaded by 66 TCP greedy sources. Table 2 shows the execution time needed to simulate the above network when both NAPT and NS2 simulators are used. Since a significant amount of processing time is due to saving trace files on the hard disk, the results are obtained with or without saving the trace file. As can be observed, NAPT is over 50% faster than NS2. Furthermore, NAPT is not only better in execution time but also in memory occupation. Indeed, in this simple case NS2 allocates 5 MB of RAM as compared with 2.5 MB allocated by NAPT. Of course, this difference is more evident in larger network simulations.

B. Cross-Layer Interface

The Cross-layer interface (see [21] for more details) defines the rules governing communication between NAPT and any P2P emulator. More specifically, it manages the exchange of

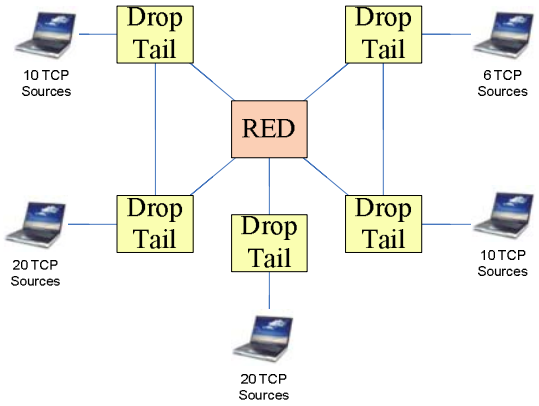

Figure 5. Topology of the second case study.

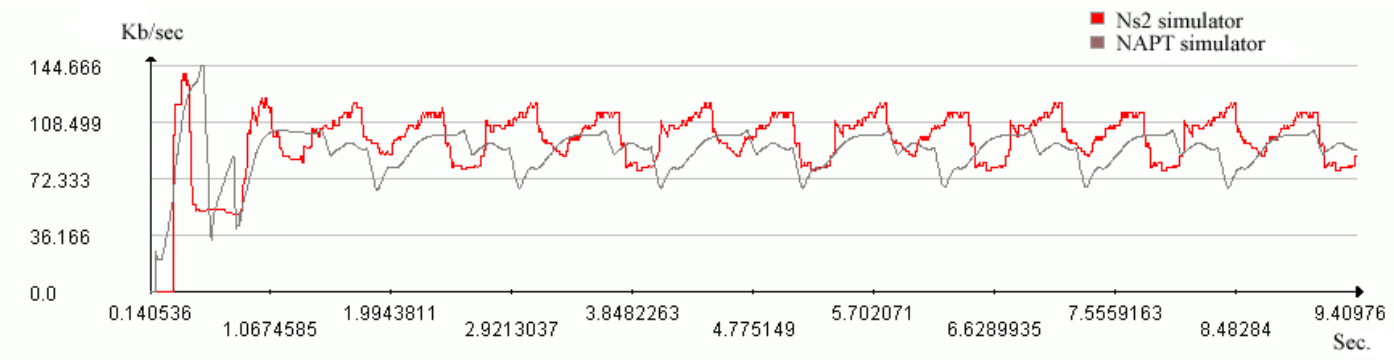

Figure 4. TCP Goodput comparison between NAPT and NS2

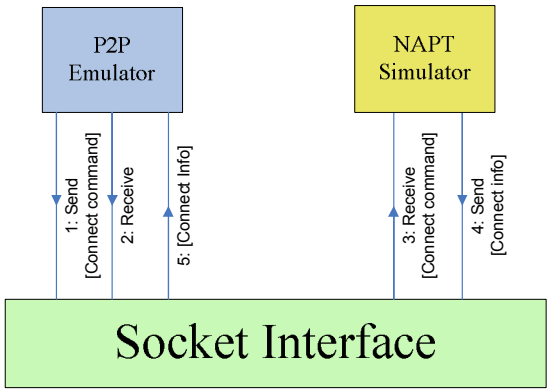

Figure 6. Communication scheme.

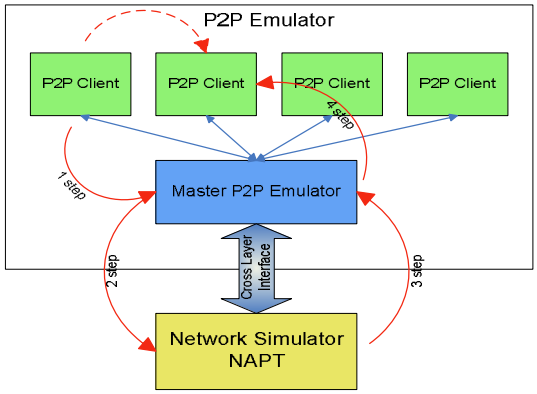

Figure 7. P2P Emulator scheme.

formatted strings through socket communications; by so doing, CLAPS is independent of both the programming language and the operating system used to develop the P2P emulator. Thanks to this interface, the P2P layer emulator can drive the network simulator to set up or shut down TCP connections among clients, and also send or receive data packets.

An example of cross-layer communication is connection establishment. The P2P emulator sends a "Connect" command through the socket, composed of the CONNECT keyword, the connection time, the connection request identifier, the P2P client source, the P2P client destination, and the connection type (TCP or UDP). The network simulator replies with a "Connect Info" command composed of the keyword CONNECT_INFO, the connection request identifier and the connection identifier. The cross-layer communication also performs synchronization between NAPT and the P2P emulator. For this purpose, the interface uses the blocking socket as in the usual socket interface. Thanks to this interface, any cross-layer simulator can be realized, simply changing the P2P emulator according to the P2P protocol to be used. Fig. 6 shows the connection establishment example described above.

C. P2P emulator at the Application Layer

Nowadays a number of P2P simulators are affected by approximation problems because not all features are implemented in a simulator. A different approach to overcome this problem is emulation. We start from real P2P video clients and modify them to realize a P2P emulator. In this way we do not make any kind of approximations because client applications are really working and so all possible data exchanges are implemented.

The P2P emulator architecture comprises two main components as shown in Fig. 7:

- the P2P Master;
- the modified peer client.

The aim of the first component is to coordinate P2P clients: it creates and terminates P2P clients according to a given probability distribution function. Creation and termination rates may be changed during simulation to simulate cases of flash crowds and crashes. The P2P Master delivers P2P traffic to the underlying network simulator (through the cross-layer interface), and generates background traffic to simulate network congestion. Peer clients in the emulator are modified to redirect connections among them towards the P2P Master (let us note that now all peer clients and the P2P Master reside in the same machine). P2P data exchanges are delivered to the Master which stores data and communicates NAPT of how many peers want to communicate with other peers. Subsequently NAPT simulates data transmission among the peers involved and then sends a notice to the P2P Master when data is received at destination. Finally, the P2P Master transmits the real data to the destination peer client. This procedure is shown in Fig. 7, where the arches represent data communication.

III. A RELEVANT CASE STUDY: VIDEO STREAMING OVER SPLITSTREAM

This section gives an outline of SplitStream, the P2P protocol for media streaming delivery, selected to demonstrate the effectiveness of the simulation/emulation approach.

SplitStream has been implemented using tree-based application-level multicast. It is designed to overcome the inherently unbalanced forwarding load in conventional tree-based multicast systems. In order to distribute the forwarding load over all participating peers, SplitStream uses multiple description coding (MDC) [10-16] to split the multicast stream into multiple stripes (descriptions). Stripes require approximately the same bandwidth and are distributed using separate multicast trees. At destination, the content can be reconstructed by any subset of the stripes with video quality proportional to the number of stripes received.

Participating peers may receive a subset of the descriptions, so they control their inbound bandwidth requirement in increments of B/k (B being the total bandwidth required for the stream and k the number of stripes). Likewise, peers may control their outbound bandwidth requirement in increments of B/k by limiting the number of children they adopt. Thus, SplitStream can accommodate nodes with different bandwidths and nodes with unequal inbound and outbound network capacities. This works well when the bandwidth bottleneck in communication between two nodes is either at the sender or at the receiver. If the bottleneck is elsewhere, nodes may be unable to receive all the desired descriptions. Nevertheless, even if an interior node in a description tree fails, clients deprived of the description are able to continue displaying the media stream at reduced quality until the description tree is

repaired. The challenge is to construct the *forest* of multicast trees so that each node is an interior node in one tree and a leaf node in all the remaining trees. The implementation of SplitStream is built using Scribe (an application-level group communication system) [5] and Pastry (a scalable, self-organizing structured peer-to-peer overlay network) [17].

IV. NUMERICAL RESULTS

CLAPS can be used to analyze the congestion level induced by SplitStream peers in the router buffers of the underlying network by monitoring both their average queue length and their packet loss probability. In addition, application-level performance metrics, such as PSNR at the receiver side and delay in streaming playout, can be derived. In this paper, the tool is applied to dimension the playout buffer at the receiver side in order to maintain both intra-description synchronization, i.e. time relationships between frames of the same description, and inter-description synchronization, i.e. time relationships between frames belonging to different descriptions related to the same video stream.

The network topology we use is shown in Fig. 8; it comprises two core routers and seven border routers (ISP routers). Three P2P clients are connected to each ISP router with a maximum download and upload capacity of 2 Mbit/s. The bandwidth between ISP routers and core routers is 4 Mbit/s, while the bandwidth between the core routers is 10 Mbit/s. The propagation delay is 5 ms for all the links. The number of descriptions emitted by the SplitStream video source is 16 (the default value in the distributed implementation). The video stream exchanged by peers is "Bridge close", coded with frame splitting MDC [22] (all 16 descriptions have the same bit rate) and a frame rate of 25 frame/sec. Frame splitting MDC works by splitting the normal sequence of frames into different groups, and then encoding each group by an MPEG encoder to obtain different descriptions. Fig. 9 shows how frame splitting MDC works.

The simulation lasts 2000 seconds. The arrival and departure rates of peers are exponentially distributed with an

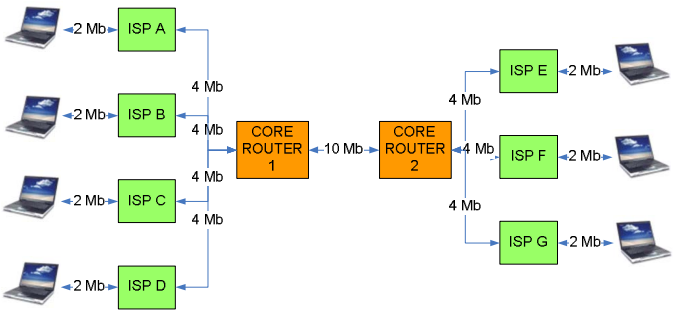

Figure 8. Case study network topology.

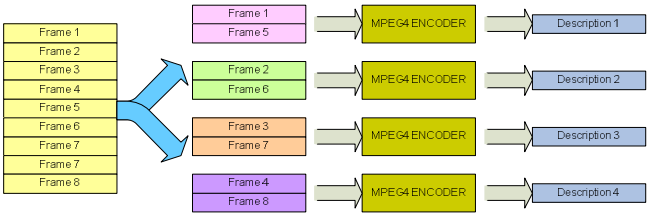

Figure 9. Model of frame splitting MDC.

average of 0.5 peers/s. In order to simulate flash crowd and flash crash occurrences, during the simulation the above arrival and departure rates are replaced by the values (0.01, 1.0) and (1.0, 0.01) respectively.

Besides the P2P traffic, the network is also loaded by a background traffic which is produced by 7 TCP greedy sources connected from the ISP's A, B, C, D, E, F and G to the ISP's E, E, F, G, A, B and C, respectively. Fig. 10 shows the temporal evolution of the number of active peers in the network. Let us note the occurrence of both a flash crowd and a flash crash at around 1000 seconds.

The tool allows us to compute the time instants at which each description frame is received by each peer; so Fig. 11 shows the percentage of frames lost by peers during their lifetime. Note that, although SplitStream is supported by TCP, frame losses occur due to the application layer topology evolution during the simulation; that is, when a peer is disconnected, its children lose one description. In the case we are analyzing the average frame loss is 43,85% and its variance is 0,027.

The normalized histogram of the total amount of frames received by all the peers per description is shown in Fig. 12. We can observe that the number of received frames is quite similar for each description; that is, SplitStream is able to uniformly distribute the content among all descriptions.

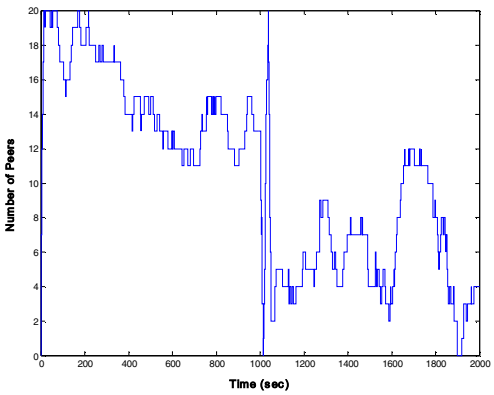

Figure 10. Temporal evolution of number of peers.

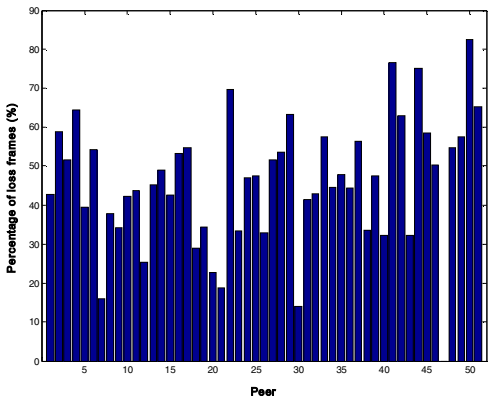

Figure 11. Percentage of frames lost by each peer.

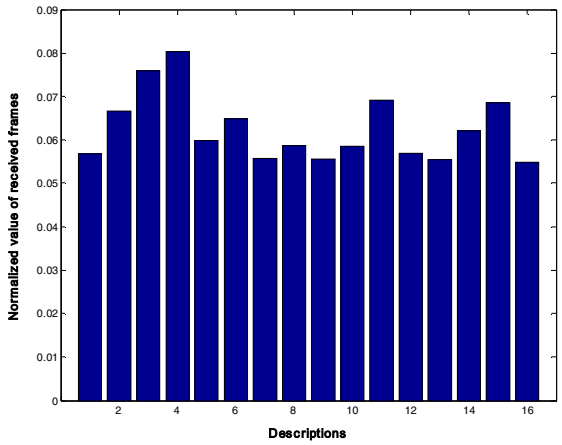

Figure 12. Normalized histogram of the total amount of frames received by all the peers per description.

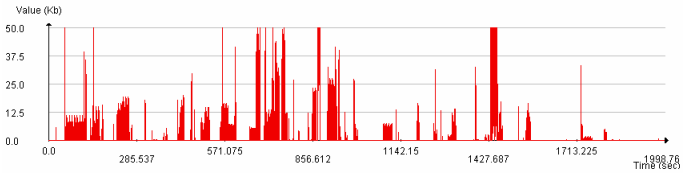

Figure 13. Queue length in core router 1.

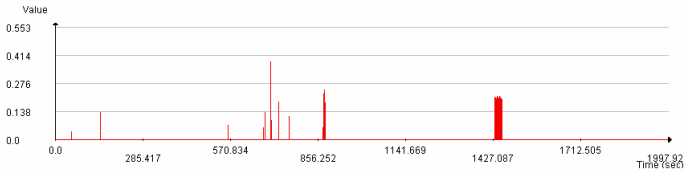

Figure 14. Packet loss rate in core router 1.

Obviously the video streaming QoS achieved depends on the underlying network congestion. All network routers may be congested by background traffic and this causes losses in the routers; so both variable transmission delay and TCP retransmissions compromise the reconstruction of the video stream at the receiving side. Figs. 13 and 14 show an example of queue length and packet loss rate in the routers.

In order to improve the video QoS, a playout buffer is used by receiving peers. We consider the delay (T_{Buffer}) introduced by the playout buffer of the reconstructed video stream as a whole, that is, the stream achieved at the destination by re-combining all the descriptions. More specifically, T_{Buffer} is the duration of the interval between the arrivals of the first and the N^{th} frame, where N is the playout buffer size, expressed in frames. By so doing, each peer starts streaming visualization when it receives N frames.

Below we will address the problem of dimensioning the playout buffer. Let us note that the first frame displayed, F_0, is not necessarily the first frame to arrive but the one with the lowest sequence number among the N frames that have arrived. A generic frame F_i is displayed if it arrives before its visualization time. To model this occurrence, a function $\varphi_N(F_i)$ representing the difference between the frame arrival time and its presentation time has been defined:

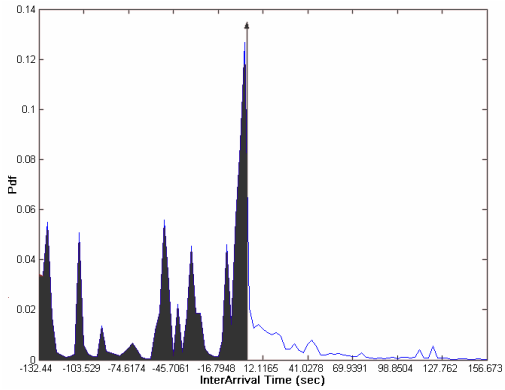

Figure 15. Probability density function of inter arrival frame time with buffer size of 10 frames.

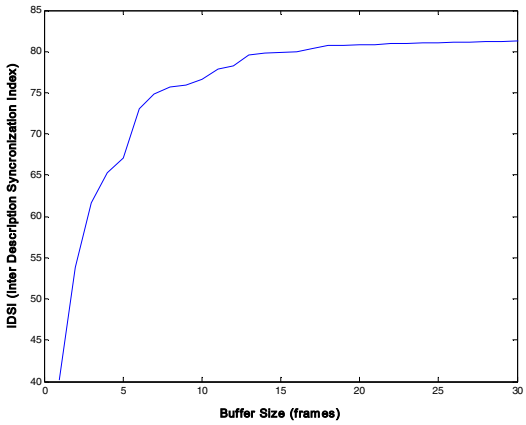

Figure 16. IDSI (Inter Description Synchronization Index).

$$\varphi_N(F_i) = \left(T_{F_i} - T_{F_0}\right) - \left\lfloor T_{Buffer_N} + \left(S_{F_i} - S_{F_0}\right)/R \right\rfloor \qquad (1)$$

where T_{F_0} and T_{F_i} are the F_0 and F_i frame arrival times, S_{F_0} and S_{F_i} represent their sequence numbers, and R is the video frame rate. So when $\varphi_N(F_i) \le 0$, then the frame arrives in time to be presented ; otherwise, when $\varphi_N(F_i) > 0$, it is discarded .

Analogously we define $\overline{\varphi}_N(F_i)$ as the $\varphi_N(F_i)$ value averaged on all the receiving peers. Fig. 15 shows the probability density function (pdf) of $\overline{\varphi}_N(F_i)$ when a playout buffer size of 10 frames has been used; we can observe that 76% of the received frames are visualized in the whole simulation.

In order to improve the analysis of buffer dimensioning, a new metric has been introduced, the IDSI (Inter-Description Synchronization Index), which represents the percentage of frames which arrive in time to be visualized when the playout buffer size is N:

$$IDSI_N = \int_{-\infty}^{0} \overline{\varphi}_N(\chi) d\chi \qquad (2)$$

For buffer sizes greater than 16, IDSI remains approximately constant (see Fig. 16), and so the best choice for the buffer size is 16 which achieves a good trade-off between the visualization delay and the percentage of visualized frames (about 80%).

V. CONCLUSIONS

Effective analysis of a P2P media streaming architecture cannot be made without considering the effects of IP network phenomena.

In order to solve this problem, a cross-layer approach has been proposed which provides accurate, scalable simulation of the P2P media streaming architecture. The approach uses a fluid-flow TCP/IP network simulator interfaced with a P2P overlay network emulator. The paper shows how the cross-layer approach can be used to analyze the inter-frame arrival time in order to dimension buffer size and evaluate the real performance of the system, while a SplitStream overlay network is used to support multiple description video streaming.

REFERENCES

[1] Banerjee S., Bhattacharjee B., and Kommareddy C., "Scalable application layer multicast", in Proc. Of ACM SIGCOMM 2002, Pittsburgh, PA, 2002, pp. 205-217.

[2] Barbera M., Busà A.G., Di Nuovo A.G., Lombardo A., Schembra G., "A simulation tool for tuning IP network parameters based on fluid-flow models and parallel genetic algorithms", Global Telecommunications Conference, Missouri, 2005. GLOBECOM apos;05. IEEE.

[3] Barbera M., Lombardo A., Schembra G., "A Fluid-Based Model of Time-Limited TCP Flows." Computer Networks Journal (Elsevier), 2004, Vol. 44, No.3 (Feb.), 275-288.

[4] Barbera M., Lombardo A., Schembra G., Trecarichi C. A., "A Fluid-Based Model of Policed RIO Router Networks Loaded by Time-Limited TCP Flows", In Proceedings of SPECTS, 2004, San Jose, California, U.S.A.

[5] Castro M., Druschel P., Kermarrec A.-M., and Rowstron A., "SCRIBE: A large-scale and decentralized application-level multicast infrastructure", IEEE JSAC, 20(8), Oct. 2002.

[6] Castro M., Druschel P., Kermarrec A.-M., Nandi A., Rowstron A., and Singh A., "SplitStream: Highbandwidth Content Distribution in a Cooperative Environment.", In Proc. SOSP (Oct. 2003).

[7] Chu Y., Rao S., and Zhang H., "A case for end system multicast", in Measurement and Modeling of Computer Systems, Santa Clara, California, 2000, pp. 1–12.

[8] Deshpande H., Bawa M., and Garcia-Molina H., "Streaming live media over a peer-to-peer network", in Work at CS-Stanford. Submitted for publication, 2002.

[9] Do T., Hua K. A., and Tantaoui M., "P2VoD: Providing fault tolerant video-on-demand streaming in peer-to-peer environment", In Proc. of the IEEE Int. Conf. on Communications (ICC 2004), Paris, France.

[10] Franchi N., Fumagalli M., Lancini R., and Tubaro S., "Multiple Description Video Coding for Scalable and Robust Transmission Over IP", in IEEE Transactions on circuits and systems for video technology, vol. 15, no. 3, march 2005.

[11] Guo Y., Suh K., Kurose J., and Towsley D., "P2cast: P2p patching scheme for vod service", Technical Report, Department of Computer Science, University of Massachusetts Amherst.

[12] Hefeeda M., Habib A., Boyan B., Xu D., and Bhargava B.. "PROMISE: peer-to-peer media streaming using CollectCast", Technical report, CS-TR 03-016, Purdue University, August 2003. Extended version.

[13] Misra V., Gong W.B., Towsley D., "Fluid-based analysis of a network of AQM routers supporting TCP flows with an application to RED". In Proceedings of SIGCOMM, 2000, Stockholm, Sweden.

[14] Padmanabhan V. N., Wang H. J., and Chou P. A., "Resilient Peer-to-Peer Streaming.", Technical Report MSR-TR-2003-11, Microsoft Research, Redmond, WA, March 2003.

[15] Ratnasamy S., Handley M., Karp R., and Shenker S., "Application-level multicast using content-addressable networks", in Proc of NGC, Nov. 2001.

[16] Reibman A. R., Jafakhani H., Wang Y., Orchard M.T., and Puri R., "Multiple description video coding using motion-compensated temporal prediction", IEEE Transactions on Circuits and Systems for Video Technology, vol. 12, no. 3, pp. 193--204, March 2002.

[17] Rowstron and Druschel P., "Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems", in Proc. IFIP/ACM International Conference on Distributed Systems Platforms (Middleware), Heidelberg, Germany, November 2001

[18] Tran D. A., Hua K. A., "A Peer-to-Peer Architecture for Media Streaming", IEEE Journal on Selected Areas in Communications, vol. 22, no. 1, January 2004.

[19] Xu D., Hefeeda M., Hambrusch S., Bhargava B. "On Peer-to-Peer Media Streaming", In Proc. of IEEE- ICDCS'02, Vienna, Austria, July 2002.

[20] http://www.isi.edu/nsnam/ns/

[21] http://www.diit.unict.it/arti/Tools/NAPT/

[22] http://trace.kom.aau.dk/tracemain.html

[23] http://research.microsoft.com/~antr/SplitStream/