OPSS: a simulator for large scale Peer-to-Peer streaming systems

Lorenzo Bracciale, Francesca Lo Piccolo, Dario Luzzi, Stefano Salsano

Dip. Electronic Engineering, Univ. of Rome "Tor Vergata"

Abstract

In this paper we present the OPSS simulator which is designed to simulate a large scale (i.e. in the order of 100K nodes) peer to peer streaming system. OPSS is able to simulate a fair (i.e. "TCP-like") sharing of the uplink and downlink bandwidth of each peer among the different peers that are connected. OPSS is based on an existing simulator for TCP-like bandwidth sharing and it offers the possibility to implement the logic of the P2P distribution algorithm as separate module. Therefore it allows to simulate the behavior of arbitrary tree-based or mesh-based approaches. In this paper, we implemented two trivial tree-based and mesh-based approaches for which we could easily find an analytic model of chunk distribution delay. The results of the model have been compared with the simulation output, showing an excellent fit. Source code of OPSS is available under the GPL license.

1. Introduction

Even if IP multicast has originally been introduced with the purpose of offering point-to-multipoint content distribution services, many deployment issues have still to be solved. As argued in [1], IP multicast calls for multicast-capable routers able to maintain per group state information, which seriously limit its scalability. Second, IP multicast is a best effort service, and providing higher level features such as reliability, congestion control, flow control, and security has been shown to be more difficult than in the unicast case. Finally, IP multicast requires changes at the infrastructural level, and this slows down the deployment pace.

Due to this, more and more researchers are investigating application level multicast as solution to stream multimedia audio and video content from a source to a large number of end users. This approach consists of end hosts, which according to peer-to-peer (P2P) paradigm auto-organize themselves in an overlay network out of unicast tunnels across participating overlay nodes. Data relaying among overlay nodes allows then for the multicast service.

Overlay multicast distribution trees represent the most natural way of extending IP level multicast to application level. To name a few, NARADA [2], HMTP [3], NICE [4], ZIGZAG [5], CoopNet [6], SplitStream [7] are tree-based peer-to-peer streaming systems. However, while tree-based topologies are well suited to dedicated IP multicast routers, they could suffer from re-configurability problems in presence of the high churn rate of P2P nodes. In reason of this, overlay mesh-based and unstructured topologies have also been proposed. CoolStreaming/DONet [8] and GridMedia [9] offer examples of the latter approach.

The performance analysis of a large scale real-time streaming system is far from trivial. One has to consider both the "control plane" related aspects (how the peers exchange the information of chunk availability and schedule the transmission of chunks each other) and the "transport plane" related aspects (how much does it take to transmit the chunks given the available transmission resources on the access and backbone links and considering the competition among the different peers). In section 2 we will review the current approaches to performance evaluation of real-time peer to peer streaming systems highlighting their limitations. In section 3 we present our contribution: the OOPS simulator [10]. OPSS is able to consider the transport plane aspects related to the sharing of transmission resources for a large scale network (in the order of 100K peers). In section 4, OOPS is used to provide a simulative evaluation of two real-time streaming algorithms, with respect to the chunk transmission delay. An analytical performance analysis of the two algorithms is also given. Note that the purpose of this section is not the proposal of a specific algorithm nor the comparison of a set of algorithms. We rather aim at showing that OOPS is a valuable tool for performance analysis and we show it by looking at the excellent matching between the simulative performance analysis and the analytic results.

2. Review of existing work

In this section we will first briefly introduce some existing systems for peer-to-peer live media streaming, trying to focus on the existing works that try to evaluate their performance. Then we will analyze a set of existing simulators for P2P systems.

CoolStreaming/DONet [8] is a Data-driven Overlay Network for live media streaming. The overlay membership management is built on a gossip-based protocol [11]. In gossip-based protocols, a node sends a newly generated message to a set of randomly selected nodes; these nodes do similarly in the next round, and so other nodes do until the message is spread. The video stream is divided into segments or chunks, and chunk availability in the node buffer is represented by a Buffer Map (BM), where bit 1 and 0 indicate that a segment is respectively available and unavailable. Each node learns about chunk availability by periodically exchanging its BM with the BMs of its partners. DONet is built on a "pull" approach, i.e. chunk downloads start only if a node requests that chunk from a supplier partner. The performance of DONet has been evaluated in [8] using PlanetLab [12][13]. PlanetLab is a global overlay network to support the design and the performance evaluation of applications widely distributed over the Internet. In [8] the control overhead and the continuity index are considered as performance metrics. The former represents the ratio between control traffic volume and video traffic volume; the latter is the number of segments that arrive before or on playback deadlines over the total number of segments. DONet performances are also compared with the performance of a tree-based overlay streaming system. Besides the continuity index, the average hop count is considered as raw approximation of end-to-end delay for delivering each segment. The number of used PlanetLab nodes ranges from 10 to 200 (passing through 50,100,150).

GridMedia [9] is a unstructured P2P live media streaming system which tries to overcome the limitation of the DONet "pull" approach. It is based on a push-pull approach that consists in requesting stream packets in pull mode at start up and having nodes relaying stream packets without explicit request in the immediate following phase. PlanetLab testbed is used to evaluate GRidMedia performance in [9]. Pull and pull-push approaches are compared. The proposed experimental results relate to a number of PlanetLab nodes ranging from 300 to 340. Among the performance indexes that are taken into account, we mention i) absolute delay, that is the delay between the sampling time at the server and the playback time at the local node; ii) the delivery ratio, that is the ratio between the number of stream packets arriving before or right on absolute playback deadline and the total number of packets; iii) the α-playback-time, that is the minimum absolute delay at which the delivery ratio is larger than α ($0 \leq \alpha \leq 1$); iv) the control overhead of the gossip protocol, that is the average ratio between the control traffic and the total traffic at each node.

The same authors as [9] focus in [13] on the optimal streaming scheduling problem in data-driven overlay networks. The optimal streaming scheduling problem aims at addressing how each node optimally decides from which neighbor to request which block, and how it allocates its limited outbound bandwidth to every neighbor, in order to maximize the throughput. This scheduling problem is formulated as a classical min-cost network flow problem and two resolution strategies are considered. The first one is a global optimal solution which assumes a centralized knowledge of all network state, the second one is an heuristic algorithm which is fully distributed and calls for only local information exchange. To validate their algorithm, they use a discrete event-driven P2P simulator to simulate a data driven overlay network of size 500 nodes. However, the authors do not give details about the kinf of simulator they use. The low number of simulated nodes makes the hypothesis of packet-level simulator reasonable. The considered metric is the average delivery ratio, that is the ratio between the number of packets arriving before or right on the playback deadline averaged on all the nodes and the total number of packets. The proposed solution is compared with DONet, ChainSaw [15] and round-robin streaming scheduling.

NICE [4] is another overlay peer to peer live streaming system, but differently from the previously described systems it is built on a hierarchically connected overlay topology. The host hierarchy is used to define different overlay structures for control messages and data delivery paths. End-to-end latency is used as distance metric between host and it drives the association of nodes in clusters. The NICE clusters and layers are created, maintained and eventually repaired by a fully distributed protocol. Data overlay delivery path is instead the tree rooted at data source and implicitly defined by the control overlay topology hierarchy. A packet level simulator is used to evaluate NICE performance. Network topologies are generated using the Transit-Stub graph model and the GT-ITM topology generator [20]. The number of end hosts in the multicast group is varied between 8 and 2048. Performance metrics such as the average link stress and the average path length are investigated. The first one is the number of identical packets sent over each underlying network link averaged across all the network links. The second one is the length (in number of hops) of the path from the source to the hosts averaged across all the hosts. The fraction of members that correctly receive the data packets in case of node failures and the byte-overhead for control traffic at the access links of the end-hosts are evaluated too. The achieved results are compared with the results obtained by simulating the application-layer multicast protocol Narada [3].

There are also other p2p streaming applications like PPLive [16] or Sopcast [17] that are widely deployed but whose algorithms are not under public domain. The only solution for investigating their performance and behaviors is to use a black-box measurement-based approach, as in [18] and in [19].

Regardless of any of the peer-to-peer live streaming systems described so far, a large number of P2P simulators has recently emerged. Most of them mainly focus on simulating the resource search phase and the related query message handling. This is the case of Aurora [21] and Serapis [22], which model the key announcement, insert or request process of Freenet-like systems. Similarly, P2Psim [23], FreePastry [24] and the Chord simulator [25], simulate only the DHT-based (Distributed Hash Table) search phase. A similar approach is employed in other general-purpose P2P simulators, such as Neurogrid [26][27], 3LS [28], and Peersim [29][30].

With regard the low level network dynamics, either their effect is totally neglected, as in [21] and [26], where the overlay message transmission is immediate, or an exponentially distributed packet delay is used, as in [25], or the concept of distance between any two nodes is somehow defined and the overlay message transmission time is identified with the latency between the relative nodes, as in [22][23][24][28][29].

Although the P2P query/search phases are undoubtedly representative of a P2P system, there is plenty of interest in quantitatively characterizing performance figures related to the resource distribution process among involved peers. All the previously mentioned simulation platforms are not suitable to this purpose, as they neglect the process of distributing data across peers.

To properly model the data distribution phase, GnutellaSim [31][32] interfaces with the ns-2 [33] discrete event packet-based network simulator, which provides a very detailed packet-level simulation model of the underlying transport network. However, such a simulation model compromises the scalability of the resulting simulation, as only a few hundreds nodes may be properly simulated in reasonable time with such a level of details.

3. Simulator description

3.1 Objective

The review of existing research works on the topic of P2P video streaming has pointed out that it is possible to identify three different approaches of performance evaluation: i) measurement-based studies or real systems ii) experimental testbeds, such as PlanetLab, and iii) simulation tools. Measurement based studies does not allow to consider different alternatives and to evaluate performance in advance of building and deploying a system. Experimental testbeds and current simulation tools suffer from scalability problems for different reasons. Experimental testbeds would require a large network of emulator nodes, which is not easy to realize and to manage. The current simulation tools either are mostly oriented to the search phase and neglect the content distribution phase or perform the simulation at packet level, making unpractical to simulate a P2P live streaming system over a network in the order of 100K peers. Typical available results concern network size in the order of hundreds or few thousands peers. This is not representative of real-life P2P video streaming systems, which aim at streaming live multimedia content to a very large number, several hundred thousands if not millions, of users. It may be the case that network dynamics simulated in small-scale networks are not representative of large-scale P2P system deployments.

On basis of the above observation, we propose OPSS [10], a new simulative approach to make P2P video streaming performance evaluation scalable.

3.2 How does OPSS achieve scalability?

In order to circumvent the tight scalability limits imposed by packet-based simulators and simultaneously to model networks dynamic with acceptable accuracy level, OPSS was conceived as discrete-event fluid-flow simulator. This allows to simulate the data distribution at the flow level, i.e. neglecting transmissions of single packets but focusing on events, such as start/end of a file or a file chunk transmission, which lead to a variation in the rate of the connections among peers. This approach dramatically reduces the number of simulation events and the related memory and computational load with respect to packet-level simulation, while retaining a satisfactory accuracy in the model of the data delivery process. We also assume that all active connections share the available transmission resources using TCP or a “TCP friendly” approach. Under this hypothesis it is possible to use a *max-min fair* [37] rate allocation algorithm in order to evaluate the available capacity for each connection, given the links bandwidth constraint. The notion of max-min fair allocation is based on the following premises: i) no entity should receive an allocation larger than its demand, and ii) increasing the allocation of any entity should not result in the decrease of the allocation of another entity that received an equal or smaller allocation. It well approximates the TCP-like sharing uploading and downloading bandwidth between concurrent flows.

Evaluating the max-min fair rate allocation in a network of hundred of thousand peers, with millions of active connections is not an easy task. The classical centralized implementation of max-min fair rate allocation (as suggested for example in [37]) does not scale well for the network dimensions of our interest. The overall computational load of max-min fair allocation in our scenario is the product of two different factors. First factor, a max-min fair re-computation is required every time a new traffic relation is established, or an old traffic relation is completed or interrupted (e.g. because of peer disconnection), the frequency of these events being linearly dependent on the number of simulated peers[1]. Second factor is the load of evaluating a max-min fair rate allocation over the full network.

The implementation suggested in [37] requires to re-compute the allocated rates per each network node, and thus it results in a complexity which grows linearly with the number of simulated peers. Whereas the first factor depends only on the simulated P2P application logic, it is possible to act on the second factor to reduce the computational load of max-min fair rate allocation.

As it was observed in [38], when a new connection is established or an old connection is interrupted or completed, such events may affect only a subset of the existing connections. The above observation have been exploited to develop an exact and more efficient max-min fair rate allocation implementation under the assumption

[1] In fact, given a peer connected to the network with an access link of capacity C bytes/second and uploading/downloading a number of files with size L bytes according to a processor-sharing queueing discipline, the average service rate is (assuming that all the link capacity is used) C/L files/second. Since a re-computation of the max-min fair rate allocation algorithm is required at each change in the served traffic relations, given N nodes, the number of re-computations per second is approximated as N C/L, hence linearly dependent on the network size. Note that we have neglected the fact that new arriving requests, if not queued, add further re-computation events.

of bottleneck links only in the access side of the network. More details about such implementation may be found in [38]. The reported results show that the algorithm proposed in [38] outperforms traditional max-min computation approaches by as much as a factor 100 for a million nodes network.

We have built the OPSS simulator starting from the implementation of the max-min fair rate allocation algorithm proposed in [38]. As in [38] we made the assumption that rate bottlenecks occur only in the access part of the network. This assumption is employed in both analytical models appeared in the literature [35] as well as in simulation programs such as [26] and [36]. It is justified by the current bandwidth gap between access links and core network trunk, and by the empirical observation that practical P2P clients typically further throttle the upload bandwidth, which in most cases results fully utilized by the uploading connections.

Obviously, the most serious limit in our approach is that the max-min fair bandwidth allocation well approximates a TCP-like steady state bandwidth sharing. Due to this, our approach is well suited to the case of persistent connections between peers. In addition, it is currently impossible to simulate Transit-Stub topologies such as the ones generated by GT-ITM topology generator. To overcome this last limit, extending the efficient and exact implementation of max-min fair rate allocation proposed in [38] to the case of generic topologies, even if not trivial, could be a reasonable solution.

3.3 Implementation details

OPSS was designed according to a modular implementation logic. All the code is written in C++ and publicly available [10] under GPL license. The simulator architecture is structured in three basic layers: User, Overlay and Network. The User layer represents the peer behavior, taking into account for example connection and disconnection policies (i.e. the “churn” distribution). The Overlay layer represents the specific peer to peer live streaming mechanism, including the “control” communication between nodes and the logic the nodes use to connect each other and to decide which chunk to download from what peer. The Network layer represents the network behavior, implementing the optimized max-min fair rate allocation approach as described in [38].

Separating network level from application-related levels makes OPSS a very flexible simulator, as it offers the possibility of implementing the logic of P2P streaming application as separate module. Moreover, users exploit different basic classes provided by User and Overlay levels and potentially implement any different kinds of P2P streaming algorithms. A set of built-in basic classes offer support for either tree-based or mesh-based topologies and allow very flexible design of P2P streaming applications. For further information about how to write algorithms, please refer to the guide available on the reference site.

The simulator outputs a log file where events are dumped with the corresponding time. The set of events that will be included in the log file is customizable to prevent log files to become too big in size.

3.4 Performance metrics

In this section we discuss the performance metrics we want to evaluate using the OOPS simulator. Due to the characteristic of the application (streaming of real-time multimedia flows), the considered metrics will be related to the delay of receiving the chunks of the flow. In the definition and evaluation of these delay related metrics, we need also to carefully consider that some chunks may not be received by some receiver.

Consider a real time multimedia streaming system where the multimedia stream is divided in chunks of duration T [s]. The origination of chunks starts at $t=t_{start}=0$, and ends up at $t=t_{end}$. The total number of originated chunks will be t_{end}/T. In order to produce consistent measurements, we need to observe the system in an interval of duration $w_{end}–w_{start}$, with $0 \le w_{start} < w_{end} \le t_{end}$. For simplicity, we assume that w_{start} corresponds to the generation time of one given chunk, and we observe C chunks starting from the one created at w_{start}. Therefore the originating time of the observed chunks will be $t_c=w_{start}+c·T$, c=0,1,2,…,C-1. The first observed chunk originates at $t_1=w_{start}$ and the last observed chunk at $t_C=w_{start}+(C-1)·T$. In order to allow the last chunk to be received by all the receivers, we need to close our observation interval at a $w_{end}>t_C=w_{start}+(C-1)·T$. In particular let $w_{end}=t_C+D_{max}$, where D_{max} is the maximum delay we are considering in our evaluation of the system. Note that the origination of chunks will continue also after the origination of the last observed chunk, in the time interval in which we still observing the system waiting for the last observed chunk to be received.

Assume that there are *(N-1)*[2] receiver nodes and that node n receives the c-th observed chunk at time $t_r(c,n)=w_{start}+(c-1)·T+d(c,n)$. Then d(c,n) is the delay of chunk c at node n. Let us consider the last chunk C, if $d(C,n)>D_{max}$, the event is out of our observation window and it will be lost. For the generic chunk c, a chunk received event goes out of our observation window if $d(c,n)>D_{max}+(C-c+1)·T$. From a methodological point of view, it is not good that the maximum observable delay for a chunk depends on the chunk number c, therefore we think it is better to set D_{max} as maximum chunk delay for all chunks and to consider that a chunk is lost if $d(c,n)>D_{max}$. Now, we can define $r_{Dmax}(c,n)=1$ if the chunk c is received by node n with $d(c,n)<=D_{max}$, $r_{Dmax}(c,n)=0$ if the chunk is not received or if it is received with $d(c,i)>D_{max}$.

[2] The total node number is N if we add the stream source to the receiver nodes

We consider that a receiver peer node n can be active or not by defining its activity function a(t,n) as follows: a(t,n)=1 if the node n is active at time t, a(t,n)=0 if the node n is not active at time t. The activity A(n) of a node n during the observation window [w_{start}, w_{end}] will be:

$$A(n) = \frac{1}{w_{end} - w_{start}} \int_{w_{start}}^{w_{end}} a(t,n)dt \tag{1}$$

The number of active nodes at time t is given by:

$$N_A(t) = \sum_n a(t,n) \tag{2}$$

Therefore the average number of active nodes N_A over the observation window will be:

$$N_A = \sum_n A(n) = \frac{1}{w_{end} - w_{start}} \int_{w_{start}}^{w_{end}} N_A(t)dt \tag{3}$$

We want to define the Chunk Delivery Ratio (CDR) for a chunk c (that depends on D_{max}), as the ratio between the nodes that have received the chunk c and the average number of active nodes when the chunk is originated. Note that we should consider the average number of active nodes in a time interval following the chunk origination event as this is the number of potential receivers for the chunk. As the chunk delivery delay is variable, it is not clear over which time interval we should average $N_A(t)$. A simpler solution is to define a "conventional" Chunk Delivery Ratio for a chunk c using the overall average number of active nodes over the observation window N_A. This is reasonable if the average number of active nodes does not change over time.

$$CDR_{D_{\max}}(c) = \frac{1}{N_A} \sum_n r_{D_{\max}}(c,n) \tag{4}$$

We can also define the overall Chunk Delivery Ratio, (that depends on D_{max} as well):

$$CDR_{D_{\max}} = \frac{1}{C} \sum_c CDR_{D_{\max}}(c) = \frac{1}{C \cdot N_A} \sum_{c,n} r_{D_{\max}}(c,n) \tag{5}$$

Now we can consider the average chunk delay for a chunk c, which can be evaluated by averaging the delay over all nodes that received the chunk (with a delay lower or equal to D_{max}):

$$\overline{d}_{D_{\max}}^{chunk}(c) = \frac{1}{N_A \cdot CDR_{D_{\max}}(c)} \sum_{n \mid r(c,n)=1} d(c,n) \tag{6}$$

It can be also of interest to consider the perspective of a given node n and to evaluate the perceived performances. First of all we can evaluate the chunk delivery fraction seen by a given node n:

$$CDR_{D_{\max}}^{node}(n) = \frac{1}{C \cdot A(n)} \sum_{c} r(c,n) \qquad (7)$$

Then we can evaluate the average chunk delay perceived by a generic node n:

$$\overline{d}_{D_{\max}}^{node}(n) = \frac{1}{C \cdot A(n) \cdot CDR_{D_{\max}}^{node}(n)} \sum_{c|r(c,n)=1} d(c,n) \qquad (8)$$

It is also interesting to consider the α Chunk Delay Percentile (CDP) of the distribution of chunk delay perceived by a generic node n. Typical values that we can consider are α = 95, α = 99.

$$CDP_{D_{\max}}^{node}(\alpha,n) = x \left| P\{d(c,n) < x\} = \alpha / 100 \right. \qquad (9)$$

The overall average chunk delay can be evaluated by averaging the delay over all received chunks (received with a delay lower or equal to $D_{\max}$):

$$\overline{d}_{D_{\max}} = \frac{1}{C \cdot N_A \cdot CDR_{D_{\max}}} \sum_{c,n|r(c,n)=1} d(c,n) \qquad (10)$$

Note that this is different than averaging $\overline{d}_{D_{\max}}^{chunk}(c)$ over c.

4. The evaluated P2P algorithm

The goal of this section is to show the OPSS performance in terms of both scalability and capability of producing correct results. To this purpose, we simulated two "trivial" P2P streaming distribution schemes ("Balanced M-ary tree" and "Trivial Mesh"), for which we could easily derive analytical models. We compared the experimental results achieved by OPSS with the result of the analytical model, verifying the OPSS correctness. On the other hand, OPSS scalability was evaluated by simulating an ever increasing number of nodes. In the following subsections, we describe the simulated distribution schemes and we report the corresponding analytical models and experimental results.

4.1 Balanced M-ary tree

This simulated stream distribution scheme corresponds to a balanced M-ary distribution tree. The stream source is the root of the tree. The stream video is divided into segments or chunks, and *R=1/T* denotes the source chunk rate [chunk/s]. Each node downloads chunks from one single node, and it uploads chunks to *M* nodes. According to the tree graphs jargon, each node has *M* children. The simulation foresees that all the nodes (including the stream source) join the system simultaneously and form the distribution tree. We also assume a static situation, in which all the nodes persist through the whole lifetime of the simulation.

Due to the last assumption, the issues related to the number of active nodes at a time instant and the average number of active nodes may be neglected. Another assumption we make is that $w_{start}=t_1=0$, where according to subsection 3.4 w_{start} and t_1 denote respectively the observation start time and the first chunk creation time. The c-th chunk will be referred to as c and the relative creation time is $t_c=(c-1)/R$. We now introduce the *level* concept: with reference to a node, the level l represents its distance from stream source as number of hops in the overlay tree. The level of the stream source is $l = 0$, while the last level is denoted as $l = L$. If all the levels are complete, the number of nodes at level l is M^l and the total number of nodes is $N = \sum_{l=0}^{L} M^l$. In the following, we will always consider trees with complete levels.

All the nodes are assigned an access link with uplink and downlink capacities W_{up} and W_{down} [chunk/sec]. To simplify, we assume symmetrical access links, that means also $W = W_{up} = W_{down}$. As consequence, each node downloads chunks at W/M chunk/sec from its father in the tree. If $W/M < R$, the distribution system cannot work as each node does not have enough capacity to download the stream of chunks. We thus restrict our attention to the case in which the available portion of the father's node uplink capacity is greater than or equal to the rate of the stream to be received: $W/M \geq R$.

It is convenient to express the delay of chunk *c* at node *n* in terms of the corresponding node level *l*. The reason is that all the nodes at the same level perceive the same delay. Specifically, given the level l, $l = 1,2,...L$, and the chunk c, the corresponding chunk delay is

$$d(c,l) = \frac{M}{W} \cdot l \tag{11}$$

The average chunk delay for chunk c is (note that it does not depend on c).

$$\overline{d}^{chunk}(c) = \frac{\sum_{l=1}^{L} M^l \cdot d(c,l)}{\sum_{l=1}^{L} M^l} = \frac{M\{M^L[L(M-1)-1]+1\}}{W(M-1)(M^L-1)} \tag{12}$$

The reference to D_{max} is omitted, as we are assuming an ideal system where chunks are not lost, we would only need to set $D_{max} > L \cdot W/M$ so that we are able to observe all chunk arrival events.

The average chunk delay perceived by a generic node at level *l* is

$$\overline{d}^{node}(l) = \frac{1}{C}\sum_{c=1}^{C} d(c,l) = \frac{M}{W} l \qquad (13)$$

The assumption in (13) is that C chunks are observed during the observation window.

Let us now consider the experimental results we achieved by setting $M = 2$. In more details, we simulated a set of binary distribution trees by varying the maximum depth of the tree L in the range [1,17] (e.g. the node number ranges from 3 to 131171). Figure 1 illustrates the binary tree corresponding to L=4. Observation time end w_{end} was set to 3600 sec. Moreover, W=2 [chunk/sec] and R=1 [chunk/sec] were used as upload capacity and chunk creation rate. We note that the simulations are "deterministic" simulations as there is no variability in the input data (we have a stream to be distributed of rate R chunk/second) and the nodes are always active. Therefore each simulation result is related to a single simulation run we will not indicate any confidence interval (the same applies to next section 4.2).

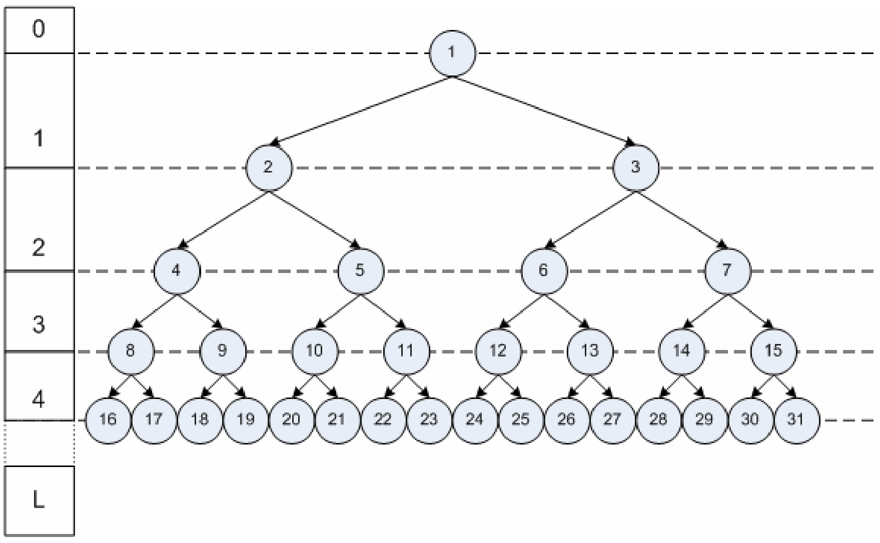

Figure 1 – Binary distribution tree with N=31 nodes

The chunk delivery ratio $CDR(c)$ defined in (4) is shown in Figure 2 versus c. As the observation end time w_{end} is 3600 sec and the last observed chunk is generated at 3600, we are not able to observe the complete diffusion of all the chunks created during the observation window. The correct approach in order to observe all the reception events is set C to the greatest c such that the condition $\frac{M}{W}(L + c - 1) < w_{end}$ is verified. For example, when the simulated nodes are 255, solving the above condition leads to c=3593, corresponding to the vertical step in the curve for 255 nodes.

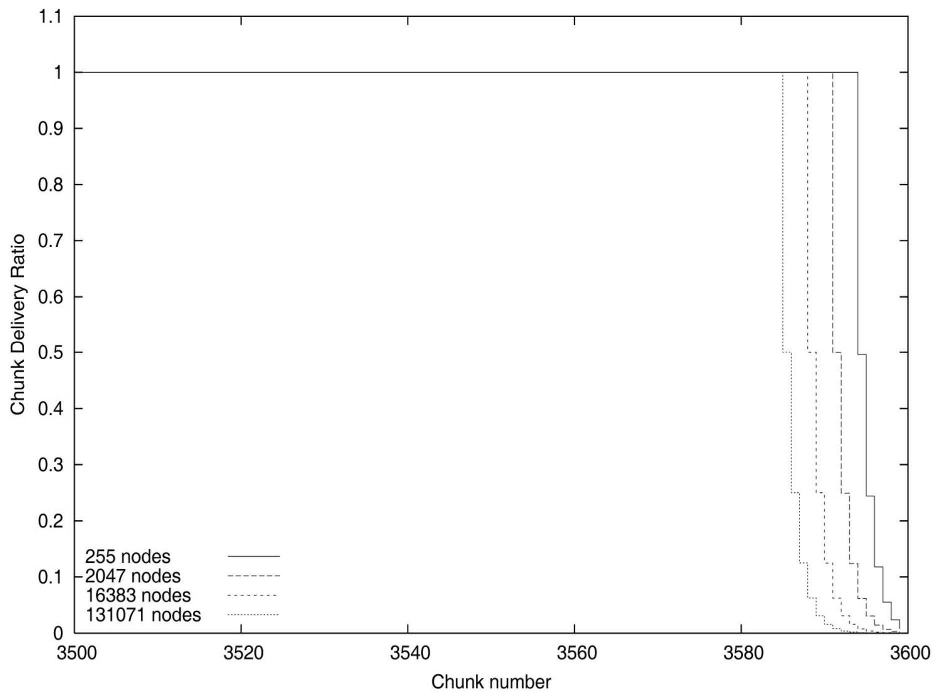

Figure 2 - Chunk delivery ratio

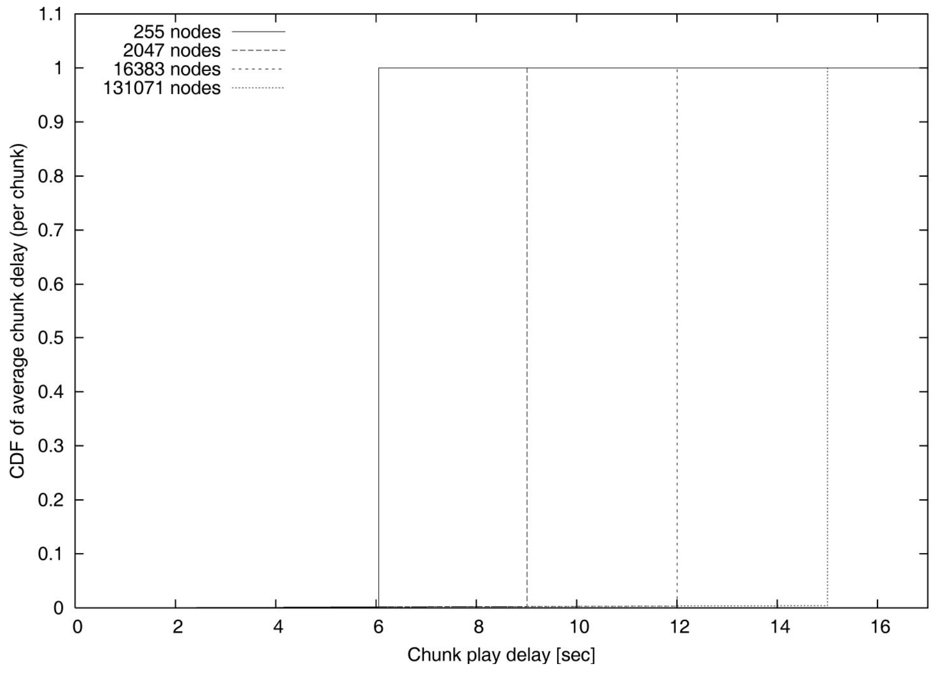

Figure 3 – Cumulative distribution function of average chunk delay for a given chunk

The cumulative distribution of the average chunk delay for a given chunk $\overline{d}^{chunk}(c)$, defined in (6), is illustrated in Figure 3. According to equation (12), the cumulative distribution function confirms that the average chunk delay is constant, and there is a perfect matching between the values achieved by equation (12) and the values achieved by simulation.

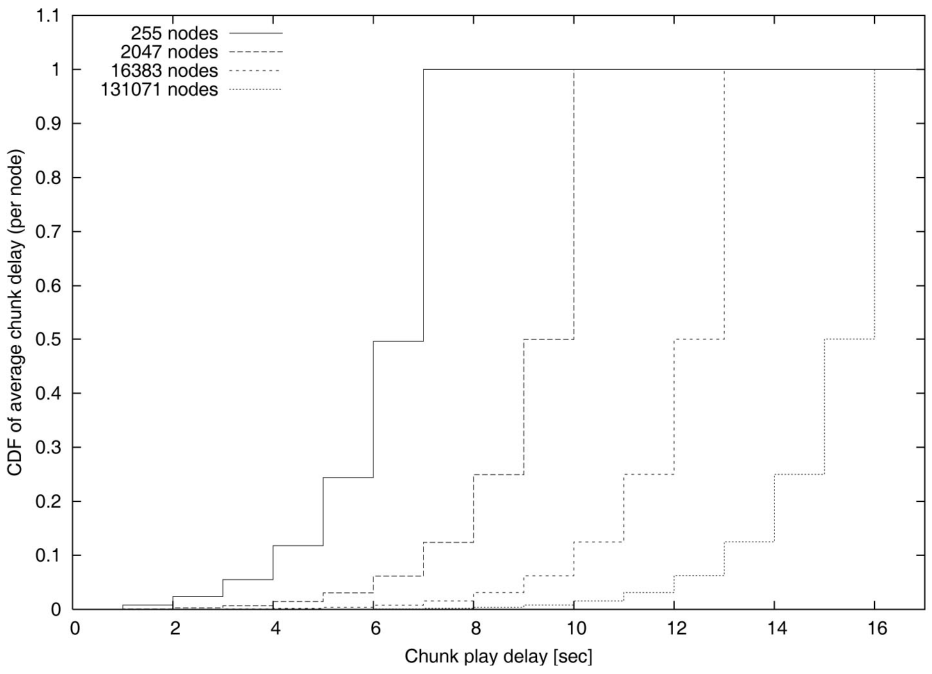

Figure 4 – Cumulative distribution function of average chunk delay at a given level

Figure 4 shows the cumulative distribution function of the average chunk delay perceived by the generic node. Given a number of simulated nodes, the steps correspond to the different tree levels and their probability values may be deduced from the ratio between the number of nodes in that level and the total number of nodes.

4.2 Trivial mesh

The algorithm represents a really simple mesh-based streaming distribution system. Specifically, the stream source node generates chunks at rate *R* [chunk/sec] and it uploads *S* chunks simultaneously to *S* nodes. All other nodes (i.e. excluding the source node) maintain *S* connections for chunk download and *S* connections for chunk upload, and they use them simultaneously to upload/download different chunks. Thus nodes form groups of *S* nodes. The first *S* nodes are connected directly with stream source and download the stream chunks from it. The second group of *S* nodes opens a connection with each node of the first group to download available chunks, and so on. In such a way, if we further assume that the total number of nodes is $N = LS+1$, it is possible to identify *L* different levels, each consisting in a group of *S* nodes. The previous assumption implies also that all levels are complete. Figure 5 shows the trivial mesh topology corresponding to $N=17$ and $S=4$.

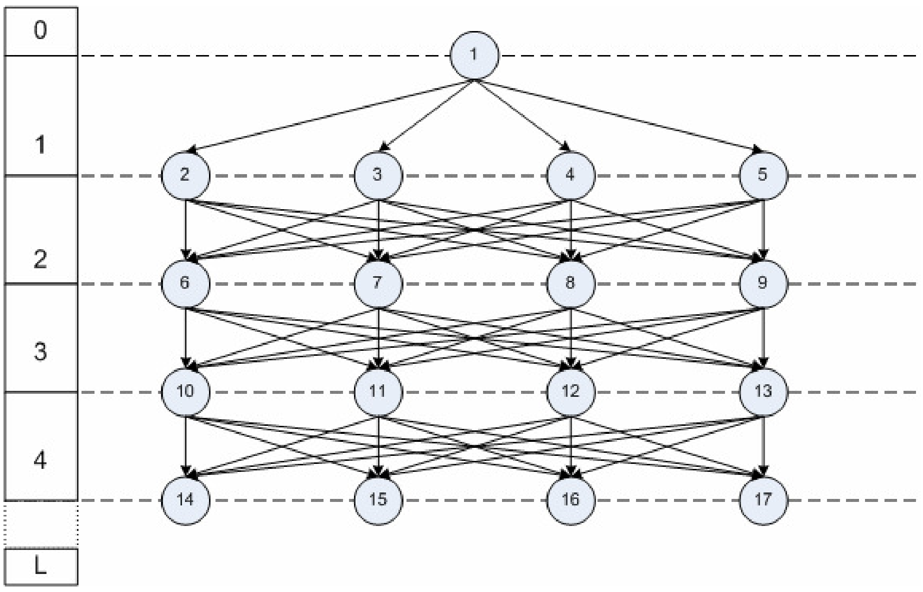

Figure 5 – Trivial mesh with N=17 nodes and S=4

Building the overlay topology according to this "trivial mesh" scheme is not smart, as the underlying algorithm does not allow to increase the number of peers at each level: the tree depth increases linearly with the node number and so it will increase the chunk transfer delay. Anyway the purpose of our work now is just to check the compliance of the OPSS simulator with the expected results, and we are going to provide an analytical model of this "trivial mesh". Like in the previous experiment, we assume that nodes persist through the whole simulation time. Moreover, with regard to access link upload and download capacities, the condition $W_{up} = W_{down} = W$ [chunk/sec] olds. There is however one exception: the stream source has a capacity that is S times the capacity of other nodes. This means that the S nodes connected to the stream source download chunks at rate W. The other nodes may download S chunks in parallel from S different suppliers, and each chunk is downloaded at a rate of W / S [chunk/sec]. We will focus on the case where $W = R$.

Since all the nodes in a level perceive the same chunk delay, we can refer to levels instead of individual nodes. With regard to the observation window, we assume that i) $w_{start} = t_1 = 0$, ii) the creation time of the generic c-th chunk is $t_c = (c-1)/R$. In addition, we assume that source node and level 1 nodes connect at time w_{start} while all other nodes connect simultaneously at time instant $t_S = S / R$, i.e. when the first S chunks have already been created and transferred to level one nodes. Therefore, at instant t_s each node of level 2 will open S connection to download the first S chunks from the S nodes at level 1. The download of these S chunks will last S/W seconds. As W=R in our hypothesis, the download will last exactly TS seconds. When this downloads end, there will be S new chunks available at level one nodes an the level 2 nodes will start S new downloads from level 1 node. This procedure is straightforwardly replicated in all levels below.

Given the level l, $l = 1,2,...L$, and the chunk c, the corresponding chunk delay is

$$d(c,l) = \frac{S}{R} + \left[ceil\left(\frac{c}{S}\right) + (l-1) \right] \frac{S}{W} - \frac{c-1}{R} \tag{14}$$

This delay will be periodic of period S, as a burst of S chunks will be received at the same time by a node, the more recent chunk of the burst will experience the lower delay, the older chunk of the burst will experience the higher delay and the delay difference among two next chunks in the burst is T.

The average chunk delay for the chunk c is

$$\overline{d}^{chunk}(c) = \frac{\sum_{l=1}^{L} S \cdot d(c,l)}{\sum_{l=1}^{L} S} = \frac{KS(L-1) + 2KS \cdot ceil\left(\frac{c}{S}\right) - 2(c-S)}{2KW} \tag{15}$$

The average chunk delay perceived by a generic node at level l is evaluated under the assumption that the number of observed chunks C is an integer multiple of S, e.g. $C = JS$ with $J \in Z^+$,

$$\overline{d}^{node}(l) = \frac{1}{C} \sum_{c=1}^{C} d(c,l) = \frac{S[R(B-1+2l) - W(B-1)]}{2RW} \tag{16}$$

At this point we report the experimental results we achieved by setting $S = 4$. The node number N ranges from 101 to 3601 passing through 1001 and 3401. Observation end time was set to $w_{end} = 3600$. Moreover, $W = R = 1$ chunk/sec.

The chunk delivery ratio is shown in Figure 6. According to formula (16), the average chunk delay at level l is linearly dependent of l. This means that, due to the fixed observation window, when the node number and consequently the level number grow, an ever increasing number of chunk reception events will be lost. When the node number is 3601, the first created chunk is received only by nodes in the first L-1 levels. In reason of this, simulating a higher node number does not make sense if the observation window is not accordingly increased.

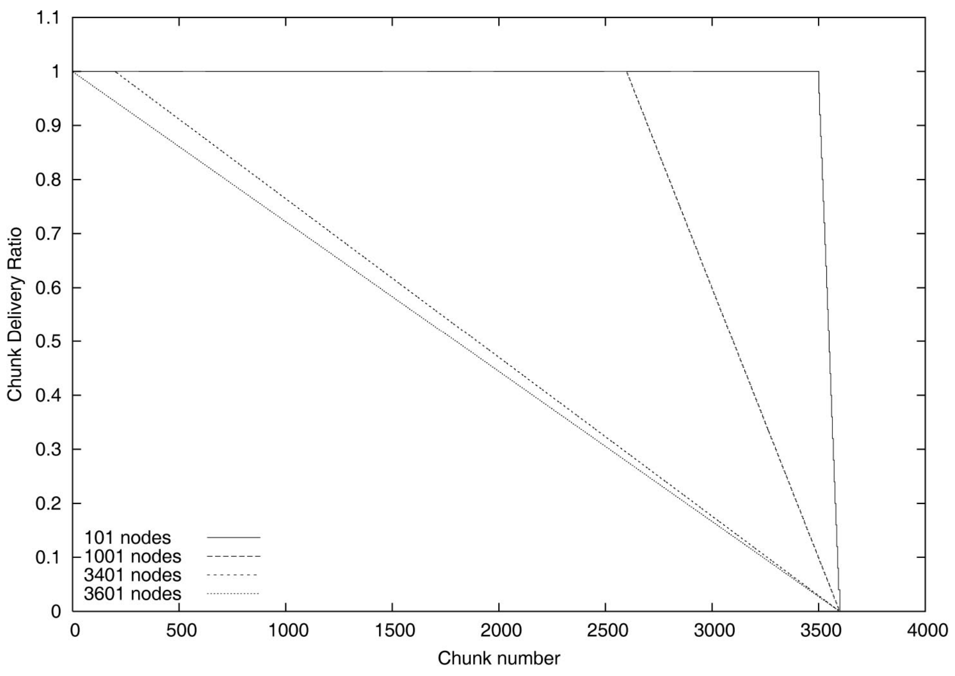

Figure 6 - Chunk delivery ratio

The cumulative distribution of the average chunk delay for a given chunk is illustrated in **Figure 7**.

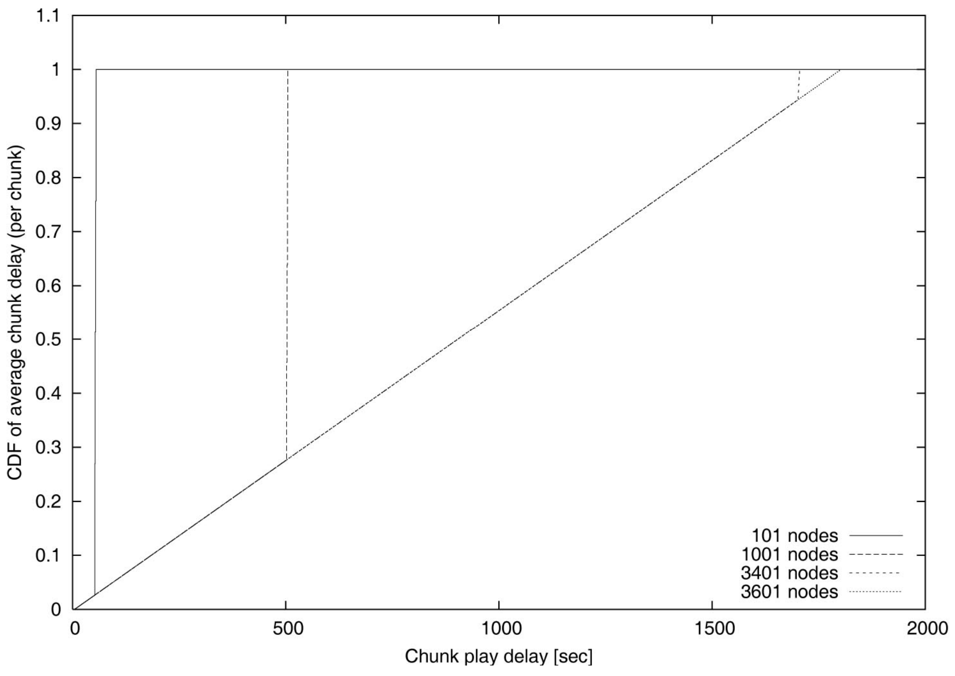

Figure 7 – Cumulative distribution function of the average chunk delay for a given chunk

As it can be seen, each curve is characterized by two different parts: the first one grows almost linearly and accounts for the chunks that are not received by all the nodes within the observation window; the second one relates to the chunks that all the nodes receive within the observation window and it is almost vertical (actually there are S different values separated by 1 second as the average chunk delay is periodic with period *S* and the values it may take increase by step T=1/R).

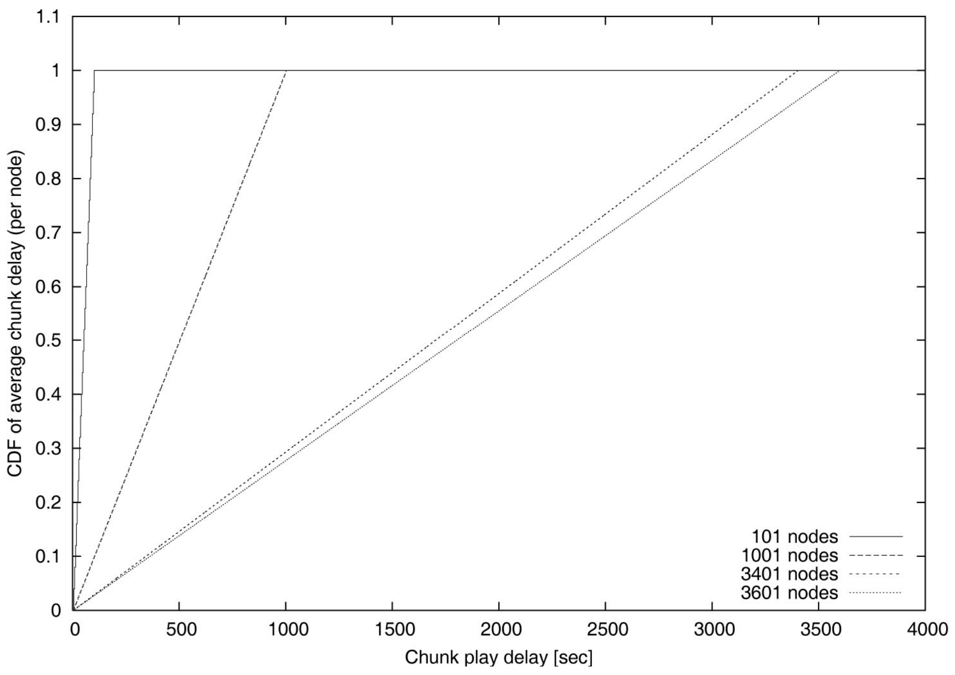

Figure 8 - Cumulative distribution function of average chunk delay at a given level

Figure 8 shows the cumulative distribution function of average chunk delay at a given level. The reported curves exhibit a step trend with each step corresponding to a specific level, even if, as the node number and consequently the level number grow, it is difficult to appreciate the step trend. This complies with equation (16), even if the last one has been derived under the approximation of observed chunk number multiple of S.

4.3 Simulator performance

Just to give an idea about the OPSS performance, we provide some details about the simulation computational load on the pc-desktop we used for our experiments. Specifically, the pc was equipped with a 3.2 GHz bi-processor CPU and 4 Gigabyte RAM. The table below refers to the implemented binary tree algorithm and shows, for a given node number, the time necessary to complete the simulation, the resulting log file size and the time necessary to analyze log file with an our own C++ application. The analysis results have been then used to achieve the previously reported graphs.

Tree			
Nodes	Simulation Time	Log file size	Analysis time
131071	~6h	15,6GB	~5h

Table 1 Main computational load parameters

5. Conclusion

In this paper we have presented the OPSS simulator and tested it using two reference algorithms for building a live streaming p2p distribution systems. The simulator has produced consistent results, perfectly aligned with the analytical models of the reference algorithm. Our current works concern first the extensions to the network part of the simulator, in order to consider more complex topology with respect to the "bottleneck-in-the-access" approach that is currently implemented. The second direction of work is to use OPSS for performance evaluation of mesh based algorithms for a live streaming p2p distribution systems. As we have shown that there are no tools allowing to make this evaluation for large scale peer to peer system considering the problem of resources utilization at network transport level, we believe that OPSS will be able to provide interesting insight on the problem.

6. References

[1] C. Diot, B. Levine, B. Lyles, H. Kassem, D. Balensiefen, *Deployment issues for the IP multicast service and architecture*, in IEEE Network, vol. 14 (1), 10-20, 2000

[2] Y. Chu, S. G. Rao, H. Zhang, *A case for end system multicast*, in Proceedings of ACM SIGMETRICS 2000, Santa Clara, CA,USA, 2000

[3] B. Zhang, S. Jamin, L. Zhang, *Host multicast: a framework for delivering multicast to end users*, in Proceeding of IEEE INFOCOM, New York, NY, USA, 2002

[4] S. Banerjee, B. Bhattacharjee, C. Kommareddy, *Scalable application layer multicast*, in Proceedings of ACM SIGCOMM, Pittsburgh, PA, USA, 2002

[5] D. A. Tran, K. A. Hua, T. Do, *ZIGZAG: an efficient peer-to-peer scheme for media streaming*, in Proceedings of IEEE INFOCOM, San Francisco, CA, USA, 2003

[6] V. N. Padmanabhan, H. J. Wang, P. A. Chou, *Distributing streaming media content using cooperative networking*, in Proceedings of NOSSDAV, Miami Beach, FL, USA, 2002

[7] M. Castro, P. Druschel, A. Kermarrec, A. Nandi, A. Rowstron, A. Singh, *SplitStream: high-bandwidth multicast in cooperative environments*, in Proceedings of the 19th ACM Symposium on Operating Systems Principles (SOSP'03), The Sagamore, Bolton Landing, NY, USA, 2003

[8] X. Zhang, J.C. Liu, B. Li, P. Yum, *CoolStreaming/DONet: A data-driven overlay network for efficient live media streaming*, In Proceedings of IEEE INFOCOM, Miami, FL, USA, 2005

[9] M. Zhang, L. Zhao, Y. Tang, J. Luo, S. Yang, *Large-Scale Live Media Streaming over Peer-to-Peer Networks through Global Internet*, in Proceedings of ACM Multimedia 2005, Singapore, Singapore, 2005

[10] http://netgroup.uniroma2.it/twiki/bin/view.cgi/Netgroup/OpssPublicPage

[11] A.J. Ganesh, A.M. Kermarrec, L. Massoulie, *Peer-to-peer membership management for gossip-based protocols*, in IEEE Transactions on Computers, 52(2), 2003

[12] PlanetLab web site, http://www.planet-lab.org/

[13] B. Chun, D. Culler, T. Roscoe, A. Bavier, L. Peterson, M. Wawrzoniak, M. Bowman, *PlanetLab: An Overlay Testbed for Broad-Coverage Services*, in ACM Computer Communications Review, vol. 33, no. 3, 2003

[14] M. Zhang, Y. Xiong,, Q. Zhang, S. Yang, *On the Optimal Scheduling for Media Streaming in Data-driven Overlay Networks*, in Proceedings of IEEE Globecom 2006, San Francisco, CA, USA, 2006

[15] V. Pai, K. Kumar, K. Tamilmani, V. Sambamurthy, A.E. Mohr, *Chainsaw: Eliminating Trees from Overlay Multicast*, in Proceedings of 4th International Workshop on Peer-to-Peer Systems (IPTPS'05), Ithaca, NY, USA, 2005

[16] *PPLive web site*, http://www.pplive.com

[17] *SopCast web site,* http://www.sopcast.org/

[18] X. Hei, C. Liang, J. Liang, Y. Liu, K. Ross., *Insights into PPLive: A Measurement Study of a Large-Scale P2P IPTV System*, in Proceedings of IPTV Workshop, International World Wide Web Conference, Edinburgh, Scotland, 2006

[19] S. Ali, A. Mathur, H. Zhang, *Measurement of Commercial Peer-To-Peer Live Video Streaming*, in Proceedings of Workshop on Recent Advances in P2P Streaming, Waterloo, ON, Canada, 2006.

[20] K. Calvert, E. Zegura, S. Bhattacharjee, *How to Model an Internetwork*, in Proceedings of IEEE INFOCOM, San Francisco, CA, USA, 1996

[21] *Aurora simulator*, http://cvs.sourceforge.net/viewcvs.py/freenet/aurora/

[22] *Serapis simulator*, http://cvs.sourceforge.net/viewcvs.py/freenet/Serapis/

[23] T. M. Gil, F. Kaashoek, J. Li, R. Morris, J. Stribling, *P2Psim simulator*, http://pdos.csail.mit.edu/p2psim/

[24] Rice University, *FreePastry simulator*, http://freepastry.org/FreePastry/download.html

[25] I. Stoica, M. Walfish, *Chord simulator*, http://cvs.pdos.csail.mit.edu/cvs/~checkout~/sfsnet/simulator/

[26] Neurogrid simulator, http://www.neurogrid.net/php/simulation.php

[27] S. Joseph, *An extendible open source P2P simulator*, P2P Journal, 1-15, 2003

[28] N. S. Ting, R. Deters, *3LS - A Peer-to-Peer network simulator*, in Proceedings of 3rd International Conference on Peer-to-Peer Computing, Linköping, Sweden, 2003

[29] M. Jelasity, G. P. Jesi, A. Montresor, S. Voulgaris, *PeerSim simulator*, http://peersim.sourceforge.net/}

[30] M. Jelasity, A. Montresor, O. Babaoglu, *A modular paradigm for building self-organizing peer-to-peer applications*, in Proceedings of the International Workshop on Engineering Self-Organising Applications (ESOA'03), Melbourne, Australia, 2003

[31] Q. He, M. Ammar, G. Riley, H. Raj, R. Fujimoto, *GnutellaSim simulator*, http://www-static.cc.gatech.edu/computing/compass/gnutella/

[32] Q. He, M. Ammar, G. Riley, H. Raj, R. Fujimoto, *Mapping peer behavior to packet-level details: a framework for packet-level simulation of Peer-to-Peer systems*, in Proceedings of MASCOTS 2003, Orlando (FL), USA, 2003

[33] The Network Simulator ns-2, http://www.isi.edu/nsnam/ns/

[34] M. Schlosser, T. Condie, S. Kamvar, *Simulating a P2P file-sharing network*, in Proceedings of 1st Workshop on Semantics in P2P and Grid Computing, Budapest, Hungary, May 2003

[35] D.Qiu, R.Srikant, *Modeling and Performance Analysis of BitTorrent-Like Peer-to-Peer Networks*, in Proceedings of ACM SIGCOMM'04, Portland, OR (USA), September 2004

[36] M.Baker, T. Giuli, *Narses: a scalable flow-based network simulator*, Technical report, Stanford University, 2002, available on line at http://arxiv.org/PS_cache/cs/pdf/0211/0211024.pdf

[37] D. Bertsekas, R. Gallager, *Data Networks*, Prentice Hall, Englewood Cliffs, NJ, 1987

[38] F. Lo Piccolo, G. Bianchi, S. Cassella, *Efficient simulation of bandwidth allocation dynamics in P2P Networks,* in Proceedings of Globecom 2006, San Francisco, CA, USA, 2006