OPSS: an Overlay Peer-to-peer Streaming Simulator for large-scale networks

Lorenzo Bracciale Francesca Lo Piccolo Dario Luzzi Stefano Salsano [1]

Abstract

In this paper we present OPSS, an Overlay Peer-to-peer Streaming Simulator designed to simulate large scale (i.e. in the order of 100K nodes) peer-to-peer streaming systems. OPSS is able to simulate a fair (i.e. ”TCP-like”) sharing of the uplink and downlink bandwidth among different connections, and it guarantees extensibility by allowing the implementation of different peer-to-peer streaming algorithms as separate modules. Source code of OPSS is available under the GPL license.

1 Introduction

Three different approaches have been used up to now to evaluate performance of P2P streaming systems. Measurement-based studies [1] have been carried out to analyze applications, such as PPLive, which are widely deployed but whose implementation details are not under public domain. The experimental testbed PlanetLab [2][3] has been used to evaluate performance of unstructured P2P streaming systems like CoolStreaming/DONet [4] and GridMedia [5]. Packet-level simulators have also been implemented, as it has been described in [6]. It is possible to identify different drawbacks in the previous approaches. Indeed, measurement-based studies do not allow to consider different alternatives and to evaluate performance in advance of building and deploying a system, while experimental testbeds and the currently available simulation tools suffer from scalability problems for different reasons. On the one hand, experimental testbeds would require a large network of emulator nodes, which is not easy to realize and to manage. Due to this, the experimental results achieved by PlanetLab and presented in [4] and [5] relate to networks whose size is at most respectively 200 and 340 nodes. On the other hand, even if a large number of P2P simulators has recently emerged, either they mainly focus on simulating the resource search phase and neglect the content distribution phase, as it happens for instance in P2Psim [7] and Peersim [8], or they provide a very detailed packet-level simulation model of the underlying transport network, as GnutellaSim simulator [9]. Moreover, the packet-level approach compromises the scalability of the resulting simulation, as only a few hundreds or thousands nodes may be properly simulated in reasonable time with such a level of details. The authors [6] propose in fact simulation results concerning networks of at most 2048 nodes.

[1]University of Rome ”Tor Vergata”, Rome, Italy.

On basis of the above observations, in this paper we aim at presenting OPSS[10], Overlay Peer-to-peer Streaming Simulator, a new simulative approach that makes P2P video streaming performance evaluation scalable. Specifically the paper is organized as follows. In section 2 we will describe the key principles which OPSS is built on by focusing on the simulation scalability aspects. In section 3 we will provide some implementation details. In section 4 we will evaluate OPSS performance in terms of both scalability and capability of producing correct results.

2 OPSS: key concepts and scalability aspects

OPSS is a discrete-event fluid-flow simulator. It allows to simulate the data distribution at the flow level, i.e. neglecting transmissions of single packets and 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, and consequently it improves the simulation scalability, while retaining a satisfactory accuracy in the model of the data delivery process. We also assume that all active connections share fairly the available transmission resources, as it happens if peer nodes use TCP as transport protocol and round trip times are of the same order of magnitude. These assumptions justify the meaning of ”TCP-like” sharing of the uplink and downlink bandwidth among different connections. Under this hypothesis it is possible to use a max-min fair [11] rate allocation algorithm in order to evaluate the available capacity for each connection, given the link bandwidth constraints.

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. In fact, the classical centralized implementation of max-min fair rate allocation does not scale well for the network dimensions of our interest. The implementation suggested in [11] requires in fact 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. However, as it was observed in [12], 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 has been exploited to develop an exact and more efficient max-min fair rate allocation implementation under the assumption of bottleneck links only in the access side of the network. More details about such implementation may be found in [12]. The experimental results in [12] show that the proposed algorithm outperforms traditional max-min computation approaches by as much as a factor 100 for a million nodes network.

We have developed OPSS starting from the implementation of the max-min fair rate allocation algorithm proposed in [12]. As in [12] we made the assumption that rate bottlenecks occur only in the access part of the network. Obviously, the most serious limit in our approach is that the max-min fair bandwidth allocation well approximates a steady state TCP-like 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 [12] to the case of generic topologies, even if not trivial, could be a reasonable solution.

3 Implementation details

OPSS is written in C++ and is publicly available [10] under GPL license. It was designed according to a modular implementation logic. Figure 1 illustrates the main blocks of the simulator architecture. It is possible to identify three layers: User, Overlay and Network. User layer represents the peer behavior, taking into account for example connection and disconnection policies (i.e. the "churn" behavior). Overlay layer is responsible to simulate the overlay network and the overlay interactions between peers. Network layer represents the network behavior, and it currently implements the optimized max-min fair rate allocation approach as described in [12]. All the above layers interact with the Engine block, which contains the discrete-event-related classes and mages the event executions. Engine block is also responsible for output 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. While User, Overlay, Network and Engine include the basic structures common to any P2P streaming system, the Algorithms block is responsible of the P2P streaming algorithm and application to be simulated, including the control communication between nodes and the scheduling algorithm of stream segments. It inherits the basic structures of User, Overlay, Network and Engine, and allows to customize them. In such a way, it allows the implementation of any P2P live streaming mechanism. The previously described approach makes OPSS a very flexible simulator, as it offers the possibility of implementing the logic of P2P streaming application as separate module. Moreover, simulator users may exploit different basic classes provided by User, Overlay, Network and Engine blocks and potentially implement any kind of P2P streaming algorithms. For further information about how to write algorithms, please refer to the guide available on the reference site.

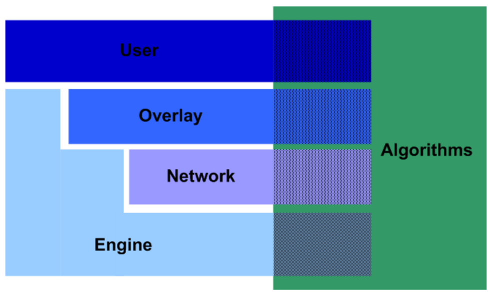

Figure 1: OPSS architecture

4 OPSS performance

We simulated a trivial streaming distribution scheme, with the purpose of easily deriving the analytical model and comparing the experimental results achieved by OPSS with the analytical results. In such a way, we were able to evaluate OPSS correctness. In the same time, an ever increasing number of simulated nodes allowed us to investigate OPSS scalability.

Specifically, we simulated a balanced binary streaming distribution tree. The stream source is the root of the tree. The stream video is divided into segments or chunks. T and $R = 1/T$ denote respectively the chunk duration [s] and the source chunk rate [chunk/s]. Each node downloads chunks from one father, and it uploads chunks to 2 children. All nodes (including the stream source) join the system simultaneously and form the distribution tree. We also assume a static situation, in which all nodes persist through the whole lifetime of the simulation. All nodes are assigned an access link with uplink and downlink capacities W_{up} and W_{down} [chunk/s]. To simplify, we assume symmetrical access links, that means also $W_{up} = W_{down} = W$. As consequence, each node downloads chunks at $W/2$ [chunk/s] from its father in the tree. If $W/2 < R$, 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 $W/2 \geq R$, in which the available portion of the father's uplink capacity is greater than or equal to the rate of the stream to be received.

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 levels are complete, the number of nodes at level l is M^l and the total number of nodes is $\sum_{l=0}^{L} M^l$. In the following, we will always consider trees with complete levels.

Due to the characteristics of the simulated application, we consider performance metrics related to the delay of the re-

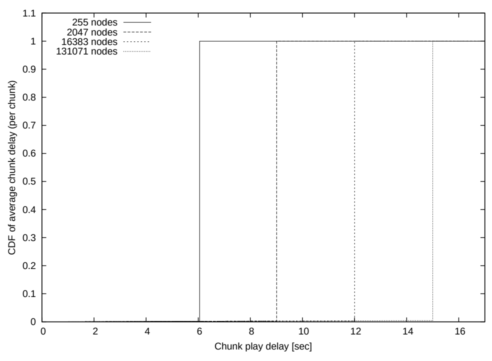

Figure 2: Cumulative distribution function of average chunk delay for a given chunk

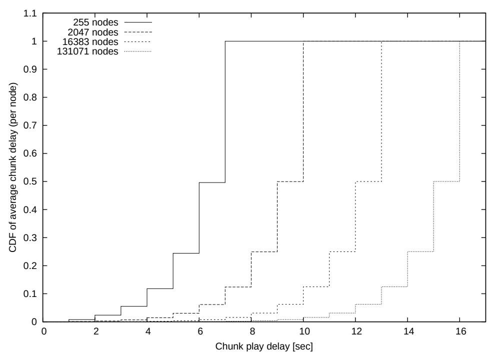

Figure 3: Cumulative distribution function of average chunk delay at a given level

ceived chunks. Specifically, with reference to a chunk c and a node n, we define chunk delay $d(c,n)$ the difference between the time instant at which the download of chunk c is complete at node n and time instant of chunk c generation at node n. 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 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 $d(c,l) = l \times W/2$. Starting from the chunk delay, we consider the average chunk delay for a chunk c, which can be evaluated by averaging the delay over all nodes that received that chunk.

$$\begin{aligned} \overline{d}(c) &= \frac{\sum_{i=1}^{L} M^l \cdot d(c,l)}{\sum_{i=1}^{L} M^l} \\ &= \frac{M\{M^l[L(M-1)-1]+1\}}{W\,(M-1)\,(M^l-1)} \end{aligned} \tag{1}$$

It could be interesting to consider the perspective of a given node n. The average chunk delay perceived by a generic node n at level l is reported in (2). The assumption is that C chunks are generated during the simulation period.

$$\overline{d}^{node}(l) = \frac{1}{C}\sum_{c=1}^{C} d(c,l) = \frac{2}{W}\cdot l \tag{2}$$

With regard to the experimental results, figure 2 and figure 3 show the cumulative distribution function of the average chunk delay for a given chunk and the average chunk delay perceived by the generic node. The presented results refer to binary trees whose the node number ranges from 255 to 131171. There is a perfect matching between the values achieved by equations (1) and (2) and the values achieved by simulation. Indeed, figure 2 confirms in fact that the average chunk delay is constant across the tree levels, while given a number of simulated nodes, the steps in figure 3 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.

References

[1] X. Hei, C. Liang, J. Liang, Y. Liu, K.W. Ross, *Insights into PPLive: A measurement study of a large-scale P2P IPTV system*, in Workshop on Internet Protocol TV (IPTV) Services over World Wide Web, Edinburgh, Scotland, May 2006

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

[3] 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

[4] 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

[5] 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

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

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

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

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

[10] OPSS simulator, `http://minerva.netgroup.uniroma2/p2p`

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

[12] 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