A push-based scheduling algorithm for large scale P2P live streaming

L. Bracciale, F. Lo Piccolo, D. Luzzi, S. Salsano, G. Bianchi, N. Blefari-Melazzi
DIE, University of Rome Tor Vergata
{lorenzo.bracciale,francesca.lopiccolo,dario.luzzi,stefano.salsano,giuseppe.bianchi,blefari}@uniroma2.it

***Abstract*—In this paper, we present a chunk scheduling algorithm for a mesh-based peer-to-peer live streaming system and we evaluate it by simulations over large-scale networks. Literature papers typically design chunk scheduling algorithms by considering the chunk delivery ratio as performance metric. We propose a push-based algorithm, which not only tries to maximize the chunk delivery ratio but it also takes into account and tries to minimize the delivery delay of chunks at the peer nodes. This is an important requirement, when dealing with real-time multimedia flows. Another important contribution of this paper is the design and implementation of a simulator able to evaluate the performance of large scale P2P networks (tens of thousands peers). The importance of this contribution lies in the fact that existing simulators and performance studies handle at most hundreds or few thousands of peers, while real-life P2P streaming systems aim at distributing contents to several hundreds of thousands, if not millions, of users. The performance evaluation study aims at providing a comprehensive view of what performance can be expected for mesh-based peer-to-peer streaming systems, both in terms of chunk delivery ratio and delay, for a large range of the number of users. The individual effect of a variety of system parameters, and especially number of partner nodes in the mesh, constrained link bandwidth, node heterogeneity, and network size, has been analyzed. Our results show that performances of the proposed push-based solution are already quite effective even with severely bandwidth constrained large scale networks.**

I. Introduction

Peer-to-peer (P2P) overlay systems are being recently proposed to stream multimedia audio and video contents from a source to a large number of end users. To this aim, end-hosts auto-organize themselves in an overlay distribution network, by using unicast tunnels among participating overlay nodes. The contents to be distributed are divided in so-called "chunks" and overlay nodes relay such chunks. Early proposals of overlay distribution networks are based on multicast distribution trees, among them NARADA [1], NICE [2], ZIGZAG [3]. However, overlay multicast distribution trees perform well when participating nodes do not move continuously and are always available, but suffer from re-configurability problems when nodes have a high churn rate. To face this issue, latest proposals use overlay mesh-based and unstructured topologies. CoolStreaming/DONet [4], GridMedia [5] and PRIME [6] offer examples of this approach.

In this paper, we focus on overlay unstructured mesh-based P2P streaming systems, and we investigate the problem of how to schedule the distribution of stream chunks between overlay neighbor nodes. We propose a push-based approach, where a supplier node takes the decision of which chunks will be served to which neighbor nodes.

Performance assessment is carried out in terms of both chunk delivery ratio and delivery delay. To design and evaluate scheduling algorithms for P2P streaming systems, the most used performance metric is indeed the chunk delivery ratio (also called continuity index, see e.g., [4], [7]), which is defined as the ratio between the number of chunks arrived at destination before the playback deadline and the total number of streamed chunks. This parameter can measure the continuity of the playback; however, little attention has been provided to delay figures. The authors of [5] do mention absolute delay as a possible performance metric. Nevertheless, they do not exploit this metric in their scheduling algorithms, presented in [7].

Another important contribution of this paper is the performance evaluation carried out through a suitably adapted custom-made C++ simulator called OPSS [8][9], capable of evaluating the performance of large scale P2P networks (tens of thousands peers). The importance of this contribution lies in the fact that existing simulators and performance studies handle at most hundreds or a few thousands of peers, while real-life P2P streaming systems aim at distributing contents to several tens of thousands, if not even several hundred thousand or more users. Thus, the availability of this tool is important to test the scalability of proposed streaming systems. Our simulator assumes that network traffic can be modeled as a fluid flow, that bottlenecks are in the access links, and that the access uplink and downlink bandwidth is shared between competing flows according to the classical max-min fairness notion [10]. In more detail, OPSS achieves scalability by exploiting an efficient implementation of the max-min fair bandwidth allocation [11] specifically conceived for the above simplifying assumptions.

The paper is organized as follows. In section II we review the state of art. We describe our scheduling algorithm in section III. Section IV presents the evaluation scenario, including the performance metrics used to evaluate the proposed solution. Performance evaluation is carried out through simulation in sections V and VI. Finally, section VII concludes the paper.

II. Related work

In this section we give a brief overview of the main P2P live media streaming systems built on mesh-based overlay topologies and we review the already available results on

the problem of the stream segment scheduling. In doing this, we will focus on the way the performance of such systems has been evaluated with particular regard to the performance figures and the size of the investigated networks.

CoolStreaming/DONet [4] is a live media streaming system which constructs a random overlay mesh to distribute the stream segments between the participating overlay nodes. Chunk availability in the node buffer is represented by a Buffer Map (BM), where a bit 1 and 0 indicate that a segment is respectively available or unavailable. Each node learns about chunk availability by periodically exchanging its BM with the BMs of its partners, which are the neighbors in the overlay mesh. DONet is built on a pull approach, i.e. scheduling decisions are taken at receivers and chunk transmissions start only if a receiver requests that chunk from a supplier neighbor. Specifically, the proposed heuristic scheduling algorithm gives priority to the chunks with more stringent playback deadline and to supplier neighbors with the highest bandwidth. So the algorithm calculates the number of potential suppliers for each segment and, starting from the segments with only one potential supplier, it selects the supplier with the highest bandwidth and enough available time in case of multiple suppliers. DONet performance has been evaluated by 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. The number of used PlanetLab nodes ranges from 10 to 200 (passing through 50, 100, 150), while the number of partners ranges from 2 to 6. Control overhead and 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.

GridMedia [5] is an 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 in push mode (e.g. without explicit request) in the immediate following phase. PlanetLab testbed is used to evaluate GridMedia performance in [5]. 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) the 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, that is the average ratio between the control traffic and the total traffic at each node.

The same authors as [5] focus in [7] 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 packet level simulator. The results reported in [7] refer to a data driven overlay network of up to 1000 nodes. The main 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.

In PRIME [6] participating peers form a randomly connected and directed mesh, where all connections are congestion controlled. The incoming and outgoing degrees of individual peers are determined by maximizing the utilization of the incoming and outgoing access link bandwidth. The content is encoded with Multiple Description Coding (MDC) which enables each peer to maximize the delivered quality by pulling a proper number of descriptions. With regard to the content delivery mechanism, PRIME combines push advertisements by parents with pull requests by child peers. The packet scheduling mechanism at child peers selects the packets to be requested according to a global pattern of content delivery that minimizes the probability of content bottleneck among peers. Such pattern consists of the diffusion and the swarming phases. The diffusion phase relates to the new stream segments that have been advertised by parents during the last scheduling event. The swarming phase relates to the packets that have already been received and are within the playout buffer. Performance of PRIME has been evaluated via packet-level *ns* [14] simulations. The number of simulated nodes ranges from 100 to 500. Performance metrics related to delivered quality, content bottleneck occurrence during the diffusion and swarming phase, bandwidth bottleneck, playout buffer capacity and average path length of delivered packets are presented.

III. Chunk Scheduling Algorithm

The operation of a mesh-based, peer-to-peer, live streaming system can be conceptually decomposed in two tasks: "overlay mesh building" and "chunk scheduling", which are typically realized by two distinct and distributed algorithms. The "overlay mesh building" algorithm is used by peer nodes to select a set of neighbor peers. The aim is to build a mesh of peers to be used to exchange control information (e.g., information related to chunk availability and chunk download requests). This control information is exploited by the "chunk scheduling" algorithm, whose aim is to define a set of independent decisions to be taken by peers to select which chunk to download from which neighbor peer (or, conversely, which chunk to upload to which neighbor peer).

In this work, we focus on the chunk scheduling algorithm, assuming that the overlay mesh is already in place. Also, we limit our analysis to a static situation, where peers do not join or leave the system. Thus, the impact of peer churn is left for further study.

The algorithm is based on a push approach, where scheduling decisions are taken at supplier nodes. To be efficient, our algorithm uploads each received chunk to the higher number of neighbors possible, trying to serve each chunk to at least one neighbor. The sending node measures the chunk transfer time to understand if it has to increase, maintain or decrease the number of neighbors to which it is uploading simultaneously the chunks. Moreover, the sending node tries dynamically to identify the neighbor nodes that minimize the waste of its uplink bandwidth, and it prefers them to the other ones. This is achieved by performing downlink bandwidth estimations of all neighbors.

We make the following assumptions: the bit rate of the stream is constant, thus the chunk duration is always equal to T; the generic node stores received chunks in a FIFO queue; chunks are numbered according to the order in which they are received (thus, chunk $\#j$ is the j-th received chunk and not necessarily the j-th chunk of the stream) and are served in the same order as they are received.

The algorithm proceeds by steps, described as follows. At the i-th step the supplier node n takes a decision on:

1) which chunk $c(i)$ to upload, chosen among the received ones;
2) how many neighbors $N(i)$ to upload simultaneously the chunk to;
3) which neighbors $\{p_1, p_2, ..., p_{N(i)}\}$ to upload the chunk to.

Each step is dedicated to the upload of a single chunk. The duration of the i-th step is equal to the time required to upload chunk $c(i)$ to the $N(i)$ selected neighbors.

Let us now explain how these three decisions are taken.

1) The chunk to upload in step i, $c(i)$, is the first chunk in queue; if the queue is empty, the chunk $c(i)$ is again that of step $(i-1)$ if not all neighbor nodes received that chunk, otherwise node n will stay idle. The initial condition is $c(1) = 1$;
2) The number of neighbors $N(i)$ to upload simultaneously chunk $c(i)$ is computed as follows:
   a. $N(i) = N(i-1) - 1$ if the total transmission time at step $(i-1)$ lasted more than T seconds and the chunk queue is not empty;
   b. $N(i) = N(i-1) + 1$ if the total transmission time at step $(i-1)$ lasted less than T seconds and the chunk queue is empty;
   c. $N(i) = N(i-1)$ in all other cases.

   The initial condition is $N(1) = 1$.
3) The neighbors $\{p_1, p_2, ..., p_{N(i)}\}$ to upload chunk $c(i)$ are selected within a proper subset of neighbors. First of all, such subset has to include neighbors interested in chunk $c(i)$. To this end, when a node n starts to upload chunk $\#j$, it advertises the availability of chunk $\#(j+1)$, that follows chunk $\#j$ in the FIFO queue[1]. All neighbors that receive the advertisement of a given chunk and lack the chunk will send a "chunk missing indication" to node n. Thus node n may exploit all received "chunk missing indications" relative to chunk $c(i)$ to identify the subset of interested neighbors. Secondly, if the subset of interested neighbors has more elements than $N(i)$, node n tries to identify neighbors that minimize the waste of its uploading bandwidth. For this purpose, it sorts its neighbors into "classes" on the basis of bandwidth estimations performed at the end of each upload step. Specifically, at step i it shall estimate its uplink bandwidth $U_n(i)$ as:

$$\max\left(U_n(i-1), \frac{C_{\text{first delivery}}}{T_{\text{first delivery}}}\right)$$

where $T_{\text{first delivery}}$ is the time of completion of the first delivered chunk $c(i)$ (in the case chunk $c(i)$ is delivered to multiple nodes) and $C_{\text{first delivery}}$ are the overall bytes which node n delivers to all its neighbors up to $T_{\text{first delivery}}$. Node n shall also estimate the downlink bandwidth $D_{n_k}(i)$ of neighbor n_k ($k = 1, 2, ..., M$ and M number of overlay neighbors per node) as

$$\frac{C_{\text{n}_\text{k},\text{first delivery}}}{T_{\text{first delivery}}}$$

where $C_{\text{n}_\text{k},\text{first delivery}}$ are the bytes transmitted to neighbor n_k up to $T_{\text{first delivery}}$. Then node n classifies neighbor node n_k in class 1 if D_{n_k} is greater or equal than U_n, while in class $h > 1$ if D_{n_k} is greater or equal than U_n/h but lower than $U_n/(h-1)$.[2] In such a way, a node in class h may be effectively exploited only if the degree of parallelism of a chunk delivery is greater or equal than h (i.e. a node in class 2 can be selected for effective delivery of two or more chunks in parallel and can not be selected alone for a chunk delivery). As consequence, once a given $N(i)$ is estimated for the i-th step, the uploading node n selects a subset of its neighbors interested in chunk $c(i)$ preferring the ones belonging to classes minor or equal to $N(i)$.
If the subset of neighbor nodes interested in chunk $c(i)$ and belonging to classes minor or equal to $N(i)$ contains more elements than $N(i)$, the supplier node either i) selects the $N(i)$ neighbors randomly, or ii) selects the $N(i)$ neighbors with the biggest value of

[1]The rationale is to advertise the availability of a given chunk in proximity of the corresponding upload step/steps. However, to guarantee at least one advertisement per chunk, if a chunk is going to be served and it has never been advertised, the uploading node advertises the chunk before uploading it.

[2]These estimates being straightforward starting from the already available estimate $B_{n_k}(i)$ and downgrading the neighbor node class (initially set by default to class 1) whenever bandwidth throttling is experienced. However technical details are quite lengthy and here omitted to save space. It suffices to add that, unlike the estimate $B_n(k)$, the neighbor node class estimate is periodically refreshed to avoid maintaining a temporary downlink bandwidth reduction (due to e.g., multiple parallel downloads experienced by the considered neighbor node) forever.

uplink bandwidth. By iterating the latter procedure, and thus having the nodes with greatest branching degree closest to the tree root (the source of the stream) an optimal (shortest) distribution tree would be readily formed. From an implementation stand-point, each node signals to its neighbors its updated uplink bandwidth estimation (independently carried out by each node as described above). To avoid selecting the same neighbor node (and thus reducing the randomness of the chunk distribution trees) even in the case of marginal differences in the estimated uplink bandwidth, a tolerance margin is introduced ($\alpha = 0.95$) so that two uplink bandwidth estimates B_a and B_b with $B_a > B_b$ are considered equal if $B_b > \alpha B_a$.

Finally, as the decisions to schedule chunk upload are taken independently at sending nodes, nodes that share a neighbor could upload the same chunk to such neighbor. To solve this problem, a node receiving more than one copy of the same chunk issues "cancel" signalling commands, thus maintaining only one upload, which is randomly chosen.

Figure 1 shows how decisions 1), 2) and 3) are taken when node A has chunks x and y in its FIFO queue and it has to serve neighbors B, C and D. The figure illustrates also the advertisement, chunk missing and cancel signalling messages.

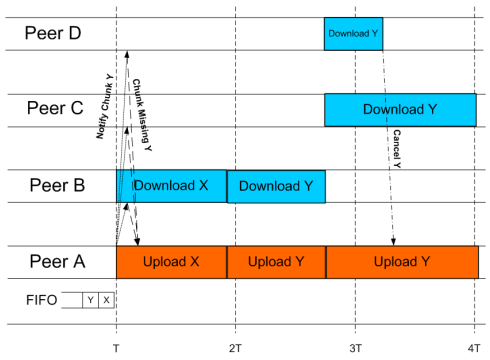

Fig. 1. The proposed algorithm with specific reference to decisions 1), 2) and 3) and signalling aspects.

IV. Evaluation scenario

OPSS simulator [8][9] was used to assess the performance of the proposed scheduling algorithm. Specifically, due to the assumption about bottlenecks only in the access network, we had no need to generate the physical topology. We only generated an unstructured random overlay connected topology, where each node is connected with M randomly selected neighbors. Connections are bidirectional. This means that each neighbor can be used either for uploading chunks to or for downloading chunks from. Each overlay/physical link is assigned a random end-to-end delay. According to the analysis in [15], we consider a Gamma distribution for the end-to-end delay. The Gamma parameters were set in such a way that $\mu = 80$ msec and $\sigma/\mu = 0.1$. Introducing end-to-end delays allowed us to assume that signalling messages take exactly the end-to-end delay to cross the link between sender and receiver.

We simulated two different scenarios in terms of access link bandwidth. The first one will be referred to as homogeneous scenario, because all nodes are assigned the same access link uplink bandwidth including the stream source. In particular, if U and R denote respectively the uplink bandwidth and the stream bit rate (in Kbps), we can characterize the homogeneous scenario in terms of the ratio U/R, which will be referred to as *uplink factor*. The second one will be referred to as heterogeneous scenario, because it includes three different categories of ADSL access links, as reported in table I. The uplink bandwidth of the stream source is 640 Kbps, while the stream bit rate is $R = 300$ Kbps.

Profile	% of nodes	Uplink	Downlink
Origin		640	0
Low	20%	256	1500
Medium	70%	640	4000
High	10%	1000	12000

TABLE I
BANDWIDTH PROFILE

Chunks are generated at the stream source starting from the time instant $t = 1$ sec, and each chunk contains $T = 1$ sec of stream content. Each simulation was run for 1800 seconds. However, in order to avoid to capture also the effects of the initial and final transient, the simulation results we report in sections V and VI refer to the simulation events relative to the chunks generated in the timing interval [100,1600] sec.

For each simulation, we focused on the following performance metrics.

Chunk delivery ratio - As section II shows, this is the most common performance metric used in the literature for real time P2P streaming systems. Specifically, with reference to a given node, the chunk delivery ratio represents the ratio between the number of (completely) received chunks and the total number of generated chunks. Note that we assume that the playout buffer at receivers has infinite capacity. After evaluating the chunk delivery ratio corresponding to each node, for each possible value x of the chunk delivery ratio we calculate the fraction of nodes which have experienced a chunk delivery ratio at least x. We will call such metric inverse cumulative distribution of chunk delivery ratio. The rationale is that it represents the 1's complement of the cumulative distribution of chunk delivery ratio across chunks.

Chunk delivery delay - Due to the characteristic of the application (streaming of real-time multimedia flows), the chunk delivery ratio evaluation needs to be complemented by metrics related to the chunk delivery delay. With reference to node n and chunk c, we define the delivery delay $d(c, n)$ as the difference between the time instant at which chunk c is generated at the source and the time instant at which the chunk c is completely received from node n. Such delay accounts not only for the time necessary for the actual transfer of the chunk (dependent on the connection rate assigned according to the max-min fair allocation), but also for the end-to-end delay. With reference to the generic node n, we

then calculate the average delivery delay by averaging on the received chunks. Thus we evaluate both the distribution of the average delivery delay across all nodes and the distribution of the 0.99-percentile of the average delivery delay. Note that due to the assumption of infinite capacity of the playout buffer, we take all downloaded chunks into account to evaluate the distribution of both the average delivery delays and the 0.99-percentiles of average delivery delays. The latter allows to identify a posteriori the proper value of the playout buffer capacity that guarantees the desired chunk delivery ratio[3].

V. Performance Evaluation - Homogeneous Scenario

Results in this section are obtained for the case of homogeneous nodes. Partners to upload chunks to are selected randomly. Two mesh scenarios are considered: the case of each node connected to 3 neighbors, and the case of 9 neighbors. Three different values for the uplink factor are considered: $U/R = 2, 1.5$, and 1.25. This choice for the considered uplink factors allows us to derive results for a "bandwidth-constrained" network, where the capacity of the chunk distribution network is limited to at most a factor of 2. In the following results, in order to focus on the effects of a single parameter, we further assume that the downlink bandwidth does not contribute in further constraining the available connection bandwidth[4].

Figure 2 shows the inverse cumulative distribution of the chunk delivery ratio for a network composed of 10.000 streaming peers. We remark that this figure is computed on a per-node basis, meaning that statistics have been collected separately for each node, i.e. as perceived by each peer node, and not averaged on different nodes (this implies that "unlucky" nodes, for instance due their topological placement, will explicitly result in the figure as nodes for which the CDR is small). The figure shows that i) for a fairly large amount of neighbors (i.e. 9), there is virtually no node with a CDR lower than 98.5% even in the highly constrained $U/R = 1.25$ case; ii) for $U/R = 2$, from the raw numerical results used in the figure it is possible to conclude that 99.9% of the nodes experience a CDR greater than 99.75% for the case of 9 neighbor nodes, and 99.5% for the case of 3 neighbor nodes, and iii) the more dense the topology (number of neighbors), the better the experienced CDR. This latter conclusion is quite interesting as it shows that the availability of a large number of neighbors is an important parameter to guarantee limited loss in the distribution of the chunks. Figure 3 reports the distribution of chunk delivery delay. A straightforward consideration that is confirmed by the figure is that by increasing the uplink factor, the delay

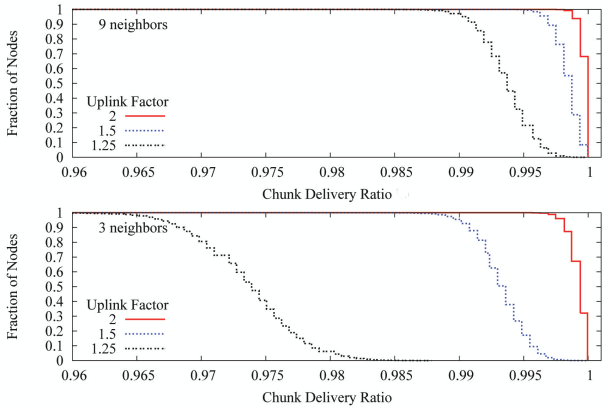

Fig. 2. Inverse cumulative distribution of chunk delivery ratio for varying uplink factor $U/R = 2, 1.5, 1.25$ and for varying number of neighbors (3, 9). The network size is 10000 nodes.

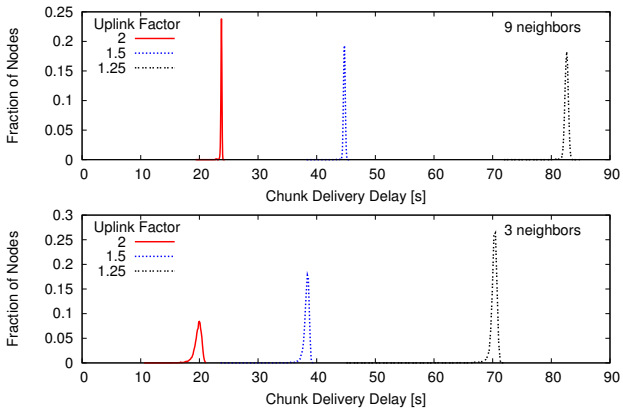

Fig. 3. Distribution of chunk delivery delay for varying uplink factor $U/R = 2, 1.5, 1.25$ and for varying number of neighbors (3, 9). The network size is 10000 nodes.

performance consistently improves. Perhaps less obvious is the comparison of the two cases of 3 and 9 neighbors, as in the latter case the nodes experience an higher delay. Nevertheless, this is easily justified. In fact, on the one side, the number of neighbors do not affect the available bandwidth (in other words, with $U/R = 2$ a node may only serve at most two neighbors during each chunk time, regardless of the number of connected nodes). On the other side, the fact that a node has several neighbors implies that it may randomly choose a node who is, in the same time, downloading the same chunk from another peer (we recall that this is possible as there is an interval of time between signalling completion and actual chunk delivery). The waste of time brought about by canceled downloads turns into an increased chunk delivery delay. Finally, another very interesting remark that can be drawn from figure 3 is that the spread of the average delay as perceived by different nodes is very limited. This is clearly motivated by the exploitation of a random node selection mechanism which statistically makes all the network nodes to experience similar delay performance. This occurs because nodes tend to act as leaves for some chunk (and thus do not use

[3]Note that we have also to consider the chunk losses due to the push mechanism.

[4]Although results in what follows are presented with a "large" - i.e. a factor of 10 - per-node downlink bandwidth to avoid any possibility of chunk delivery throttling by the downlink channels, we have verified via simulation that no notable difference in performance occurs as long as the downlink bandwidth is twice the uplink bandwidth. It is also worth to note that in typical P2P nodes, frequently deployed through asymmetric ADSL connections, it is reasonable to consider the uplink bandwidth as the throttling factor.

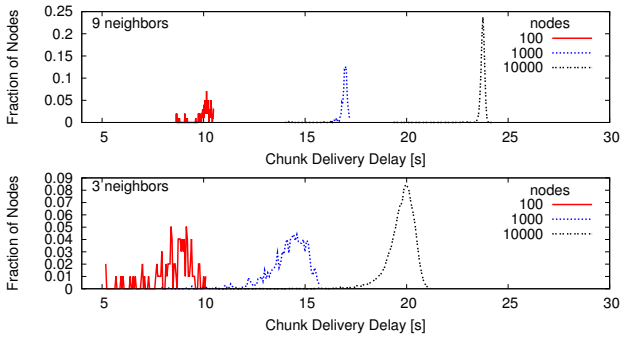

Fig. 4. Distribution of chunk delivery delay for varying network size (100,1000,10000 nodes) and for varying number of neighbors (3,9). The uplink factor is $U/R = 2$.

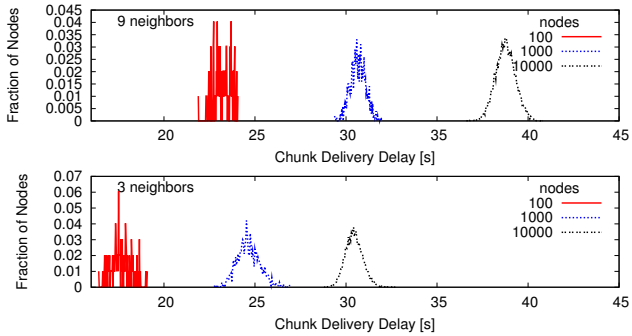

Fig. 5. Distribution of 0.99-percentiles of chunk delivery delay for varying network size (100,1000,10000 nodes) and for varying number of neighbors (3,9). The uplink factor is $U/R = 2$.

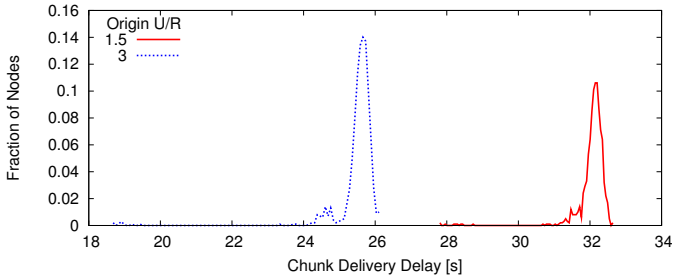

Fig. 6. Impact of increased source node capacity on the distribution of chunk delivery delay. The network size is 1000 nodes, the uplink factor is $U/R = 1.5$ for non source nodes, the number of neighbors is 9.

their uplink bandwidth for said chunk), and as intermediary nodes for other chunks.

Figure 4 reports the distribution of the perceived average delay by each node, for varying network sizes, 100, 1000 and 10000 nodes, and $U/R = 2$. Both cases of neighbor number equal to 3 and 9 are reported. As expected, delay performance clearly degrades with the network size, although the degradation is somewhat limited and appears to follow a logarithmic law with the number of network nodes (this is indeed quite intuitive as the depth of the distribution trees that randomly form clearly follows such a logarithmic law). Figure 5 shows the corresponding results for what concerns the distribution of the 0.99-percentiles of the average chunk delivery delays.

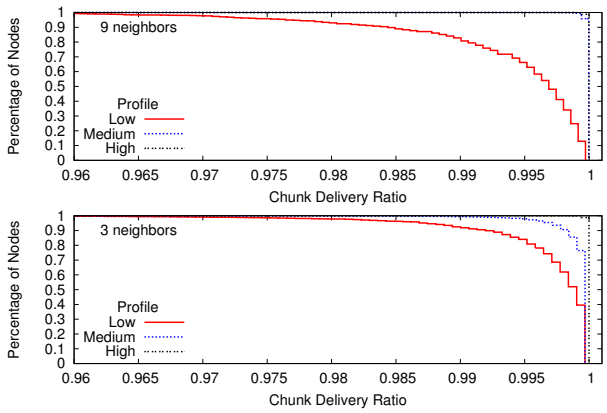

Fig. 7. Inverse cumulative distribution of chunk delivery ratio classified for uplink bandwidth profile (Low, Medium and High) and for varying number of neighbors (3,9) and for network size of 10.000 nodes.

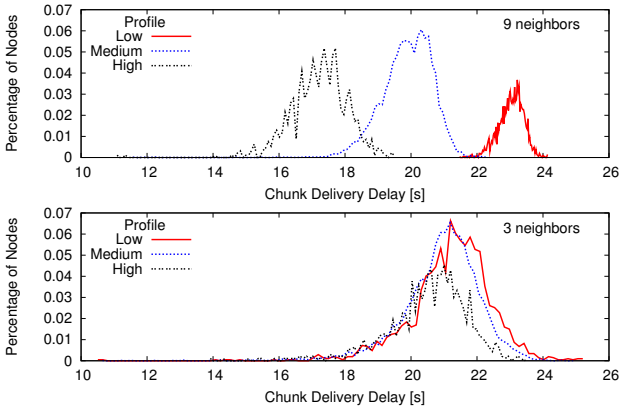

Fig. 8. Distribution of chunk delivery delay classified for uplink bandwidth profile (Low, Medium and High) and for varying number of neighbors (3,9) and for network size of 10.000 nodes.

Finally, while all the above results were obtained with a bandwidth-constrained node as source of the stream, results shown in figure 6 show the beneficial effect of having a source node with increased capacity. The figure shows the case of a homogeneous network composed of 1000 nodes, with $U/R = 1.5$ and 9 neighbors per node, and it compares the case of a source node with the same U/R as all the other nodes versus a source node with double uplink bandwidth, i.e., $U/R = 3$. The figure shows that by just doubling the uplink capacity of the source node, a gain of almost 20% in terms of delay performance is achieved for the whole network.

VI. Performance Evaluation - Heterogeneous Case

Results in this section are obtained for the case of heterogeneous scenario described in section IV.

The ability of the algorithm to select among the possible destinations the nodes which exhibit the largest estimated uplink bandwidth is fundamental to avoid using the low bandwidth nodes as intermediary ones for the chunk distribution. Figure 7, obtained for a network composed of 10000 nodes, indeed shows that, thanks to this ability, the CDR experienced

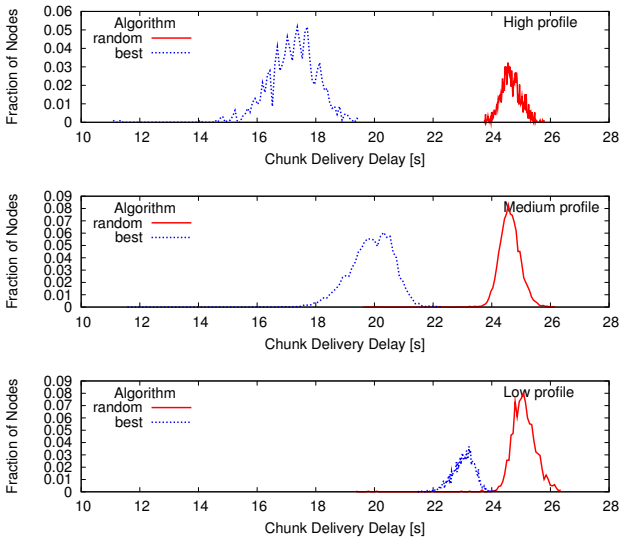

Fig. 9. Comparison of the distribution of the chunk delivery delay classified for uplink bandwidth profile (Low, Medium and High) of the two algorithms for 9 neighbors and for network size of 10.000 nodes.

by nodes belonging to different classes is different, and specifically low bandwidth nodes are the only ones which may exhibit, with 9 neighbors, a CDR lower than 99.9%. Clearly with only 3 neighbor nodes there are less nodes among which the choice of delivering a chunk may be taken, and this turns into a performance impairment also for the Medium class nodes. Note that increasing the number of neighbors improves the CDR of medium profile nodes to the detriment of low profile ones (which are more rarely chosen by neighbor nodes).

Figure 8 provides the corresponding results for what concerns the chunk delivery delay. The positive impact of a greater number of neighbor nodes on the chunk delivery tree formation is clearly envisioned, as chunks are delivered with significantly lower delay to the high bandwidth nodes (and also to the medium bandwidth ones when compared with the low bandwidth nodes). This clearly shows that high (and to a lower extent medium) bandwidth nodes are more frequently chosen as intermediaries in the chunk delivery process.

The benefit of such a choice in terms of overall performance is highlighted in figure 9, which compares the overall chunk delivery delay distribution obtained by the algorithm choosing the ”best” neighbor node(s) versus an algorithm who randomly chooses the neighbors to serve. It shows clearly that the benefit of choosing to upload chunks to nodes with the higher uplink bandwidth leads not only to better performance perceived by high and medium profile nodes with respect to the random choice algorithm, which is obvious, but surprisingly also to smaller average chunk delivery delay values for low profile nodes.

VII. Conclusion

In this paper we propose a push-based scheduling algorithm for P2P mesh-based overlay streaming systems. Performance assessment of the presented solution is based on OPSS, a discrete event fluid flow simulator for the simulation of large scale P2P streaming networks. OPSS allowed to produce simulation results relative to networks of up to 10000 nodes, whereas experimental results relative to networks of up to 1000 nodes have been proposed up to now. Moreover, differently from the most of literature works, which focus on the chunk delivery ratio as performance metric, we also consider the chunk delivery delay as metric somehow representative of the absolute delay with respect to the stream source. We analyzed the individual effect of a variety of system parameters, such as the number of neighbor nodes, constrained link bandwidth, node heterogeneity or network size. The achieved results show that performance is already quite effective even with severely bandwidth constrained large scale networks. Future work will be focused on analyzing the effect of the peer churn on the performance of the proposed scheduling strategy.

References

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

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

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

[4] X. Zhang, J.C. Liu, B. Li and 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 and 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] N. Magharei and R. Rejaie, *PRIME: Peer-to-Peer Receiver-drIven MEsh-based Streaming*, in Proceedings of IEEE INFOCOM, Anchorage, Alaska, USA, 2007.

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

[8] OPSS simulator, http://netgroup.uniroma2.it/twiki/bin/view.cgi/Netgroup/OpssPublicPage.

[9] L. Bracciale, F. Lo Piccolo, D. Luzzi and S. Salsano *OPSS: an Overlay Peer-to-peer Streaming Simulator for large-scale networks*, to be appear in ACM Performance Evaluation Review and available at http://netgroup.uniroma2.it/twiki/bin/viewfile.cgi/Netgroup/OpssPublicPage?rev=1;filename=opss_acm_perf.pdf.

[10] D. Bertsekas and R. Gallager, Data networks, Prentice-Hall, Inc., Upper Saddle River, NJ, 1987

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

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

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

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

[15] C.J. Bovy, H.T. Mertodimedjo, G. Hooghiemstra, H. Uijterwaal and P. Van Mieghem, *Analysis of End-to-end Delay Measurements in Internet*, in Proceedings of the Passive and Active Measurements Workshop (PAM2002), Fort Collins, CO, USA, 2002 (on line available at http://www.ripe.net/projects/ttm/Documents/PAM2002_TUD.pdf).