

A Bandwidth-Aware Scheduling Strategy for P2P-TV Systems

Abstract

P2P-TV systems distribute live streaming contents by organizing the information flow in small chunks that are exchanged among peers. Different strategies can be implemented at the peers to select the chunk to distribute and the destination neighboring peer. Recent work showed that a good strategy consists in selecting the latest received chunk and a random neighboring peer (latest useful chunk, random peer). In this paper, leveraging on the idea that it is convenient to favor those peers that can contribute the most to the chunk distribution, we propose to select the destination peer with a probability proportional to the peer upload bandwidth.

Considering the overlay topology, we evaluate both systems in which nodes have fixed degree and systems whose overlay setup takes into account the actual peer bandwidth by assigning more neighbors to peer with higher bandwidth. We show that the proposed scheme has a limited sensitivity to cheating peers that maliciously declare higher bandwidth than they actually have.

We evaluate the performance in terms of delay percentiles and loss probability and evaluate the achieved improvements. Simulation results considering scenarios with up to 10,000 peers shows that the proposed schemes significantly outperform the traditional ones, so that the chunk distribution delay drops to less than 2s from about 12s.

1. Introduction and Related Works

Peer-to-peer represents a viable and effective approach for the large scale distribution of video contents over the Internet.

Several P2P video streaming systems, P2P-TV for short, such as PP-Stream [1], PPLive [2], SOPCast [3] and TVants [4], just to mention the most popular ones, offering video contents at moderate bit-rate (200-300 kbit/s), have already attracted several hundred thousand users all over the world. A second generation of P2P-TV systems such as Joost [5], Babelgum [6], Zattoo [7], TVUnetworks [8], currently at an advanced stage of prototyping and beta-testing,

are targeted to distribute large bandwidth video streams (1-5 Mbit/s) to large population of users. Video can be either encoded in real time following a “live” approach (so that time constraints are stricter), or it can be delivered using a Video-on-Demand approach (so that peers can watch the stream at different times).

In most systems, peers are arranged so to form an highly connected overlay topology. The stream is divided into small segments, called chunks, which are randomly and independently delivered to the peers, exploiting simple fully distributed, swarm-like dissemination mechanisms on top of the overlay topology. This is in contrast with systems like SplitStream [9] where the content is distributed over an overlay composed of one or few static trees.

On the one hand, the inherent scalability and robustness of systems exploiting swarm-like mechanisms make them more suitable for the heterogeneous, dynamic environment of the Internet where peers have different upload and download capacities and may join or leave the system at any moment [10]. On the other hand, the price to pay is to deal with random, hardly predictable performance, that may critically depend on the peer/chunk selection algorithm used for the chunk transmission between any two peers.

These algorithms may be broadly partitioned into two main categories: *push-based* or *pull-based* systems, depending on whether it is the sender or the receiver that does the selection, respectively. *Pull-based* schemes seem more appropriate for download-constrained systems, since the rate of chunk requests adapts to the download capacity of each peer, for instance, allowing peers to request those chunks that are closest to their playback deadline. *Push-based* schemes are more suitable for upload-constrained systems, such as those comprising a massive percentage of peers connected through ADSL, since the dissemination of chunks is then regulated by the sender, as a function of its own upload capacity.

While several studies have analyzed the performance of *pull-based* systems, only very few works have recently explored the performance of *push-based* mechanisms.

In [11, 12, 13], an analysis of the performance of *push-based* mechanisms have been carried out. Three different mechanisms considering (Chunk selection, Peer selection) have been proposed and analyzed: (*random useful chunk*,

most deprived peer); (latest blind chunk, random peer) and (latest useful chunk, random peer). In particular in [13], under some assumptions, the asymptotic optimality (optimality in order sense when the number of users $n \rightarrow \infty$) of the last policy has been proved both in terms of delay and throughput. Although results [11, 12, 13] make an important step in the direction of understanding the fundamental limiting performance of *push-based* P2P-TV systems, their value is mainly theoretical due to their asymptotic nature. Despite its asymptotic optimality, (latest useful chunk, random peer) could potentially result largely suboptimal in realistic scenarios. In particular, notice that no knowledge of the neighbors' upload bandwidth is exploited. In this paper we show how significant performance improvements can be achieved by intelligently exploiting this information during the peer selection. In addition, we also investigate how the information about peers' bandwidth can be exploited during the overlay topology setup. Experimental results obtained by considering overlay with up to 10,000 peers show that the proposed approach leads to improvements up to 80% considering both chunk delivering delay and loss probability.

2 System

In this paper we consider a "live" mesh-based P2P-TV system in which a single source is responsible to encode and seed the content. Peers in the system adopt a swarm-like mechanism to distribute the chunks, using a *push-based* protocol. The information to be transmitted is organized into small chunks each of fixed size L . The system is composed of one source and a set of peers \mathcal{N} , with cardinality $|\mathcal{N}|$: the source generates chunks at rate λ and sends them to its neighbors; the chunk diffusion in the network is then accomplished by the peers themselves.

We are interested to the effect of the chunk and peer selection policy, so that the effect of peer churning can be neglected (i.e., $|\mathcal{N}|$ remains constant during the content diffusion) since peer lifetime is much larger than the chunk distribution time. Let \mathcal{L} be the set of links that connect the peers and $\mathcal{N}_p \subset \mathcal{N}$ the neighbor-list of a given peer p . For any $p \in \mathcal{N}$, we denote by $u(p)$ the upload bandwidth of p ; $u(p)$ is the maximum amount of bits per time unit that a peer can send. We assume that the amount of bits received per time unit by peer p is unbounded; peers send one chunk at a time. Thus, the chunk transfer rates are limited only by the peers upload bandwidth. In this paper we explicitly consider the case of heterogeneous upload bandwidth.

As well-known, the overlay topology can be described by a random graph $G = (\mathcal{N}, \mathcal{L})$ [16]. We shall focus on the class of *Quasi Regular Random Graphs (QRRGs)* in which the arcs are randomly placed. Each peer selects k_0 random neighbors. Links are bidirectional, so that if peer p is cho-

sen by peer s as a neighbor, the neighbor-list of p also includes s , and chunks are exchanged between s and p in both directions. Thus, the neighbor-list of p , \mathcal{N}_p , contains, on average, $\langle k \rangle = 2k_0$ links. In Section 4, we will investigate different ways of creating the neighbor-list.

The results derived in this paper are based on *push-based* diffusion schemes where the transmission of a chunk is initiated by the sender. The scope of our study is limited to schemes in which the chunk to be delivered is chosen before selecting the receiver peer.

Each peer p maintains a sender-collection-list \mathcal{C}_p , that contains the chunks that p can transmit to its neighbors in \mathcal{N}_p ; the list is a *sliding window* of size w that keeps the w -latest received chunks. The window mechanism "forces" the system to slide on time, avoiding that very old chunks are kept; the rationale being that old chunks are likely dropped by the application, due to tight delay constraints of live streaming services. Each peer has also a received-collection-list $\mathcal{C}_p^{(r)}$ that keeps the information of all chunks already received.

Peer p can adopt different strategies to choose the chunk to transmit to its neighbors. The one we use here, that was proved to be one of the most effective, is called *latest first* (or, following [13], *latest blind chunk*). The sender p chooses the chunk with the highest index (which is usually the latest received one) in its sender-collection-list \mathcal{C}_p . Then, it chooses a destination peer in its neighborhood \mathcal{N}_p . Different policies are possible for the peer selection. These policies may use the information on whether the destination peer has already received the chunk or not; the latter cases are referred to as *blind* strategies. Blind policies require less signaling overhead, since the sender collection list \mathcal{C}_p must not be continuously updated to reflect the status of neighbors' missing chunks. The main focus of this paper is the proposal of a new peer selection strategy. Our scheme, that is described in the next section, is not blind, and will also make a smart use of the window mechanism.

3 Bandwidth-Aware Peer Selection

3.1. Description

The basic idea of *Bandwidth-Aware (BA)* is to select the peer with a probability proportional to its upload bandwidth. Intuitively, *BA* tends to first deliver the chunk to peers that, due to their high upload bandwidth, can contribute more to the chunk diffusion.

According to the *BA* strategy, peers that have the largest upload bandwidth are favored: the probability $\pi_p(n_i)$ that p chooses neighbor n_i , $\forall i \in \mathcal{N}_p$, is proportional to the upload bandwidth $u(n_i)$. Moreover, in order to avoid useless transmissions of duplicate chunks, peer p uses the information

given by the received-collection-lists $\mathcal{C}_{n_i}^{(r)}$, of all n_i . Assume that the latest chunk in the list of p is j ; p chooses neighbor n_i in the set of its neighbors that do *not* have j , namely $\mathcal{N}_p^{(j)} \subset \mathcal{N}_p$:

$$\mathcal{N}_p^{(j)} = \{d | (d \in \mathcal{N}_p) \wedge (j \notin \mathcal{C}_d^{(r)})\}.$$

Then, the probability that p chooses n_i to deliver j is:

$$\pi_p(n_i) = \frac{u(n_i)}{U_p}; \quad \text{with } U_p = \sum_{d \in \mathcal{N}_p^{(j)}} u(d). \quad (1)$$

Once n_i is selected, p immediately starts transmitting chunk j . After the transmission is completed, i.e., after $L/u(p)$ units of time, a new push phase starts. In case all neighbors already have chunk j , the chunk $(j - 1)$ is considered, and possibly delivered to a neighbor that does not have it yet. The procedure continues until either all chunks in the window of p are delivered to all the neighbors, or a new chunk is received.

For comparison only, in the following we will consider a Random peer selection scheme. The scheme works similarly to the *BA* for what concerns the window mechanism and the chunk selection, but neighbor peers that do not have the chunk are chosen with the same probability, i.e., the probabilities in (1) are substituted by:

$$\pi_p(n_i) = \frac{1}{|\mathcal{N}_p^{(j)}|}.$$

3.2. Simulation Results

Implementation assumptions

First, it is worth describing some key issues related to the implementation of the *BA* strategy. First, we consider only the chunk transmissions, assuming that peers have a perfect knowledge of the lists $\mathcal{C}_p^{(r)}$ of their neighbors. This is achieved by signaling messages that are continuously exchanged among neighbors. A discussion on the required signaling bandwidth will be presented later. Second, the chunk diffusion is *no preemptive*. If a peer p receives a chunk with higher index than the one that it is sending to a neighbor n_i , the new chunk is scheduled to be delivered only after finishing the current upload. Third, the cost of delivering any chunk is given by only the *transmission time* $L/u(p)$ in each link. At last the window mechanism always discards the chunk with the lowest index (among the w -latest chunks).

Simulation setting

An event-driven simulator was developed to evaluate the performance of the proposed scheduling policy. A simulation is organized in two phases. First, a random topology is generated according to the desired graph specification and system parameters. During the second phase, the chunk distribution dynamics are simulated until all nodes have completed the content download.

Chunks are generated periodically by the source at a rate of $\lambda = 1$ chunk/s, chunk size is constant and equal to 0.3 Mb ($L = 37kB$), the source upload bandwidth is 1Mbps. All peers have infinite download and finite upload bandwidth, i.e., the bottleneck is the uplink access bandwidth. Based on their upload bandwidth, peers are partitioned into three classes: peers in class \mathcal{P}_1 have upload bandwidth uniformly extracted in $[0.18, 0.22]$ Mbps, peers in \mathcal{P}_2 have $[0.3, 0.4]$ Mbps and peers in \mathcal{P}_3 have $[3, 4]$ Mbps. These rates are set in accordance with several measurements performed using the PPLive, one of the most popular P2P-TV system [14].

In the following, the main performance indices we consider are

- **Delays.** Chunk delay distribution, i.e., the time since the chunk is generated at the source up to when it is received at the peers, is derived at each peer; mean values and percentiles of these distributions are then considered. In particular, since chunks received with large delays are equivalent to losses and losses may have a strong impact on video quality, much attention is devoted to the distribution of the 99-*th* delay percentiles over all peers.
- **Losses.** For each peer, the number of lost chunks (i.e., chunks that are never received) is evaluated. Again, the distribution and mean value of losses over all peers are considered.
- **Improvement** of scheme A over scheme B, considering a given performance index, is defined in the following way: let X and Y be the measured index for scheme A and B, respectively; the improvement I is given by $I = (Y - X)/Y$.

Reference scenario

We start by considering a reference scenario that consists of a *QRRG* with 1,000 peers (including the source) and an average node degree $\langle k \rangle = 8$ ($k_0 = 4$). 270 peers are in class \mathcal{P}_1 ; 660 in \mathcal{P}_2 and 69 in \mathcal{P}_3 .

The scenario is simulated considering 25 different randomly chosen *QRRGs*. For each of them, simulation results are obtained by averaging over 50 independent simulation runs. The video content consists of 200 chunks¹. The peer

¹Longer video has been simulated, leading to the same conclusions.

window size is equal to 5.

BA Performance

In this section we compare the performance of the *BA* scheduling strategy against the Random strategy.

For comparison purposes, we select two overlays out of 25, namely overlays *A* and *B* which respectively reached the worst and the best improvements, considering the mean value of the 99–th *delay percentile* distribution.

Fig.1 shows the cumulative distribution function over all peers of the 99–th *delay percentile* for overlays *A* and *B* under the two considered scheduling schemes. The system enjoys significant performance gain by applying a peer selection that takes into account the knowledge of neighbors' upload bandwidth. Consider *overlay B* (best improvement). The dramatic delay improvement (up to 50%) can be explained based on how the neighbor-lists are assigned to peers. From the analysis of the graph structure, it results that peers with the largest upload rate are either directly connected to the source or placed a few hops apart of it. Just for a better picture, the first jump of the curve represents the delay of roughly 28 peers that are connected to the source by paths with 1, 2 or 3 hops of \mathcal{P}_3 peers. So, the chunks can be diffused among them in a delay less than 2s. This peer configuration leads to the phenomenon of a *virtual clusterization*: chunks are more likely to be first diffused among the fastest peers that create a sort of fast backbone, these peers then effectively distribute the chunks to the slowest peers. *BA* policy then maximizes the utilization of \mathcal{P}_3 peers' upload capacity compared to the Random Selection policy. On the contrary, in *overlay A*, no \mathcal{P}_3 peer is connected directly to the source, so that higher delay is suffered. Also in this case, *BA* policy outperforms Random Selection policy (18% of improvements). (This behavior gives us a clue about the performance improvement of establishing a real clusterization, during the overlay setup; we have preliminary simulation results that encourage further studies on this topic.)

We now examine more closely the peer behavior under the *BA* policy. Let \mathcal{N}_S be the set of peers that are neighbors of the source *S*. Consider a peer $j \in \mathcal{P}_3$ and $j \in \mathcal{N}_S$. Fig.2 compares the delay of each chunk delivered to *j* under the Random and *BA* scheduling policies. As expected, since *BA* favors \mathcal{P}_3 peers with respect to other peer classes, the chunk delay of *j* is smaller in *BA* than in the Random case. Small variations of the delay are given by the presence of another \mathcal{P}_3 peer into \mathcal{N}_S and the rare cases in which peers with smaller upload rates are chosen first. The kind of periodical pattern is due to the limited source upload bandwidth that allows for two or three chunks to be uploaded during the chunk intergeneration time $T_s = 1/\lambda$, this translates into

We prefer to show results obtained considering 200 chunks so that the assumption of no peer churning can be realistic.

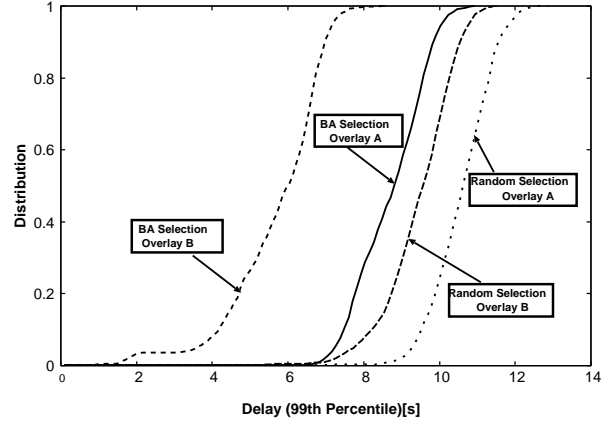


Figure 1. Distribution of the 99-th Delay Percentile for overlays *A* (worst enhancement) and *B* (best enhancement).

alternating large or small delays in starting transmitting the following chunk. Under the Random selection policy, the highest values of delay occur when the scheduling mechanism is, by chance, unfavorable and some additional delay is introduced by the latest chunk selection policy. Slow peers in \mathcal{N}_S , not shown here for the sake of brevity, perceive larger delays under the *BA* than in the Random case. To give the intuition of the better resource utilization that *BA* policy achieves, the total number of chunks uploaded by peer *j* is 66% larger than considering the Random policy.

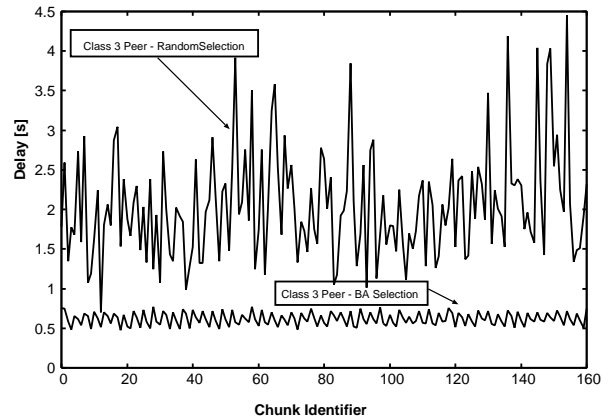


Figure 2. Chunk delays for a \mathcal{P}_3 peer connected to the source of the P2P system.

Let us now consider the total number of lost chunks, reported in Fig. 3. Under the *BA* policy, losses drastically decrease. Indeed, by better exploring the system resources, the *BA* guarantees a faster diffusion of chunks that diminishes the possibility of discarding chunks. The number of

peers with no losses at all increases by a factor 2 in *overlay A* and by a factor of 3 in *overlay B*. Interestingly, in *overlay A*, the peer that undergoes the largest number of losses is the same for both scheduling policies, and the number of losses itself is about the same (around 85): this means that main responsible for this bad performance is a specific very unlucky position of the peer in the topology. For *overlay B*, the *BA* makes the maximum number of losses drop by 47%.

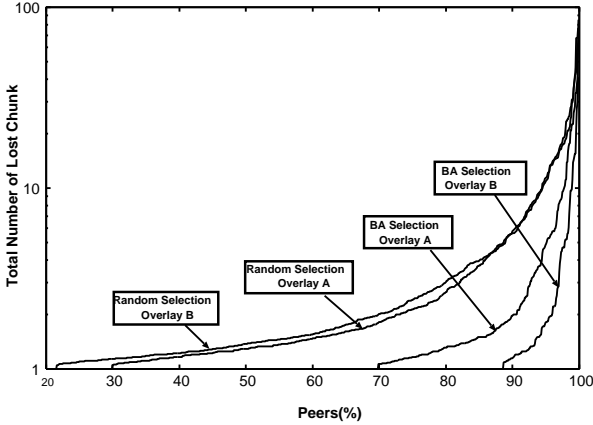


Figure 3. Log scale plot of the total number of lost chunks seen by the peers.

At last, we present the performance improvement reached by using the *BA* policy, considering all the 25 simulated overlays. Figs. 4 and 5 show the improvement on the mean delay and losses, respectively, sorted in increasing order for easy of visualization. The *BA* policy consistently reduced the average delay and losses, achieving improvements up to 40%. It is worth noticing that the enhancement is not completely correlated. A particular overlay that has the best delay improvement not necessarily has the smallest number of losses also.

Impact of graph degree

To gauge the impact of the graph degree in the *BA* policy, Figs. 6 and 7 report the improvements in terms of mean delay and losses when the average node degree $\langle k \rangle$ changes from 8 to 20 ($k_0 = 10$). It is not surprising that increasing the degree, delay and losses decrease. The high connectivity makes \mathcal{P}_3 peers exploit at best their upload capacity. On the contrary, low values of $\langle k \rangle$ limit the upload bandwidth utilization. However, an important cost of increasing connectivity is related to signaling traffic, ignored in previous presented simulation results, as well as in the simulation results reported in [13].

Peers have to exchange with neighbors several control messages to update the buffer map of the chunk collections

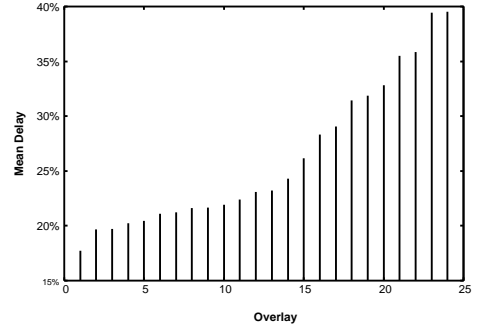


Figure 4. Mean delay improvement of the *BA* policy over the Random Selection policy for each overlay.

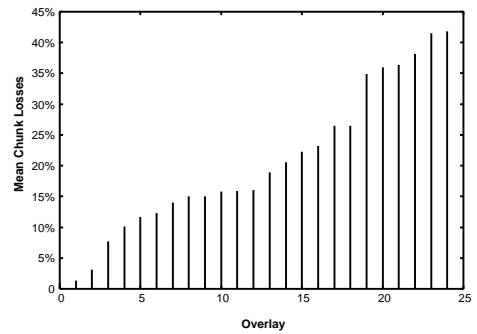


Figure 5. Chunk losses improvement of the *BA* policy over the Random Selection policy for each overlay.

owned by the peers[15], to propagate information about their neighborhood and its possible changes due to churning, or any other possible network information. Signaling traffic is roughly proportional to node degree. Being it not marginal, it should be accounted for as a limitation to the connectivity increase.

For a first estimation of the possible signaling traffic overhead, we refer to recent measurements performed over the PPLive system that show that the mean signaling traffic exchanged between any two peers is approximately equal to 5Kbps [14]. Fig. 8 reports the total rate needed at peers for the maintenance of *overlay A*. In case of $\langle k \rangle = 20$ half of the bandwidth of peers \mathcal{P}_1 must be devoted to signaling.

Impact of window size

In this section we discuss the impact of window size on the *BA* policy. Let us consider that each peer has a window size $w = 10$ (it was $w = 5$ in previous results). We expect that the window size have little impact on delay

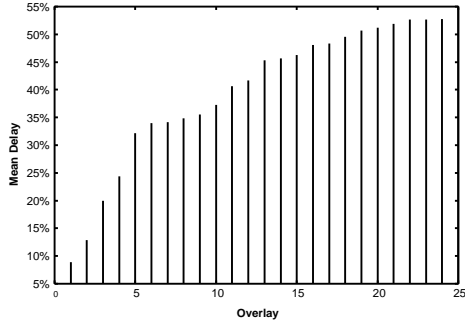


Figure 6. Mean delay improvement in the *BA* policy when the average node degree increases from 8 to 20.

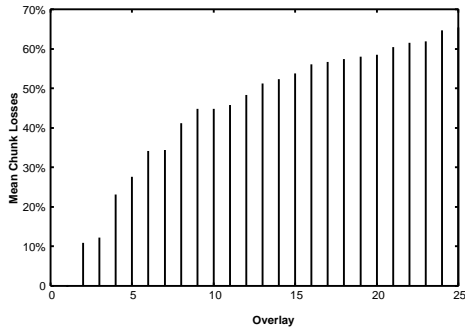


Figure 7. Mean losses improvement in the *BA* policy when the average node degree increases from 8 to 20.

but it can reduce the occurrence of losses. Fig. 9 reports the results. Larger window sizes give higher chances to chunk to be delivered, e.g., improvements by factors of 1.2 and 2 considering topologies A and B respectively.

Impact of the presence of malicious peers

The *BA* policy requires each peer p to tell each neighbor its upload rate $u(p)$. This mechanism poses a potential issue of trust among peers. We thus study the presence of *malicious* (or untrusted) peers, and evaluate the impact of these peers on the *BA* performance.

We suppose that \mathcal{P}_1 peers may be malicious and propagate false values of $u(p)$, pretending that it is the same as the one of \mathcal{P}_3 peers. We consider cases in which the amount of *malicious* peers varies among 10%, 50% and 100% of \mathcal{P}_1 peers. While these rates are very high and a little unrealistic, they allow us to stress the scheduling strategy and show its robustness in extreme scenarios. Figs. 10 and 11 report the results, considering the previous *overlays A* and *B*.

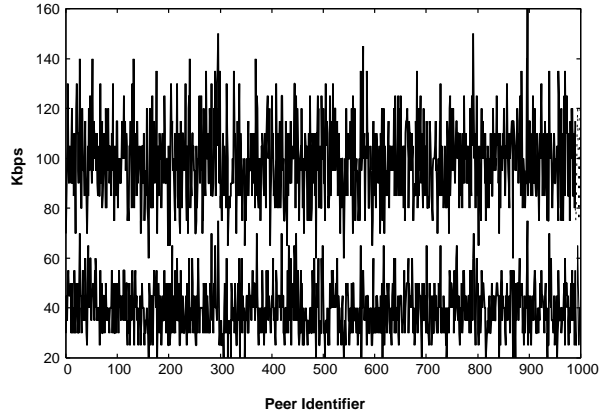


Figure 8. Signaling overhead, for each $p \in \mathcal{N}$ when average node degree is equal to 20 and 8, respectively.

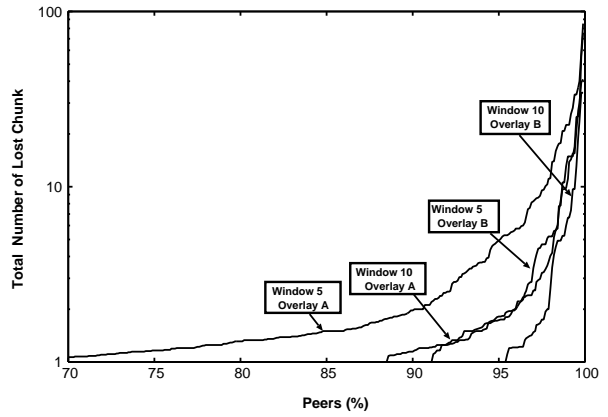


Figure 9. Log scale plot of the total number of lost chunks seen by the peers.

As expected, the presence of *malicious* peers on the overlay degrades the *BA* performance. Observe the case with 100% of malicious \mathcal{P}_1 peers: the *BA* performance almost turns to the Random peer selection one; indeed, the *BA* randomly chooses between small peers with the same declared bandwidth. The remaining small improvement is given by the presence of \mathcal{P}_2 peers. Similar conclusions are drawn from Fig. 11.

Let us now examine more closely the behavior of a \mathcal{P}_1 peer, using the *overlay B* and 50% of \mathcal{P}_1 cheating peer scenario. Let us first consider a *malicious* peer p that is neighbor of the source. Fig. 12 shows the impact of cheating on the delay of p and on the delay of a *honest* peer that belongs to \mathcal{P}_3 and to \mathcal{N}_S . Curves **A** and **D** show the *BA* performance for a system with no *malicious* peers; as expected, peer p perceives much larger delay than the considered \mathcal{P}_3 peer. In curves **B** and **C**, that refer to the scenario

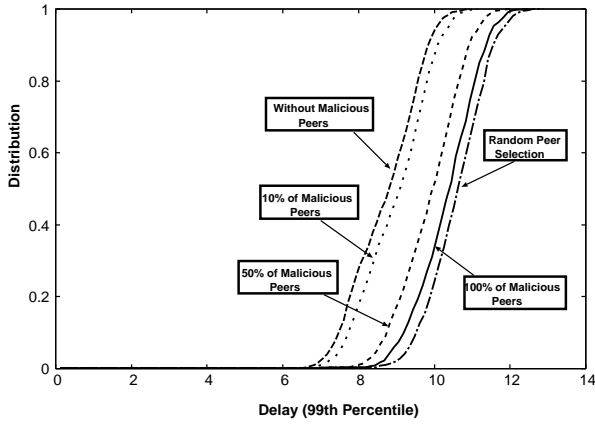


Figure 10. The impact of the presence of malicious peers in class \mathcal{P}_1 for the overlay A.

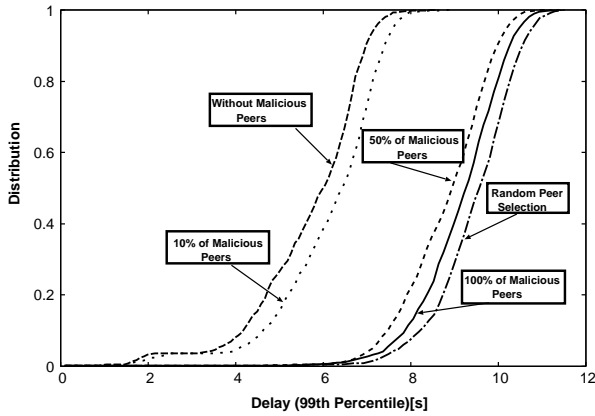


Figure 11. The impact of the presence of malicious peers in class \mathcal{P}_1 for the overlay B.

with malicious peers, we can see that, by *cheating*, peer p drops its own delay and increases the delay of the trusted peer. The delay increase for peers that could effectively deliver the chunk to other peers translates into larger overall system performance degradation. The propagation of the performance *degradation* can be seen on Fig. 13, in which we consider a *malicious* peer far away from the source; the system performance degradation is such that this cheating peer perceives worse performance than in the case with no cheating peers at all. Clearly, a similar but less significant performance degradation is observable also in the case with 10% class \mathcal{P}_1 cheating peers.

Finally, the presented results are representative of *BA* robustness also in case of erroneous declaration of bandwidth. Assume that, in order to propagate information about its upload bandwidth, a peer performs some measurement of the link capacity. Some peers may, not intentionally, return in-

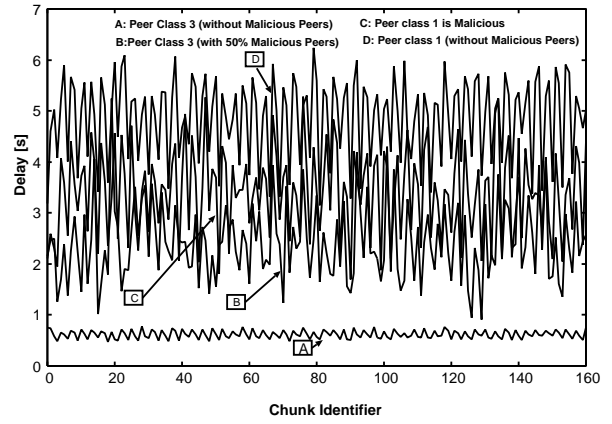


Figure 12. Impact on the chunk delay when 50% of peers are malicious (overlay B); considering one \mathcal{P}_1 peer and one \mathcal{P}_3 peer that are neighbors of the source.

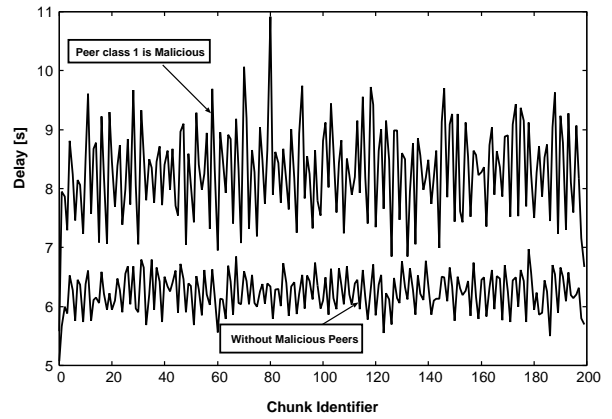


Figure 13. Impact on the chunk delay when 50% of peers are malicious (overlay B); considering one \mathcal{P}_1 peer apart of the source.

accurate values. The scenario would be similar to the one analyzed above.

4. Impact of Overlay Topology

4.1. Description

Considering the overlay structure, any peer p selects at random k_0 neighbors with constant k_0 regardless of $u(p)$. It is not surprising that the system may waste valuable resources (upload bandwidth) during the chunk diffusion process if k_0 is small.

Table 1 gives some numerical results to corroborate this statement by reporting the upload bandwidth utilization for

some \mathcal{P}_3 peers; results were collected from our previous simulations in *overlay B*. A natural and straightforward modification in the overlay setup is to assign to any peer p a neighborhood that takes into account the actual $u(p)$. Once more, it is important to emphasize the trade-off between the neighborhood cardinality and the signaling overhead generated by exchanging information among peers.

Table 1. Upload bandwidth utilization, for a subset of \mathcal{P}_3 peers.

Peer	Random Peer	BA ($k_0 = 4$)	BA ($k_0 = 10$)
1	6.8%	9.7%	49.7%
2	6.0%	10%	76.3%
3	7.32%	14%	47.5%
4	10.4%	24.2%	21.3%
5	16.7%	28.3%	27.8%

We propose a new scheme to build the overlay in a more efficient way which is called *Variable Neighborhood (VN)* overlay construction.

Denote by ϕ_p the neighborhood size of peer p . Let T_p be the time p needs to deliver a chunk, $T_p = L/u(p)$. T_s is the time interval for generating chunks at the source. In order to select the neighborhood size of p , we establish the following constraint:

$$\phi_p T_p \leq c T_s; \forall p \in \mathcal{N},$$

with c an constant.

In words, we are coupling the time to deliver a chunk with the chunk generation time interval. The value of ϕ_p should be large enough to make the overlay structure have large degree and to allow that the upload bandwidth of p is almost always used. If $c = 1$, the bandwidth of p is potentially fully used to deliver the chunks to all the neighbors; the total needed time is $\phi_p T_p$. However, since often some neighbors already have received the chunk through an alternative path, a setting $c > 1$ is preferable. We choose $c = 2$, which guarantees that the bandwidth of p is always used, even if only half of its neighbors do not have the chunk.

4.2. Simulation Results

Reference scenario

In what follows, we consider a reference scenario that consists of a *QRRG* graph with 10,000 peers. The peers are divided into classes with the same partition as before. We also keep the same values for the upload bandwidths.

We analyze the performance of the proposed *VN* overlay approach in a range of 20 randomly chosen *QRRG*s graphs.

For each of them, simulation results are obtained by averaging over 15 independent simulation runs. The video content consists of 500 chunks. The peer widow size is equal to 5.

The first set of results compares the Random Selection and the *BA* policies performance considering an overlay with constant k_0 , referred to as *Fixed Neighborhood* overlay (FN), and the *VN* overlay.

VN Overlay Performance

Fig.14 reports the cumulative distribution function of the 99–th *delay percentile* for *overlays A* and *B* (as usual, selected with the worst and the best enhancements), respectively. Random Selection policy is used. As expected, *VN* overlay provides a very significant delay improvement. Note that, as for the previously noted in Fig 1, the performance of the chunk diffusion process is closely related with the overlay construction.

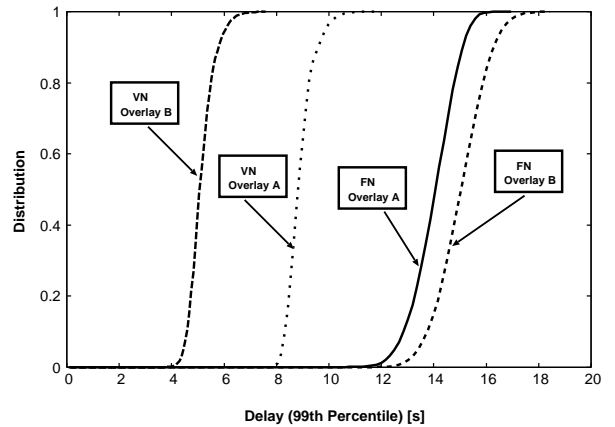


Figure 14. Distribution of the 99–th delay percentile for overlays A (worst enhancement) and B (best enhancement). Comparison of Fixed Neighborhood and Variable Neighborhood topologies. Random Selection policy case.

Let us briefly introduce the results regarding the set of overlays. Considering the mean delay, the *VN* topology guarantees improvements between 35% and 65%. The performance enhancement also extends to the mean losses, where the *VN* topology improves performance in a range from 25% to 60%.

Fig. 15 shows that coupling the Bandwidth-Aware selection policy with the Variable Neighborhood topology large improvements can be achieved. Indeed, the *BA* policy reduces the delay of about 50% with respect to the Random Selection policy also in case a *VN* topology is considered. The total improvement reached by adopting *BA* and *VN* mechanisms is 85% (i.e., delay drops to less than 2s from

about 12s)!

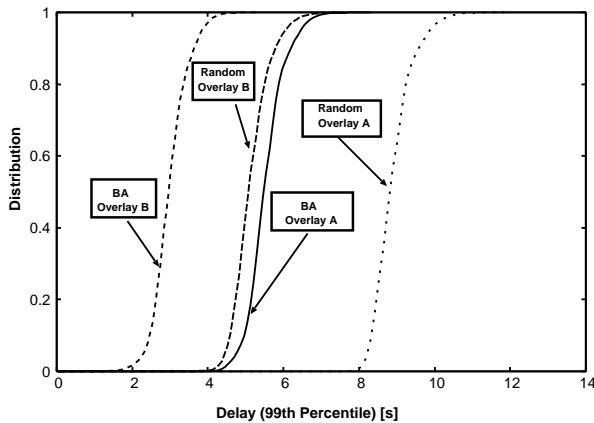


Figure 15. Distribution of the 99–th delay percentile for overlays A (worst enhancement) and B (best enhancement), using Variable Neighborhood topology and BA policy case.

In what follows, we show the signaling overhead by applying the VN overlay approach. Coupling the upload bandwidth to the neighborhood size permits at the same time to limit the impact of signaling on nodes with limited upload bandwidth and to fully exploit the upload bandwidth of nodes with larger bandwidth. Fig. 16 shows the signaling overhead considering a subset of 2000 peers. It clearly shows that peers with large bandwidth (all of them have peer identifier larger than 9000) must manage a large signaling overhead, while slow peers do not have to deal with (useless) large amount of signaling.

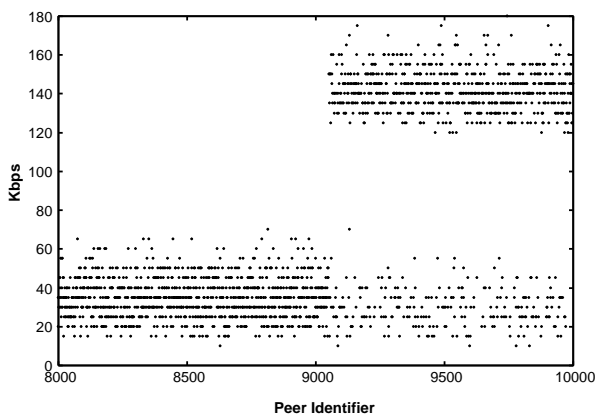


Figure 16. Signaling overhead for the VN overlay B.

5. Conclusions

In this paper, we have explicitly considered heterogeneous scenarios in which peers with different characteristics coexist. By leveraging on this heterogeneity, performance of “live” P2P-TV systems can be significantly improved by intelligently exploiting the information on the peers’ upload bandwidth. This information can be capitalized both at level of overlay topology setup and chunk distribution.

A new scheduling algorithm along with a scheme for the overlay topology setup have been proposed. Simulation results in scenarios with up to 10,000 peers have shown that the proposed schemes significantly outperform the traditional ones, so that the chunk distribution delay drops to less than 2s from about 12s.

References

- [1] PPStream, <http://www.PPStream.com>.
- [2] PPLive, <http://www.pplive.com>.
- [3] SOPCast, <http://www.sopcast.com>.
- [4] TVAnts, <http://www.tvants.com>.
- [5] Joost, <http://www.joost.com>.
- [6] Babelgum, <http://www.babelgum.com>.
- [7] Zattoo, <http://www.zattoo.com>.
- [8] Tvunetworks, <http://www.tvunetworks.com>
- [9] M. Castro, P. Druschel, A.-M. Kermarrec, A. Nandi, A. Rowstron, and A. Singh, “Splitstream: High-bandwidth multicast in cooperative environments”. *Symposium on Operating System principles (SOSP 2003)*, Bolton Landing, NY, October 2003.
- [10] N. Magharei, R. Rejaie, and Y. Guo. “ Mesh or multiple-tree: A comparative study of live p2p streaming approaches” *INFOCOM, 2007*, Anchorage, AK, May 2007.
- [11] L. Massoulie, A. Twigg, C. Gkantsidis, and P. Rodriguez. “Randomized decentralized broadcasting algorithms” *INFOCOM, 2007*, Anchorage, AK, May 2007.
- [12] S. Sanghavi, B. Hajek, and L. Massoulie, “Gossiping with multiple messages”, *INFOCOM, 2007*, Anchorage, AK, May 2007.
- [13] T. Bonald, L. Massouli, F. Mathieu, D. Perino, A, Twigg, “Epidemic Live Streaming: Optimal Performance Trade-Offs”, *Sigmetrics 2008*, Annapolis, ML, June 2008

- [14] Technical Report, reference omitted to preserve authors' anonymity.
- [15] X. Hei, Y. Liu, K.W. Ross, "Inferring Network-Wide Quality in P2P Live Streaming Systems," *IEEE JSAC, special issue on P2P Streaming*, December 2007.
- [16] Béla Bollobás. "Random Graphs", Cambridge University Press, 2001.